Models of interactive systems: a case study on Programmable User Modelling

Ann Blandford,
UCL Interaction Centre,
UCL,
26, Bedford Way,
London, WC1H 0AP, U.K.
A.Blandford@ucl.ac.uk

Richard Butterworth and Paul Curzon
Interaction Design Centre,
School of Computing Science,
Middlesex University,
Bramley Road,
London, N14 4YZ, U.K.
{R.J.Butterworth, P.Curzon}@mdx.ac.uk

Accepted for publication in IJHCS. This is a pre-print version of the paper.
Full citation:

BLANDFORD, A., BUTTERWORTH, R. & CURZON, P. (2004) Models of interactive systems: a case study on Programmable User Modelling. *International Journal of Human–Computer Studies*. 60.2. 165-216.

Models of interactive systems: a case study on Programmable User Modelling

Ann Blandford,
UCL Interaction Centre,
UCL,
26, Bedford Way,
London, WC1H 0AP, U.K.
A.Blandford@ucl.ac.uk

Richard Butterworth and Paul Curzon
Interaction Design Centre,
School of Computing Science,
Middlesex University,
Bramley Road,
London, N14 4YZ, U.K.
{R.J.Butterworth, P.Curzon}@mdx.ac.uk

Abstract

Models of interactive systems can be used to answer focused questions about those systems. Making the appropriate choice of modelling technique depends on what questions are being asked. We present two styles of interactive system model and associated verification method. We show how they contrast in terms of tractability, inspectability of assumptions, level of abstraction and reusability of model fragments. These tradeoffs are discussed. We discuss how they can be used as part of an integrated formal approach to the analysis of interactive systems where the different formal techniques focus on specific problems raised by empirical investigations. Explanations resulting from the formal analyses can be validated with respect to the empirical data.

The first modelling style, which we term 'operational', is derived directly from principles of rationality that constrain which user behaviours are modelled. Modelling involves laying out user knowledge of the system and task, and their goals, then applying the principles to reason about the space of rational behaviours. This style supports reasoning about user knowledge and the consequences of particular knowledge in terms of likely behaviours. It is well suited to reasoning about interactions where user knowledge is a key to successful interaction. Such models can readily be implemented as computer programs; one such implementation is presented here.

Models of the second style, 'abstract', are derived from the operational models and thus retain important aspects of rationality. As a result of the simplification, mathematical proof about selected properties of the interactive system, such as safety properties, can be tractably applied to these models. This style is well suited to cases where the user adopts particular strategies that can be represented succinctly within the model.

We demonstrate the application of the two styles for understanding a reported phenomenon, using a case study on electronic diaries.

1. Authors are listed alphabetically; all have contributed substantially to the paper.

1. Introduction

There have been many attempts to characterise the 'discipline' of HCI, in terms of both the object of study and the legitimate methods that might be applied in that study. The object of study might be cognition, the computer system, the work system, the interaction, etc.; the method might be derived from the scientific, design, mathematical or social sciences tradition. One important strand within these many traditions is that of developing models of users and of interactive systems for explanation and prediction. In this paper, we present two such models, comparing and contrasting their properties and the kinds of results they can deliver.

There is a long tradition of modelling in HCI; any particular modelling technique has features that make it well suited to addressing particular user-oriented questions about a design or interaction (Carroll, 2003). For example, Cognitive Complexity Theory (CCT: Kieras and Polson, 1985) can be used to reason about ease of use of a device in terms of the number and nature of rules a user has to know to be able to operate it effectively. Similarly, TAG (Payne and Green, 1986) supports reasoning about consistency of operations in terms of user knowledge of how to perform 'similar' operations. John and Kieras (1996) present the GOMS family of models, including a summary of which technique is best suited to answer particular usability-related questions. The work of John and Kieras is unusual in explicitly scoping and comparing models, albeit within a fairly tightly knit 'family' of models. More recently, as discussed further below, two distinct traditions of modelling have emerged: cognitive modelling has tended towards more complex models that have higher empirical validity, typically based on a cognitive architecture such as ACT-R (Anderson, 1993), while mathematical modelling has focused on the development of mathematical techniques that support reasoning about interactive behaviour, encapsulating more limited aspects of cognition but supporting stronger abstract reasoning.

For any modelling to be tractable it must (explicitly or implicitly) define its own boundaries. Typically there is a fairly straightforward trade-off to be made over where such boundaries are drawn: narrow boundaries lead to simple, maybe elegant, investigations, with limited external validity; broad boundaries to complex, 'messy', time consuming investigations with higher external validity. Modelling is a process whereby explicit boundaries for an investigation are set and a real world referent is abstracted over and represented in some abstract notation or language. The process of abstraction is fundamental to the process of modelling; an 'accurate' model of a referent (accurate in the sense that every possible feature of the referent is included in the model) is an oxymoron. Here, we investigate some of the smallest, most abstract models of interactive systems possible, to see whether they can be used to reason about interactive systems and, if so, what questions they can answer.

In summary, any model supports reasoning about particular aspects of the design of an interactive system, and one essential challenge is to select a modelling approach that will answer interesting questions about a design. In this paper, we present two models of the same interactive system, each of which supports reasoning about particular aspects of that design:

- A simulation model, in the tradition of Programmable User Modelling (PUM: Young, Green and Simon, 1989; Blandford and Young, 1996), supports reasoning about user knowledge needs for each of two alternative strategies for achieving the same goal, and yields predictions about likely user behaviours. Below, we refer to this as an 'operational' model.
- An abstract mathematical model supports reasoning about a safety property (i.e. that the goal state is actually reached), and hence the reliability of each of the two behavioural strategies identified.

One common feature of both is that they model a simple interactive system, consisting of one user and one device, and that they treat user and device together: the operational model brings together a user model and a device model to reason about their conjoint behaviour, while the other model abstracts away from the details of user and device to reason only about interactive behaviour. The two styles of modelling can be compared in terms of their scope and their explanatory power.

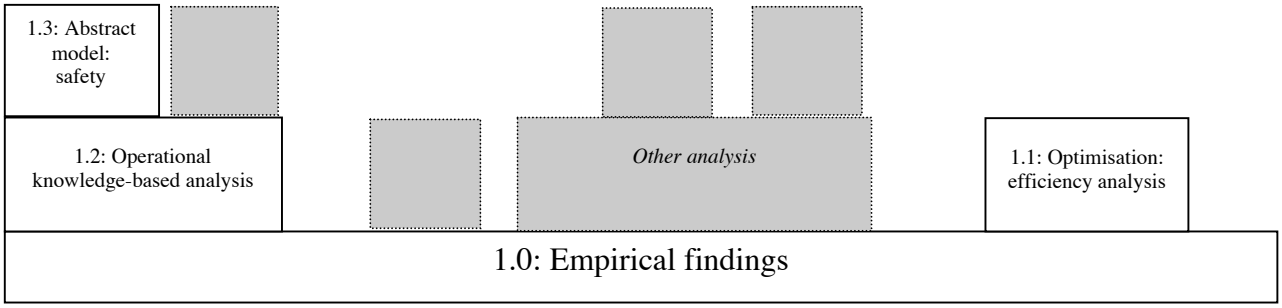

Figure 1: the shape of the reasoning (shaded areas in principle possible but not done)

Broadly, the shape of this work is as follows (schematically illustrated in Figure 1): an empirical study was conducted in which a range of usability concerns relating to a particular device – in this case, an electronic diary – were identified (1.0). These included issues relating to roles, power relationships, group culture, screen layout, etc.: these are discussed informally but comprehensively by Blandford and Green (2001). Modelling takes a particular aspect of that general picture and probes it in more detail. In principle, a variety of different more detailed analyses can be conducted; in the work presented here, we conduct three further analyses: an efficiency analysis (1.1) that supports reasoning about optimal behaviour and two analyses based on user knowledge – an operational analysis, (1.2) that analyses how users exploit their knowledge and an abstract analysis (1.3) that conducts a safety proof, establishing whether or not the user assumptions made lead to a prediction of user errors. It retains important aspects of rationality derived from the first. These analyses are amenable to different degrees of verification, but all need to be validated against empirical findings to test to what extent their findings match real-world data. The strengths and weaknesses of each kind of modelling are discussed below.

2. Background and related work

The work reported here brings together cognitive and mathematical modelling in HCI. Both are well developed traditions. For example, cognitive modelling work has been established in the traditions of GOMS, CCT and TAG, as well as approaches that are more grounded in the larger context within which the user is working, such as the Cognitive Reliability and Error Analysis Method (Hollnagel, 1998) and Cognitive Work Analysis (Vicente, 1999). In terms of detailed modelling, researchers such as Gray (e.g. Gray, 2000) and Byrne (e.g. Byrne, 2001) have constructed models based on an established cognitive architecture to better understand user behaviour when working with particular interactive devices, and Salvucci and Lee (2003) have investigated integrating GOMS models with a detailed cognitive architecture (in their case, ACT-R). In these cases, the concern is with constructing a detailed model that well matches empirical data, and the resulting models are necessarily complex; the focus is on cognition rather than interaction.

Mathematical modelling of interactive systems has tended to focus on device design – for example, in the work of Palanque and Bastide (1995), Paterno' (1993), Thimbleby, Cairns and Jones (2001) and Harrison and Thimbleby (1990). Relatively little work has been done involving formal modelling of users: examples include the work of Moher and Dirda (1995), who propose a Petri net model that supports reasoning about users' changing mental models and their changing

expectations as they work with a device, and of Rushby (1999), who applies model checking to model a pilot's knowledge of aircraft systems behaviour alongside a corresponding model of actual behaviour, to reason about safety-critical divergences between the two. Work such as that of Doherty, Campos and Harrison (2000) and Duke, Barnard, Duce and May (1998) has explicitly incorporated a user model component within a mathematical model of an interactive system. The work of Doherty *et al* exemplifies an approach that focuses on the perceptual element of cognition relative to a particular task; the question the model answers is: does the interface display provide support to users in addressing their domain goals? In contrast, the work of Duke *et al* does not consider domain tasks, but focuses on the channels of communication and resources of user and device. Therefore, it supports reasoning about compatibility of the agents (users and computers) at a detailed level; this is particularly relevant for multi-modal interactions and those involving novel input/output devices whose usability properties are poorly understood.

The work reported here shares features in common with each of these approaches: the models are mathematical, and incorporate explicit representations of users and computers. They also support reasoning about various aspects of user knowledge and interactive system behaviour, as discussed below.

The operational models presented here have been instantiated in a Lisp implementation, to generate simulations of behaviour. Work by others (e.g. Fields, 2001; Abowd, Wang and Monk, 1995) has shown that model checking can also be used for reasoning about interactive behaviour – though in both of the cases cited, the focus is on modelling overt behaviours, rather than any underlying user cognition. Ongoing work (Curzon and Blandford, 2001, 2002) is investigating the strengths and limitations of machine assisted proof for reasoning about rational user behaviour, using the HOL machine assisted proof system. However, in this paper, we focus on the principles underpinning the model rather than the details of any particular implementation. In the remainder of this section, we relate the work reported here to issues within the broad area of cognitive and mathematical modelling.

2.1 A brief history of PUM

The work reported here is the product of reflecting on what we have learnt about PUM over several years of development and investigation. The original vision for PUM was presented by Young, Green and Simon (1989), in which they argued that it should be possible to develop a user model that embodied a simple model of human problem solving and common sense knowledge, that could be 'programmed' with knowledge to establish how it would perform when interacting with a device that demanded that knowledge. Essentially, this approach looks at the role that user knowledge plays in interaction. A PUM analysis initially lays out a description of the knowledge a user might be expected to have to interact with a proposed device. This knowledge can then be analysed in several ways: is that knowledge adequate for the user to successfully interact under all circumstances? If not, why not, and how can the system be redesigned so that successful interaction is better supported? Is it reasonable to expect the user to have the necessary knowledge? Where does the necessary knowledge come from (training, the display, etc.)? How will the user behave given this knowledge, and is that as the designer intended?

With this initial view, the main value of modelling would derive from running the model and comparing the resulting behaviour with that which was expected. However, at this time, no implementation of a PUM existed.

Over the next few years, the focus remained on developing running models – initially based on the Soar architecture (Newell, 1990). The approach taken was to develop an Instruction Language that represented the required user knowledge succinctly, which could be compiled into Soar productions to generate a Soar model (Blandford and Young, 1993) that implemented 'mini-planning' – i.e.

localised planning based on means–ends reasoning. A very similar approach has recently been reported by Salvucci and Lee (2003), converting GOMS descriptions into ACT-R. As an approach to fundamental cognitive science, such an approach is promising; however, as an approach to delivering usable modelling in HCI, it had the clear shortcoming that models could only be realistically run, debugged and modified by analysts with a working knowledge of the Soar architecture. In addition, at that time, linking a Soar model to any external representation (e.g. of a device) was disproportionately difficult and time-consuming. To overcome these perceived limitations, a simpler Programmable Interactive Problem Solver (PIPS) that captured the key elements of means–ends reasoning, and could accommodate a separate device description, was developed and tested (Blandford and Young, 1995).

By this time, various important points about the modelling were becoming clear:

- that the process of laying out the knowledge users needed to work with the device (and particularly any difficulties experienced by the analyst in describing that knowledge) often yielded greater insights than the process of actually running the model – an observation that has also been made about the use of formal methods in software development (Hall, 1990);
- that the process of programming the model sometimes demanded the inclusion of implementational details that could obscure the fundamental principles of the model; and
- that for some systems, that could be characterised as involving repetitive activity with different (but similar) data objects, the development and running of an operational model was completely missing the point about the interaction, which was typically whether simple user knowledge could be applied repetitively to data objects – whether systematically or randomly – to achieve a given goal. For such systems, it appeared that a more abstract style of modelling would be more appropriate than the operational PUM, but based on the same fundamental principles.

Effort was consequently focused on developing principles and on abstraction – on simplifying as far as possible, to investigate what leverage could be gained through avoiding complexity. The principles are presented below; they have not previously been published. Various abstract models of particular systems have been developed (e.g. Butterworth, Blandford and Duke, 1999; 2000).

The operational models of electronic diaries presented here, including the implementation based on the principles, have not been previously published. The abstract model presented here is largely reproduced from Butterworth, Blandford and Duke (2000), and readers are referred to that paper for the full mathematical proof that is described in outline below (section 6.5). The main contribution of this paper is not in the individual models, but in the comparison and scoping of those models, and a simple presentation of the principles underpinning the modelling.

The main features of the approach are: that it focuses on a 'middle' level – the knowledge level (Newell, 1982) – rather than (for instance) the biological or social level; that it combines a re-usable framework with device-specific information to construct a particular model; and that the modelling is as simple as possible. Each of these features are discussed in the remainder of this section.

2.2 Levels of description

Newell (1990; p. 122) outlines a series of levels at which human actions can be studied or analysed, organised into four bands: the biological, cognitive, rational and social. This idea is developed by Barnard, May, Duke and Duce (2000), who relate the same levels to computer systems and to interactions within social and technological systems, developing theories of how interactions take place within and across levels. While HCI has had little cause to concern itself with the biological band, work has taken place at all the higher levels, yielding different kinds of insights about

human–computer interaction. For example, work in the cognitive tradition is exemplified by papers in a special edition of this journal (Ritter and Young, 2001) on cognitive modelling that focuses on 'embodied' models, including the addition of simulated eyes and hands to enable rich interactions (at a fine grain of detail) with simulated devices.

At the other extreme, work on social aspects of computing pays attention to larger-scale phenomena such as patterns of interaction between people and devices, the resources used within the workspace (Fields, Wright and Harrison, 2000; Hollan, Hutchins and Kirsh, 2000), and social phenomena such as changes in roles and relationships as activities are restructured around new technologies (e.g. Adams and Blandford, 2002). As with different kinds of models, attention paid at different levels yields different insights into the qualities of interactions between users and devices and makes it possible to answer different kinds of questions.

The work reported here falls at a 'middle' level: it is concerned with how users apply their knowledge in interactions with devices to develop patterns of behaviour with particular (desirable or otherwise) properties. The questions that this kind of approach can answer are therefore those that pertain to achievement (or not) of user goals, sources of human error and efficiency of interactions. While these are not the only questions that matter when considering usability, they are contributing factors to overall usability, and in this paper we use the term 'usability' as a short-hand for these particular issues, all of which fall within Newell's (1990) 'rational' band.

2.3 Frameworks and models

One approach to studying rationality within the context of cognition is the development of cognitive architectures. Gray, Young and Kirschenbaum (1997) assert that "cognitive architectures provide the most important new contribution to a theoretical basis for HCI", arguing that a complete architecture will ensure the development of consistent models over a range of behavioural phenomena. They distinguish between a model, which represents one particular situation, and an architecture, which captures the generalities – the cognitive phenomena that manifest themselves in a range of different situations. In the work reported here, we make an analogous distinction: between a particular model and the underlying framework. The framework encapsulates a generalised collection of guidelines for constructing a model. These guidelines encode findings about rational behaviour derived from the cognitive science literature (e.g. Newell, 1990; Pollock, 1993), so that any model developed from that framework inherits the same rationality principles. By working at the rational, rather than the cognitive, level, we develop models that are simpler, and therefore more amenable to formal reasoning.

Simply instantiating a framework generates an operational model. Instantiating a framework and abstracting over the resulting model yields an abstract model. In this paper, we follow that process through: propose a framework, instantiate this framework with a specific case study to get an operational model, abstract over this to get an abstract model, and finally compare and contrast the roles played by these two types of models.

2.4 Minimalist modelling

Recognising that an important question is where to draw the boundary for a model (what is useful information to include in a model and what can be abstracted over), we have taken a minimalist approach. Recognising that cognitive models can rapidly become very complicated, and therefore difficult to reason with, our approach has been to investigate the smallest, most abstract models of interactive systems possible, to see whether they can be used to reason about interactive systems and, if so, what questions they can answer. These models do not model learning, probabilistic reasoning (e.g. prospect theory: Tversky and Kahneman, 1992), bounded rationality (Simon, 1987),

decision making under uncertainty (Klein, 1999) or many other key aspects of cognitive psychology. The approach presented here could, in principle, be extended to support reasoning about any or all of these features – in which case, the modelling would demand much more of the analyst and the benefits of simplicity would be sacrificed for the benefits of accuracy. One of the questions of the approach being adopted is what leverage one can get with simple models.

For the same reason, other aspects of users, such as individual differences due to variations in culture, cognitive abilities, etc. are not modelled. As illustrated below, it is possible to model users who have different knowledge; it would equally be possible to model users who have different basic conceptions of the task, or different understandings of the device and how it works. In this way, it would be possible to develop a diverse set of models for different users. Equally, it would be possible – though arguably no longer a PUM, in terms of intellectual tradition – to replace the principles presented below with others that reflected a different view of human interactive problem solving. Any analytical approach to evaluating interactive systems is based on assumptions about the users, which may be more or less explicit. This is true, for example, of GOMS (John and Kieras, 1996) and Cognitive Walkthrough (Wharton, Lewis, Rieman and Polson, 1994), which are closest in tradition to the approach presented here. One of the distinguishing features of PUM as presented in this paper is that the assumptions are explicit, inspectable, and hence adaptable by anyone who has a reason and a theoretical basis on which to adapt them – e.g. to model more diverse users.

Focusing on abstract models, Butterworth, Blandford and Duke (1998) demonstrate a very abstract model of interactive system behaviour and show that it can highlight potential problems with a device design, and that the process of proving behavioural properties with the model gives a fairly clear indication of where the problem with the device design lies. If appropriate, other methods could then be used to more deeply analyse and remedy the problem.

There are two complementary assumptions we can make when compiling very abstract models. In earlier work we modelled users with very limited capabilities; if such a modelled user was able to interact successfully, then we had an argument that real users would also be able to successfully interact (Young, Green and Simon, 1989). Conversely, we could make stronger assumptions about users and their capabilities than in reality, but show that an interactive system model still predicts that such a user will make an error; we then have an argument that real users are likely to make that error too. This is the line of reasoning used in this paper.

3. A framework for modelling interactive systems

Our first step is to lay out a framework for interactive systems that incorporates principles that describe how a rational user will interact with an interactive device. As discussed above, this framework is based on findings in the cognitive science literature and refined from operational models that have been developed and tested over several years.

We present here a general framework. For clarity, the presentation is simplified as far as possible. In particular, in common with most work on mathematical and cognitive modelling:

- the system consists of only one user and one device (we use ‘system’ to refer to the total interactive system, and ‘device’ to refer to the computer component),
- the user employs the device as a tool to achieve a certain goal and we do not address the issue of how the user acquires new goals (either through the interaction or from any external source),
- the device is treated as being ‘passive’, i.e. its behaviour is fully determined by input from the user.

In order to lay out this framework we start by describing what 'behaviour' is. We discuss how the device should be modelled and lay out the building blocks for the user side of the framework by looking at the different types of knowledge needed for the framework and what roles they play. We look at how the display is modelled before bringing all the components together into a framework for modelling interactive system behaviour.

Note that there is a difference between the device and user sides of this framework. The device side makes the smallest possible set of assumptions about the device; it is effectively just some guidelines about how to express a device model so that it can be combined easily with the user side. The user side, however, includes a collection of assumptions which are fixed. One of the overarching aims of this work is to provide a framework in which devices can be designed and then placed in the context of a user model to predict the usability of the system (Young, Green and Simon, 1989). In this sense the user cannot be 'designed' in the same way as the device can.

3.1 Modelling behaviour

Behaviour describes how a system's state develops through time; typically, we model it as a simple sequence of system states. The model we use is inductive, meaning that it defines a collection of possible initial states and a collection of transitions that describe how the system can change state. Given a sequence of states, we can say that it is legal for a certain model if it starts in one of the defined initial states and every subsequent state change in the behaviour is described by one of the transitions.

If a model has only one legal behaviour then it is deterministic. Such models are useful when we want to automate them so that they generate simulated behaviour, but enforcing determinism on a model tends to restrict it considerably. In general, we reason about the **space** of behaviours that are legal for a model which gives much more flexibility in analysis; one focus of this paper concerns how much of this space of behaviours can be easily reasoned about with certain models.

3.2 Modelling the device

A device specification consists of a description of the device state, a set of legal initial states and a set of legal 'actions' which describe ways in which the device state can be updated. This approach to modelling devices is intentionally not novel; we have chosen a modelling style with a thoroughly researched and understood mathematical basis. It is based on work such as that of Lamport (1994) and Pnueli (1992), who describe inductive specification techniques. Furthermore the idea of an 'action' meshes well with the user model we present below.

3.3 Modelling the user

Central to the user side of the framework is the idea of rationality. Rational behaviour is a function of the user's knowledge and goals. Given a certain collection of knowledge about the state of the world, how to change the state of the world and a desired goal state, a rational user will interact in order to move the state of the world nearer to the desired goal state (Pollock, 1993). There are two types of knowledge which we express: knowledge about the state of the world, which we refer to as 'beliefs' and knowledge about how to change the state of the world, which we call 'operations'.

A belief expresses what the user believes to be true. There is no stipulation that beliefs should be correct, accurate or complete; indeed many beliefs are not. Typically the user's beliefs about the state of the device are of particular interest to us when modelling interactive systems, but other kinds of belief will obviously come into play – for example, more domain oriented beliefs about the task and environment.

Learning new beliefs is not simply a case of adding newly learnt beliefs to the current belief state as this may introduce contradictions. Here we make the simplifying assumption that recently learnt information takes priority over old information. Therefore if a new belief is learnt that does not contradict existing beliefs it is simply added to the belief state, but if it does contradict existing belief then it replaces that existing belief (Gärdenfors, 1988; Ryan, 1992); for example, if there is only one cursor in a program and it is at position Y then it is reasonable for the user to replace any pre-existing belief that it is at X with one that it is at Y (where $X \neq Y$). In the Lisp implementation presented below, we deal with belief revision by distinguishing between 'relations' and 'properties': a predicate, such as that encoding cursor position, whose value is updated is encoded as a 'property', whereas a predicate that can take multiple values simultaneously (so that predicates are added to or deleted from the system state) is encoded as a 'relation'.

Operations represent the user's beliefs about how he can change the state of the world. For interactive systems, operations typically describe the user's knowledge of the how and why of device actions; typically a device action represents how the device state is updated, and there is a corresponding operation describing how the user believes that action affects the device state, why the user would wish to invoke that action, and under what circumstances. There is no requirement that there be a one-to-one relationship between actions and operations. For example the user of a word processor may have two operations: to delete text and to move text to the paste buffer, but they may both be carried out by the single device action 'cut'.

An operation is represented by a collection of beliefs about actions and effects, as summarised in Table 1. As noted in the table, beliefs about the action and its purpose are essential for any operation; a particular operation may optionally include beliefs about preconditions, filters and tracking.

Element	Description
purpose	The purpose of an operation describes why an operation will be selected. It is a belief about what goal it is good for addressing.
precondition *(optional)*	The precondition of an operation describes what the user needs to believe is true before the operation can be performed.
filter *(optional)*	The filter of an operation describes what the user needs to believe is true before the operation can be committed to.
tracking *(optional)*	The tracking of an operation describes what effects the user believes that the operation will have.
action	The invoked action of an operation is (typically) the device action which needs to be invoked for the operation to be performed.

Table 1. A summary of the elements that make up an operation.

An operation's purpose is a belief about what goal the operation is good for achieving. A rational user will select an operation to perform if its purpose at least partially matches an outstanding goal and there is no reason not to select it.

An operation may have a precondition, a filter, or both. These describe under what circumstances the user will perform the operation. The difference between the two is subtle but important. A precondition describes what the user needs to believe to be true before the operation can be performed. In contrast, a filter describes what needs to be true before an operation will be committed to. The user may work to make a precondition true, but not to make a filter true. Given

an operation that is rational to perform (i.e. its purpose matches the goal) but for which the preconditions are not satisfied, the user will 'commit' to performing that operation and adopt its unsatisfied preconditions as new goals. A filter expresses beliefs about circumstances (which the user will not try to change) in which it is rational to select a particular operation to achieve a desired effect.

The tracking of an operation describes what the user believes the effect of performing an operation is. One characteristic of expert use is that operations are performed and some of their predictable effects known without the user having to look at the display – though, as Payne (1991) points out, much interaction is typically display-based, and users keep track of very little in their heads. The tracking describes the beliefs the user has about the effect that an operation has (typically on the state of the device) without looking at the display (or anywhere else) to verify this. For example, the user may track the belief that the operation 'move text to paste buffer' adds the currently selected text to the paste buffer (the user must track this belief, as it is not readily visible from the display).

3.4 Modelling the display

The display is treated as a shared resource between the user and device (Fields, Wright and Harrison, 2000; Butterworth, Blandford and Duke, 1999). The device renders information onto the screen and the user interprets that information as a belief. Within their model of interactive devices, Duke and Harrison (1993) describe a rendering function which takes the internal state of the device to an external, visible state. In the model presented here, we take the simple approach of representing what is displayed and assuming that the user perceives and correctly interprets displayed information.

3.5 A framework for deriving behaviour

We can now bring together the elements of the framework for modelling devices and users and combine them into a framework for modelling interaction. Users are the primary motivating agents in the interaction; they are assumed to be rational and goal driven. The main aim of the framework is to lay down rules that describe how users will do this, i.e. we are making the assumptions underlying the phrases 'rational' and 'goal driven' explicit and inspectable.

The system state

The overall system state is subdivided into three areas: the device state, the user state and the display state. The device state is simply the composition of all the variables in the device specification. The user state is described by:

- the user's current beliefs about the state of the world,
- a goal (expressed as a belief) which describes the user's desired belief state,
- a collection of 'commitments', being the operations that the user has selected to perform, but has not yet done owing to their preconditions not being satisfied, and
- a collection of 'sub-goals', being the preconditions of all the operations in the set of commitments.

The display state is represented by a belief; i.e. the display is modelled as the user's interpretation of what is on the screen.

System behaviour

System behaviour is defined by a collection of principles which describe how the system state can be updated and under what circumstances. As discussed above, these principles are based on Newell's (1990) principles of rationality which have been developed specifically so as to be applicable to a given class of interactive systems. The principles rely on the adjectives **rational**, **possible**, and **motivated** and the verbs **commit**, **drop** and **perform** to which we assign strict definitions and which we place in bold to distinguish them from their more informal meanings.

The principles of behaviour are shown in Figure 2. Definitions of the adjectives and verbs are given in Figures 3 and 4 respectively. Figure 5 shows the conditions for a legal initial state.

Principle of goal driven behaviour: The user interacts in order to achieve his overall goal. A goal is said to be achieved when the user's belief state implies the goal.

Principle of immediate performance: If there is an operation that is **rational** and **possible** then it may be immediately **perform**ed.

Principle of commitment: If there is an operation that is **rational**, but not **possible** then it may be **commit**ted to.

Principle of commitment performance: If there is a commitment that has already been made which has now become **possible**, then one of two things can happen:

- if the operation is still **motivated** then it may be **perform**ed and the commitment to it **drop**ped, or
- if the operation is now not **motivated** then the commitment to it may be **drop**ped *without* the operation being **perform**ed.

Principle of belief updating: The user maintains beliefs about the state of the system by interpreting the display state (see also the definition of performance in Figure 4).

Figure 2: The principles of behaviour

Three of the principles describe legal transitions in the system state. The exceptions are the "principle of goal driven behaviour", which describes under what circumstances the system can cease interacting and the "principle of belief updating", which states how the user maintains beliefs from the display state. A behaviour is considered to be legal according to the principles if each step in the behaviour can be described by one of the principles and the behaviour ceases with the system in a state described by the principle of goal driven behaviour. One possible implementation – in this case, in Lisp – of the principles is presented in Appendix A.

Rational: An operation is **rational** if all the following conditions hold:

- there is a goal which is either the ultimate goal or from the set of sub-goals which would be at least partially fulfilled by the purpose of the operation, so long as…
- there are no goals (either the ultimate goal or one of the sub-goals) which are made permanently unachievable by the purpose of the operation,
- the filtering condition of the operation is satisfied. i.e. the current belief state implies the filtering condition.

Possible: An operation is **possible** if its preconditions and filters are satisfied, i.e. the current belief state implies the operation's preconditions and filters.

Motivated: An operation is **motivated** if its purpose is not already satisfied, i.e. the current belief state does not imply the operation's purpose, but it has been committed to and it is rational.

Figure 3: Definitions of adjectives

As discussed above, what is rational in any given context depends on the user knowledge in that context. In particular, whether a given operation takes the user nearer the goal depends on what the user knows about other operations that might take him even closer towards the goal (e.g. by constructing a plan involving more than one operation).

Commit to: An operation is **commit**ted to by adding it to the set of commitments and its precondition to the set of sub-goals.

Drop: An operation is **drop**ped by removing it from the set of commitments and removing its preconditions from the set of sub-goals.

Perform: An operation is **perform**ed by updating the device state with the operation's invoked action and by the operation's tracking being learnt (i.e. added to the current belief state without introducing contradictions.)

Figure 4: Definitions of verbs

An interactive system must start in a state such that:

- the device state is legal according to the device specification,
- the user has a goal and
- the sets of commitments and sub-goals are empty.

Figure 5. Legal initial states for an interactive system

This framework is both inspectable and reusable. It is also consistent with previous work on rational behaviour, as discussed above. We have outlined our reasons for keeping the set of principles as simple as possible, but others could adapt them to capture more complex features of normal interactive behaviour, such as resource-bounded decision making.

4. Case study

In the previous section we proposed a framework for modelling interactive systems. In this section we instantiate that generalised model to a specific example: an electronic diary system. The process of instantiation is effectively about 'filling in the blanks' in the general model – for example, by defining what beliefs the modelled user has and what operations are available. Instantiating the general model in this way gives us an operational model. We then look at two strategies for achieving an example task and draw out the usability consequences for these two strategies and the impact that the device design has on them. As an illustrative case study, this focuses on one particular issue; in Section 7 we draw out lesson learnt from this and other case studies on applying PUM using both styles of modelling illustrated here.

4.1 The diary system

For the purpose of this study, we consider a particular, commercially available, diary system – Meeting Maker (ON Technology, 1995). We had already conducted an informal usability analysis that had raised several important points that merited further investigation. Diaries and their use have been extensively studied; for example, Kelley and Chapanis (1982), Kincaid, Dupont and Kaye (1985), Payne (1993) and Palen (1999) report on studies of diary use, and draw on those studies to propose recommendations for the design of new diary systems. In particular, Blandford and Green (2001) present results of a qualitative study of Meeting Maker, considering the role of the electronic diary within its social context of use, and also as one of a suite of time-management tools (including paper diaries and many other resources) that need to work well together. In that paper, we consider many aspects of usability; here, the focus is on particular features that are amenable to the modelling approach that is the subject of this paper.

Meeting Maker implements a 'frequency' feature which was originally proposed by Kincaid *et al* (1985) and implemented in the VS diary (Beard *et al*, 1990). This feature allows users to set up a series of events at regular intervals by defining a 'base' event, an interval (e.g. 'weekly' or 'the third Tuesday of every month') and an optional 'end' date. The inclusion of such a feature is one of the benefits of electronic tools highlighted by Palen (1999). This example was chosen for two important but distinct reasons:

- There are two common strategies for entering multiple events that can be compared: entering events one by one, or using the frequency feature. In our study of diary use (Blandford and Green, 2001), it was found that only four of the sixteen interviewees used the 'frequency' feature confidently and effectively; the other twelve reported never using it. Two of these reported simply remembering regular events or keeping a separate record of them, while the remaining ten chose to enter each event in the series separately. Four of these ten were either not aware that the 'frequency' feature existed, or believed that it was inadequate (e.g. that it does not allow the user to express frequencies such as 'the third Tuesday of the month' – which is does permit, but as a 'weekly' frequency, not a 'monthly' one), while the others expressed a measure of distrust in their ability to use it without making errors. It appeared that a better understanding of the use of this feature could be gained by constructing mathematical representations of it, so that these informal user claims could be investigated further.
- Previous work had led us to the conclusion that operational models are most appropriate in situations where user knowledge is the key determinant of usability (e.g. Blandford and Young, 1996), and abstract models are most appropriate where heuristic rules appear to guide behaviour (e.g. Butterworth, Blandford and Duke, 1999). This example shares features of both, depending on both user knowledge and patterns of behaviour, so that it could be examined from both perspectives.

To construct a model, we need to work with particular scenarios of use. Here, we propose one such scenario for use of the diary. Suppose that, in a certain organisation, information about upcoming seminars and events is collated by a secretary and then disseminated amongst the staff as a paper circular. It is up to the individual members of staff to enter these events into their diaries. These events are characterised by being almost, but not quite, regular. For example seminars typically occur on Wednesdays at 3pm in the meeting room, but some weeks the time and venue may change, there may not be seminars on certain weeks and so on. A sample paper diary is shown in figure 6.

Day	Date	Time	Event	Room
Weds	20th Jan	3-4pm	Seminar	Meeting room 1
Weds	27th Jan	4-5pm	Seminar	Meeting room 2
Weds	10th Feb	3-4pm	Seminar	Meeting room 1
Weds	17th Feb	3-4pm	Seminar	Meeting room 1
Weds	24th Feb	3-4pm	Seminar	Meeting room 1
Weds	3rd Mar	3.30-5pm	Management meeting	Boardroom
Weds	10th Mar	3-4pm	Seminar	Meeting room 1
Weds	24th Mar	3-4pm	Seminar	Meeting room 1
Weds	31st Mar	3-4pm	Seminar	Meeting room 1

Figure 6. A sample paper diary to be entered into Meeting Maker.

Meeting Maker displays a diary on screen (see figure 7), typically showing the current week. The diary can be scrolled from week to week using a scroll bar and arrow buttons in the normal way.

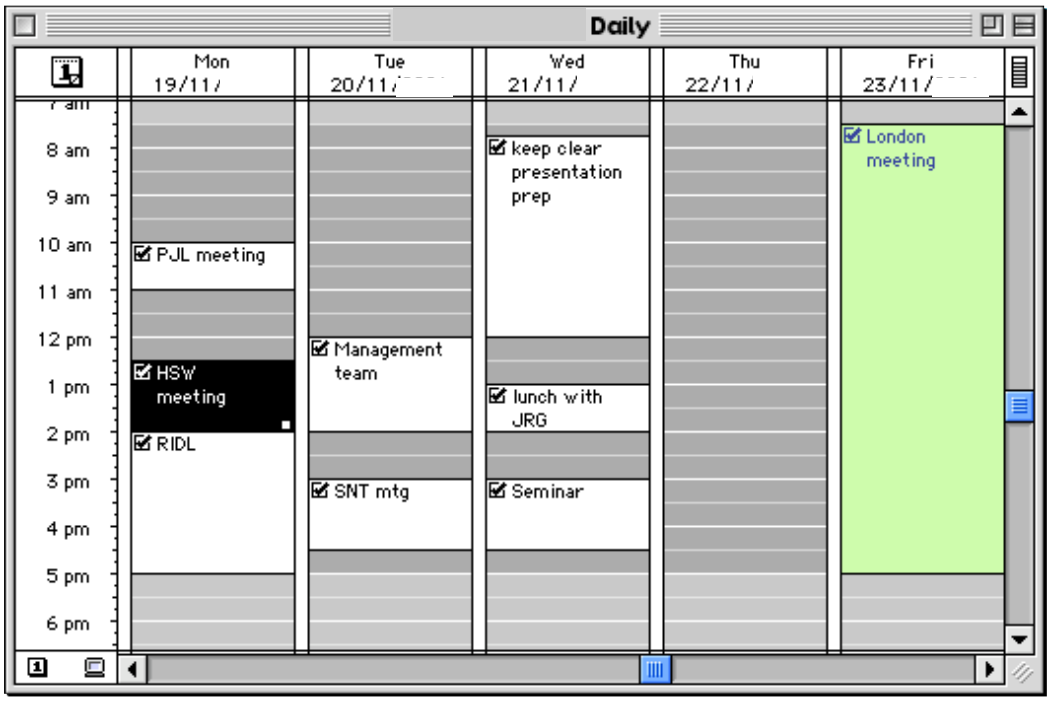

Figure 7. A typical week's diary displayed by Meeting Maker

To enter an event into the electronic diary the user clicks on the start time of the event. This brings up a data entry form into which the user types the information concerning that event. Once the user confirms the data to be correct the event is entered into the diary and a representation of the event is displayed on the screen.

A user can also enter a series of events. The user clicks on the time of the base event in the series which brings up the data entry form as before, but the user can also select a frequency for that event and an end date for the series. Once confirmed, a series of events which occur at regular intervals

are entered into the electronic diary. These events share all the other attributes, such as duration and title.

The two strategies we look at for achieving the task correspond to these two methods for entering events into the diary. The first strategy, known as the 'one by one' strategy, involves the user going through each individual event on the paper diary and entering it correspondingly into Meeting Maker. The second strategy, known as the 'en masse' strategy, involves the user entering a series of events at one go which will approximately correspond to those in the paper diary and then going through those newly entered events and correcting them if needs be.

4.2 Efficiency analysis

A very simple analysis of the two strategies shows that in many situations it is 'better' to use the en masse strategy. Consider the paper diary in Figure 6; there are nine events to enter, of which two are unusual, and there are two 'gaps' in the series. It takes roughly the same amount of effort (in terms of mouse clicks and key presses) to enter an event or delete an event as it does to correct an event, and it takes about twice as much effort to enter a series of events. For sake of argument we call the unit of effort to enter, delete or correct an event E, so the unit of effort to enter a series of events is $2E$. In general, if there are e events, of which g are non-standard or superfluous (i.e. need to be changed or deleted from a series), the effort of following the one by one strategy is eE, and that of the en masse strategy is $(2 + g)E$. Based on this general analysis it is only better to use the one by one strategy in a few situations. In the particular scenario under consideration, the one by one strategy requires $9E$ of effort and the en masse strategy requires $(2 + 4)E$, so the en masse is more efficient. However, as discussed above, many users choose to use the one by one strategy. Thus, if one equates rationality with optimisation, real users' behaviour is generally not rational. However, as highlighted below, *knowledge-based* rationality as defined in this paper supports reasoning about more limited behaviours than this.

This simple analysis is similar in spirit to that done by Card, Moran and Newell (1983), in which they compare the effort needed to correct a document using a text editor with that required to type it out from scratch. Their model deals with surface effects – the time to press keys and set up typewriters – and goes into much greater mathematical detail. However, the general style of analysing efficiency, and using that as a measure of which approach is better, is analogous to this informal analysis.

So, to summarise: we have a system we wish to model, a task which users perform with that system and two strategies for achieving that task. Informally the question we wish to ask about the system using our models is 'which strategy is better?' Our model-based analyses address knowledge requirements and proneness to error; these are alternatives to efficiency as measures of which approach might be 'better'. We can now compare the process and results of modelling our example by performing both styles of modelling.

5. Operational models

We first construct an operational model of the system and scenario, based on the framework presented above. We then use this model to reason about user behaviour for each of the two strategies. In the following section (6), we use the operational model as a starting point for constructing an abstract model.

5.1 An operational model of Meeting Maker

To develop an operational model of Meeting Maker we first describe the device model, then the beliefs that the user employs in interacting with the device, and finally the operations the user employs.

The device model

The device model consists of descriptions of the device state and the actions that can transform the device state.

The device state consists of a set of events that are recorded in the electronic diary. An event is a record of the details of an entry in the diary; it consists of the time that the event takes place (where 'time' refers to a definitive marker, including date, day, hour, minute, etc), the duration of the event, its title, etc. For the purposes of the model it is not necessary to define precisely what information is stored in an event other than the time at which it takes place. So we declare a set of times *T*, and a set of events *E,* both of undefined type…

$$T : \ldots \tag{1}$$

$$E : \ldots \tag{2}$$

Given an arbitrary event *e* which is of type *E* then *e.time* denotes the time at which *e* takes place.

There is a set of events denoted *ed* (for 'electronic diary') which represent the events stored in the Meeting Maker diary and, similarly, a set of events denoted *pd* for the paper diary.

$$ed : \mathcal{P}(E) \tag{3}$$

$$pd : \mathcal{P}(E) \tag{4}$$

We also need to capture the fact that the device makes a certain portion of the electronic diary visible on the screen. All we need do is declare variables to denote the earliest time visible on the screen and the latest time visible on the screen (the variables are *start* and *end* respectively).

$$start, end : T \tag{5}$$

The device actions describe ways in which the device state can change. There are five actions, as follows:

addEvent(e): The result of invoking the action *addEvent(e)* is that the event *e* is added to the electronic diary *ed.*

scroll(t): The result of invoking action *scroll(t)* is that the time *t* becomes visible in the diary window. i.e. after *scroll(t)* has occurred *start* will be earlier than *t* and *end* will be later than *t.*

correctEvent(e_1, e_2): The result of invoking *correctEvent(e_1, e_2)* is that event e_1 is removed from the electronic diary *ed* and replaced by e_2.

deleteEvent(e): The result of invoking the action *deleteEvent(e)* is that the event *e* is removed from the electronic diary, *ed.*

addSeries(base,interval,finish): The action *addSeries* is a little more complicated. It adds a series of events similar to *base* to the electronic diary, *ed*. The first event it adds is *base* and then it adds events after *base* at a regular interval determined by *interval*. It does not add events after the time *finish*. For example, if *base.time* is "5th Nov", *interval* is weekly, and *finish*

is “3rd Dec” then the result of *addSeries(base,interval,finish)* would be that events like *base* are added to *ed* on 5th Nov, 12th Nov, 19th Nov, 26th Nov and 3rd Dec.

To complete the description of the device model we need to assert which states are valid initial states. A valid initial state is, quite simply, one in which *start* is earlier than *end*. The Lisp implementation of this device model is included in Appendix B.

User beliefs

As discussed above (Section 3.3), user beliefs are collections of non-contradictory facts. Here we describe what facts the user employs when interacting with Meeting Maker. We need facts that express:

- whether a certain event is believed to be in the Meeting Maker diary or not,
- what times the user believes to be visible on the display, and
- that the user believes that two given events are approximately (but not exactly) equal to one another.

The user’s goal

To define the user’s goal we assume that there is a finite collection of events in the paper diary that need to be added to the electronic diary. The user’s goal is therefore that the user believes that all events from *pd* are in the Meeting Maker diary.

5.2 The user’s operations

As discussed in Section 3.3, operations are expressions of how and why the user would invoke the device actions. There are five operations, each one corresponding to one of the five device actions.

The first operation we describe is *newEvent(e)* which is an encoding of the user knowledge relating to the device action *addEvent*. The purpose of performing *newEvent(e)* is that the event *e* is added to the Meeting Maker diary, i.e. the user will commit to performing *newEvent(e)* if he has a goal that includes having *e* in the Meeting Maker diary. The user will not be able to invoke the device action *addEvent(e)* unless the user believes that the time *e.time* is visible on the display. However the user may ‘sub-goal’ on that belief, i.e. if the user wants to perform the operation *newEvent(e)* but cannot because he does not believe that *e.time* is visible he will adopt the sub-goal of changing the device state so that he believes the time *e.time* is visible. The user tracks that the effect of performing *newEvent(e)* is that *e* is added to the Meeting Maker diary; i.e. the act of performing *newEvent(e)* is enough for the user to believe that *e* is in the diary, without having to look at the display to verify the belief. The operation *newEvent(e)* is shown in a semi-formal notation below:

newEvent(e) = (6)

purpose	event *e* is in the diary
precond	time *e.time* is visible
track	event *e* is in the diary
action	action *addEvent(e)* is invoked

The operation *scrollTo(t)* is very straightforward. Its purpose is to make the time *t* visible and it invokes the device action *scroll(t)*. It has no preconditions and so can be performed at any time.

scrollTo(t) = (7)

purpose	time *t* is visible
action	action *scroll(t)* is invoked

The operation *makeCorrection(wrong,right)* encodes the beliefs relating to the device action *correctEvent*. Its purpose is that the event *right* is in the Meeting Maker diary. Its sub-goaling precondition is that the time *wrong.time* is visible. The user tracks that the event *right* is added to the diary and that the event *wrong* is removed from the diary. Its filtering condition (which describes what the user must believe before the operation is committed to) is that the user believes that *wrong* and *right* are approximately the same event, and that *wrong* is already in the Meeting Maker diary.

makeCorrection(wrong,right) = (8)

purpose	event *right* is in the diary
precond	time *wrong.time* is visible
filter	*wrong* is approximately, but not exactly, equal to *right*
track	event *right* is in the diary, event *wrong* is not in the diary
action	action *correctEvent(wrong, right)* is invoked

The operation *delExtra(wrong)* encodes beliefs relating to the device action *deleteEvent*. Its purpose is that the event *wrong* is not in the Meeting Maker diary. Its sub-goaling precondition is that the time *wrong.time* is visible. The user tracks that the event *wrong* is removed from the diary.

delExtra(wrong) = (9)

purpose	event *wrong* is not in the diary
precond	time *wrong.time* is visible
track	event *wrong* is not in the diary
action	action *deleteEvent(wrong)* is invoked

Just as the device action *addSeries* was more complicated than other actions, so its corresponding operation is also more complicated. *newSeries(base, V, f)* is an operation to add the events in *V*, starting with event *base* and finishing on date *f*, into the Meeting Maker diary. Unlike the other operations we have described, *newSeries* may not result in the purpose being fulfilled. Its precondition is that the time of *base*, the base event in the series, is visible. The user tracks that a series of events has been added to the Meeting Maker diary, but that this series may not be exactly that required. The user invokes the device action *addSeries* and calculates the parameters to that device action so that the series *base*,e_2'... e_n' is as similar to the required $V=e_1 ... e_n$ as possible. How the user actually does this is not addressed in this analysis, but it can be a complex mental operation, probably achieved by calculating the modal average of the time intervals in the series $e_1 ... e_n$.

newSeries(base, V, f) = (10)

purpose	events in *V*, which start with *base,* are required to be in the diary
precond	time *base.time* is believed to be visible
track	events in *V'* which is approximately the same as *V* are believed to be in the diary
action	action *addSeries(base,interval,f)* is invoked, where *base* is the base event in the series, *f* is the date of the last event in the series, and *interval* is the modal average of the intervals between events in *V*.

Clearly, there are many alternative ways of expressing this operation, each of which captures the way some individuals might represent this operation. We have chosen an expression that captures a

precise knowledge of the base event in the series, the date of the last one and a less explicit representation of those in between. The very difficulty of expressing this operation highlights the fact that it demands substantially more complex knowledge and reasoning than the earlier operations.

The Lisp implementation of this knowledge is included in Appendix C.

The display

The display shows the currently visible events and times, which is the subset of all events that occur between *start* and *end*. That is: *e* in *E* such that $start \leq e.time \leq end$. In the Lisp implementation, this information is encoded in the function 'visibility-test' (Appendix B).

5.3 Reasoning about the operational model

Now that we have a model of a user that interacts with Meeting Maker we can propose behaviour patterns and see if those behaviours are rational by our definition. To do this we need to establish whether or not each step in the behaviour can be described by one of the principles of behaviour described in the framework. For both strategies, the goal is that the events $e_1 \ldots e_n = pd$ are in the electronic diary, *ed*.

The one by one strategy

As discussed in Section 4.1, the one by one strategy is the simple addition of events from the paper diary into the electronic diary. Clearly, since events can be entered in any order into the diary, for simulation purposes we need to define an order. On reasonable approach is to define a heuristic that determines the order in which the events will be added – for example, starting with the first event and adding events in chronological order. This is the most likely order, since users can then use their knowledge of order to keep track of which events have been added and which remain; Hutchins (1995, p.297) discusses in more detail how users might keep track of which subtasks have been completed and which are outstanding. Here, we present abstract reasoning about the behaviour of the operational model, which can be compared to the simulation run (presented in Appendix D).

Assume that the user has no outstanding commitments or subgoals and does not believe that the first event, e_1, is in the electronic diary. It is therefore rational to select the operation *newEvent*(e_1) because its purpose (that event e_1 is in the electronic diary) partially fulfils the goal of having all the events $e_1 \ldots e_n$ in the electronic diary.

Having selected the *newEvent*(e_1) operation there are two possibilities: that $e_1.time$ is visible and the principle of immediate performance can be invoked, or that $e_1.time$ is not visible and the principle of commitment can be invoked.

If the principle of immediate performance is invoked then the user performs *newEvent*(e_1) straight away. The device action *addEvent*(e_1) is invoked so the event e_1 is added to the diary and the tracking condition of the operation *newEvent*(e_1) ensures that the user believes that this has happened. This is the case modelled in Appendix D.

In the other case the operation *newEvent*(e_1) is committed to; it is added to the set of commitments and its subgoaling precondition – that time $e_1.time$ is believed to be visible – is added to the subgoals. The user now has the subgoal of making $e_1.time$ visible. The purpose of the operation *scrollTo*($e_1.time$) matches this new subgoal and so the user selects it to perform. It has no subgoaling precondition and so the principle of immediate performance can be applied. The device action *scroll*($e_1.time$) is invoked and results in the time $e_1.time$ being visible on the display. The user

can now look at the display and learn that $e_1.time$ is indeed visible. The principle of commitment performance can now be invoked, as the precondition to *newEvent*(e_1) is satisfied and it is still appropriate to perform the operation. From this point onwards in the interaction, it is not rational to re-adopt this subgoal, as it will not take the state nearer to the goal state.

So we have established that it is rational to move from a state where e_1 is not in the diary to a state where e_1 is in the diary, and the user believes it to be there. That part of the goal has been achieved. The same process is applied to the goal again, to add e_2 (seminar on 27th January), and so on until all the events from the paper diary *pd* have been added to the electronic diary *ed*.

Our conclusion is, therefore, based on the user model we have constructed, that there is a rational behaviour whereby the user can perform the one-by-one strategy and successfully add events e_1 … e_n to the electronic diary. Note that we have only actually established this for one possible sequence of actions; we can not conclude from this analysis that a real user following this strategy would always achieve the goal, only that there is a rational way to do so. Exactly the same reasoning can be applied for any other ordering, though in practice – a feature not included in this model – users would probably have to tick off entered items to keep track of their progress toward the goal if events were entered in an arbitrary order.

The en masse strategy

Similarly we can reason about the en masse strategy by describing behaviour that characterises the en masse strategy and arguing its rationality. Again, edited highlights from a trace of running the en masse strategy are included in Appendix D.

Initially let us assume that the user has no commitments, believes that none of the events from e_1 … e_n are in the Meeting Maker diary, but believes that the time of the base event e_1 is visible on the screen. (As we argued above, if that time is not visible the user can simply scroll to it; this is not problematic.) Now we argue that it is rational to select the *newSeries*(e_1, *pd, f)* operation and that because its precondition is already satisfied, by the principle of immediate performance, it can be done straight away. Once it has been done, the user believes that there is a collection of events *E'* in the Meeting Maker diary which are similar to $pd=e_1 \ldots e_n$. Now the user can repeatedly step through each of these events correcting them as necessary.

One of the critical success factors in this case is precisely how the user formulates the task. The user may mentally step through the events that have been entered; this involves mentally tracking every week or using the electronic diary as an external memory aid. However, the user may, alternatively, use the paper diary as an external memory aid, and rely on that as a list of the events that have been entered. We consider this case in detail.

If the user checks the correctness of an arbitrary event e_i, from the paper diary, the result of doing so can be either of the following:

- the user knows that the event e_i is indeed in the diary and no correction needs to be done (i.e. $e_i = e'_i$).
- the event e'_i in the diary is not exactly the same as e_i and therefore needs correcting (i.e. $e'_i \neq e_i$).

In the first case no further action is required of the user and he can either choose a new event to check, or cease interacting if there are no further events to check.

In the second case it is rational to select the operation *makeCorrection*(e'_i, e_i), which the user does, and the user can move on to correcting another event.

Working this through as an operational model: the user has invoked *newSeries(e_1, pd, f)*, where (see Figure 6) e_1 is the seminar on 20th January, *pd* is the contents of the paper diary and *f* is 31st March. The modal average is weekly, so the corresponding device action is *addSeries*(e_1, *weekly, f*), which adds weekly seminars to the diary (all on Wednesdays at 3pm in Meeting Room 1, lasting one hour). The first event, e_1, is visible and known to be correct, so the user considers the next event, e_2. This event needs correcting (to a different meeting room), so the user performs *makeCorrection*(e'_2, e_2). This continues until all nine events (up to 31st March) have been checked and corrected where necessary. At this point, the user believes that the diary entries are all correct. However, inspection of the device state shows that the user has an incorrect belief: there are additional (unwanted) events entered in the diary on 3rd February and 17th March. The strategy together with use of the paper diary as a memory aid has *not* led to a successful outcome. As in the one-by-one strategy, changing the order in which events are corrected yields the same result. The current Lisp implementation (Appendix D) models this behaviour slightly differently: it models a user who knows about the additional events being in the diary (because their inclusion has been tracked as part of the operation of entering a new series) but who does not recognise this difference as being important relative to the desired state.

As noted above, there is an alternative user strategy, which is to use the electronic diary as the external memory aid, rather than relying on the paper one. In this case, there is a third possibility as the user corrects an arbitrary event:

- the event e'_i has no corresponding entry e_i in the paper diary so e'_i needs deleting.

Working through the operational model for this strategy shows that this behaviour leads to a successful outcome. However, as discussed, success depends on whether the user uses the paper or the electronic diary as a memory aid.

In the Lisp implementation of the knowledge of operations and user goal (Appendix C), the user knowledge for the correct en masse strategy is represented by encoding in the user knowledge of the goal state that certain events are *not* in the diary (whereas this knowledge is omitted from the earlier incorrect – but simpler – version).

Comparison of strategies

Overall, we have established that there are two particular procedures for entering events into the diary that yield either the intended result or a high likelihood of an erroneous result. In the latter case, the user would be unaware of the error. In addition, we have shown that the user knowledge needed for applying the one by one strategy is substantially simpler than that needed for applying the en masse strategy – especially for the correct version of the en masse strategy. In particular, for the one by one strategy, the user only ever has to invoke operations *newEvent* and *scrollTo*, whereas for the en masse strategy, he has to invoke *setSeries*, *makeCorrection* and *delExtra* as well as *scrollTo*.

In summary, this approach to operational modelling has highlighted two key differences between the two user strategies being considered: the sophistication of the user knowledge needed and the proneness to error. The en masse strategy may be more efficient, but this is at the price of increased cognitive effort (in calculating the parameters for the 'frequency' operation) and mental workload (in keeping track of which diary entries need correcting). The user also has to *learn* the additional operations to perform the *en masse* strategy. A richer psychological model might identify other factors that also influence the choice of strategy, such as the mental effort and accuracy of assessing which strategy would be more efficient for any given set of diary entries. However, those are outside the scope of this modelling. The concern here has not been to produce a fully

explanatory model, but to illustrate what this particular kind of operational model can and cannot deliver.

Running through the models for each possible rational behaviour would be time-consuming, though clearly this process could be automated for any finite set of events. An alternative is to abstract away from the details. Since our purpose in this paper is to compare models, we now consider this approach.

6. Abstract models

In the previous section we demonstrated how an operational system model can be derived from principles of rational behaviour and showed some of the reasoning that can be applied to such models.

In essence we use operational models to prove the existence of legal behaviours, as well as to analyse user knowledge needs. While this can be useful, we would also like to use models to demonstrate stronger properties over (possibly infinite) classes of behaviour. In other words, as well as being able to test claims such as 'there exist legal behaviours which result in the goal being achieved' or 'there exist legal behaviours that fail to achieve the goal', we would like to test claims such as 'all legal behaviours result in the goal being achieved'. In order to demonstrate these universally quantified claims we need to use more formal proof techniques than we did in the previous section.

In this section we outline how we can use proof by refinement techniques to do this. However the models we proposed in the previous section are too complicated, detailed and unwieldy to sensibly perform such a proof on. Therefore we need to simplify those models.

6.1 Aspects to be abstracted over

When considering the models described in the previous section, two aspects immediately suggest themselves as being unnecessarily complicated:

- what is and is not visible, and
- the distinction between the device state and the user's knowledge of the device state.

Consider the way we had to deal with the user making certain times in the diary visible; the user typically had to subgoal on making the appropriate time in the diary visible before it was possible to enter the appropriate event. In order to make certain areas of the diary visible the user had to use the 'scroll' operation. However the 'scroll' operation has no precondition and so the user can always scroll to the appropriate place. In effect a 'scroll' operation with no precondition makes the whole of the diary eventually, if not immediately, visible. In this analysis we are not concerned about the distinction between eventually and immediately, and therefore we make the simplifying assumption that everything in the diary is visible, so we do not need to worry about scrolling.

Secondly, and more challengingly, in the operational models we made a distinction between the actual device state and the user's beliefs about the device state. In particular we modelled the set of events that are in the diary, and the user's beliefs about what events are in the diary. We now simplify the models so that the device state and the user's beliefs about the device state are conflated. In the abstract model we propose a single variable which represents the set of events in the electronic diary and, by inference, the user's beliefs about which events are in the electronic diary. This is evidently a strong claim to make, as it would appear that we are asserting that the user has perfect knowledge, and so requires justification. Using the abstract model we will show that the modelled user is liable to make mistakes using the en-masse strategy. Our argument is that the

model suggests users will make mistakes *in spite of* having perfect knowledge, and therefore real users with imperfect knowledge are liable to those errors (and probably others besides).

Butterworth, Blandford and Duke (2000) describe how very abstract models of interactive systems can be proposed that can still be held to have a level of 'cognitive plausibility'. We define cognitive plausibility to be a weak claim that all the assumptions about the user's cognition made by a model are justifiable. Cognitive plausibility lies between 'cognitive inspectability'; which asserts that all assumptions in a model can be inspected by cognitive scientists, and 'cognitive validity' which asserts that all the assumptions in a model are firmly linked to accepted theory and empirical evidence. Cognitive validity is not a claim that can practically be made without a much firmer basis of theoretical HCI (Long and Dowell, 1989) and therefore cognitive plausibility is the best modellers can manage.

Butterworth, Blandford and Duke (2000) recognise that cognitive plausibility for abstract models is problematic: the assumptions made about the user in an abstract model may be very implicit and therefore difficult to 'extract' in any way that is comprehensible to cognitive scientists. However although the model itself may be difficult to inspect the *process* by which it is arrived at need not be, and therefore cognitive plausibility may be claimed for an abstract model by justifying the assumptions made about the users during the process of abstracting. Butterworth, Blandford and Duke use a very similar model to the one presented here and discuss its degree of cognitive plausibility; interested readers are directed to that paper.

6.2 Abstract system models

We define abstract system models of both the one-by-one and the en masse strategies. First, we lay down the system model groundwork that is common to both models. The notation used here is that defined by Morgan (1990).

A diary is a set of events.

$$Diary \quad \triangleq \quad \mathrm{P}(E) \tag{12}$$

We denote two diaries: the paper diary and the electronic diary, denoted *pd* and *ed* respectively.

$$pd \ : \ Diary \tag{13}$$

$$ed \ : \ Diary \tag{14}$$

In representing the two diaries equivalently (as the same type) we again make an abstraction over the model. A paper diary and electronic diary are evidently different things (Blandford and Green, 2001) but, at least at a first level of analysis, they store the same information in different ways. How a paper diary and electronic diary store their information is not the focus of this analysis, which is concerned with what information is stored and how it is moved between the two diaries.

Now we propose a representation of the user's goal. This states that when the goal is achieved the contents of the paper diary have been added to the original contents of the electronic diary. We represent this as follows…

$$goal \ \widehat{=} \ \ ed, pd : [ed = ed_o \cup pd_o] \tag{15}$$

This specification fragment is in two parts. *ed, pd*: is the 'frame' of the specification and describes which parts of the state can be affected by the specification – in this case, *ed* and *pd*, the electronic and paper diaries. The equation in the square brackets asserts what is true after the specification has taken place (it is the 'postcondition'). Any variable name subscripted by a zero denotes the value of that variable before the specification fragment has occurred.

Therefore the specification *goal* should be read as '*goal* allows the values of *pd* and *ed* to alter, and finishes with the value of *ed* being equal to its initial value unioned to the initial value of *pd*.' Note that the goal says nothing about the final value of *pd*, i.e. it can finish with *pd* having any value.

Now we propose two abstract programs which represent the two strategies.

6.3 An abstract model of the one by one strategy

The program fragment $strat_1$ represents the one by one strategy. It is simply a loop that repeats while there are events in the paper diary *pd* still left to be entered into the electronic diary, *ed*. In each repetition of the loop an event *e* is selected from those remaining in the paper diary, added to the electronic diary and then marked off the paper diary.

$$strat_1 \mathrel{\widehat{=}} \textbf{do} \quad unmarked \rightarrow \left[\!\!\left[\begin{array}{l} \textbf{var } e \bullet \ selectEvent; \\ \qquad\quad enterEvent; \\ \qquad\quad markEvent \end{array} \right]\!\!\right] \quad \textbf{od} \tag{16}$$

To complete the model of the one by one strategy we need to define the fragments *unmarked*, *selectEvent*, *enterEvent* and *markEvent*, all of which are very simple. *unmarked* is a predicate that holds true while there are events on the paper diary, i.e. when *pd* is not the null set.

$$unmarked \mathrel{\widehat{=}} pd \neq \varnothing \tag{17}$$

selectEvent is an abstract code fragment that ensures that the event *e* is selected from the set *pd*.

$$selectEvent \mathrel{\widehat{=}} e : [\, e \in pd \,] \tag{18}$$

enterEvent adds the event *e* to the electronic diary *ed*.

$$enterEvent \mathrel{\widehat{=}} ed : [ed = ed_o \cup \{e\}] \tag{19}$$

markEvent removes the event *e* from the paper diary *pd*.

$$markEvent \mathrel{\widehat{=}} pd : [pd = pd_o \setminus \{e\}] \tag{20}$$

This completes the abstract model of the one by one strategy.

6.4 An abstract model of the en masse strategy

The program fragment $strat_2$ represents the en masse strategy for the user who relies on the paper diary to prompt them about which events to check or update. It is modelled in a similar way to the one by one strategy. Initially a set of events is added to the electronic diary and subsequently there is a loop that corrects the events in the electronic diary.

$$strat_2 \mathrel{\widehat{=}} setSeries; \tag{21}$$
$$\textbf{do} \quad unmarked \rightarrow \left[\!\!\left[\begin{array}{l} \textbf{var } e, e' \bullet \ selectEvents; \\ \qquad\quad \textbf{if} \\ \qquad\qquad needsCorrecting \rightarrow correctEvent; \\ \qquad\quad \textbf{fi} \\ \qquad\quad markEvent \end{array} \right]\!\!\right] \quad \textbf{od}$$

The fragment *setSeries* adds a collection of events to the electronic diary that are approximately the same as those in the paper diary. The subsequent loop goes through each of the events in the paper diary seeing if it is correctly entered on the electronic diary. The variable *e* is the event on the paper

diary and the variable *e'* is a similar event on the electronic diary and both are set by the fragment *selectEvents*. The **if** statement checks to see if the event *e'* needs correcting and corrects it if it does. Lastly the event *e* is marked off the paper diary by the fragment *markEvent*.

Again, to complete the model we need to go through and define all the program fragments. (Note that *unmarked* and *markEvent* are as defined in the one by one strategy.)

setSeries adds a set of events *es* to *ed* that are similar to *pd*.

$$setSeries \mathrel{\widehat{=}} ed : [\ \exists es \bullet es \approx pd \wedge ed = ed_0 \cup es\ ] \tag{22}$$

(This reads 'there exists a set of events *es* which are similar to the paper diary *pd* and *es* is added to the electronic diary *ed*.')

selectEvents selects two approximately equal events *e* and *e'* so that *e* is from the paper diary and *e'* is from the electronic diary. (The model abstracts over exactly what it means for events to be 'approximately equal'; a precise definition is not necessary, but in more detailed analyses it would be necessary to define what constitutes 'approximate'.)

$$selectEvents \mathrel{\widehat{=}} e, e' : [\ e \in pd \wedge e' \in ed \wedge e \approx e'\ ] \tag{23}$$

needsCorrecting is a predicate that hold true if *e* is not the same as *e'*.

$$needsCorrecting \mathrel{\widehat{=}} e \neq e' \tag{24}$$

correctEvent removes *e'* from the electronic diary and replaces it with *e*.

$$correctEvent \mathrel{\widehat{=}} ed : [\ ed = (\ ed_o \setminus \{e'\}) \cup \{e\}\ ] \tag{25}$$

This completes the abstract model of the en masse strategy.

6.5 Reasoning with the abstract models

When working with the operational models in the previous sections we reasoned with the models by showing that at least one behaviour that reached the user goal of having all the events from the paper diary correctly added to the electronic (i.e. we showed that it is *possible* to get to the goal state).

Using the abstract models we can test the stronger claim that it is *necessary* that the user reaches the goal state, or in other words, all the behaviours described by the models result in the goal state being reached. If the behaviour spaces of the models are infinite it is clearly not possible to test this stronger claim by generating the behaviours explicitly. To illustrate this stronger approach, we employ proof techniques to demonstrate that $strat_1$ and $strat_2$ are 'correct' with respect to *goal*. (Recall that *goal* is a mathematical statement that describes the contents of the paper diary being correctly added to the electronic diary.) If we can prove that the two strategies are correct with respect to *goal* then we have an argument that all the behaviours described by the two strategies result in the goal being achieved.

We employ Morgan's (1990) refinement calculus, which is a collection of 'refinement' translations that can be applied to models, decreasing their level of abstractness while retaining their correctness. Here we present simplified versions of the proofs given in Butterworth, Blandford and Duke (2000).

Reasoning about the one by one strategy

We need to demonstrate that the result of $strat_1$ is that all the events initially in *pd* are in *ed*. We can prove this by showing firstly that the loop in $strat_1$ adds only events from *pd* to *ed*, and secondly

that every time the loop occurs more events are added to *ed*. Technically these two properties are known as the loop 'invariant' and 'variant' respectively and are shown formally below:

$$invar \mathrel{\widehat{=}} (ed_0 \cup pd) \subseteq ed$$

$$var \mathrel{\widehat{=}} ed_0 \subset ed$$

These two properties are both needed for the proof. The loop invariant ensures that only items from *pd* are added to *ed*, and therefore that *ed* must finish up containing only items originally in *ed* added to a set of events that is a subset of or equal to *pd*. The loop variant ensures that items are repeatedly added to *ed*.

Putting these two assertions together: that *ed* contains only items from *ed* and *pd*, and that *ed* increases in size, we can deduce that eventually $ed = ed_0 \cup pd$ as required.

So far we have shown that a loop with the two properties *invar* and *var* will result in *goal* being satisfied. Now it only remains for us to show that the loop in $strat_1$ does indeed fulfil these two properties.

What actually happens in the loop is that a single event *e* is added to *ed*, so we can restate the invariant as:

$$(ed_0 \cup \{e\}) \subseteq ed$$

... and it is clear that *e* comes from *pd*, so we can say that $\{e\} \subseteq pd$.

Putting this together, the loop does not delete anything from *ed*, so $ed_0 \subseteq ed$, and adds subset of *pd*, so $(ed_0 \cup pd) \subseteq ed$, and the loop invariant is shown.

The variant *var* is more simple: event *e* is added to *ed* and nothing is removed from *ed* and therefore *ed* must increase in size, i.e. $ed_0 \subset ed$.

Reasoning about the en masse strategy

The en masse strategy is not correct with respect to *goal*. Demonstrating the failure of correctness is typically easier than demonstrating correctness, as we simply need to show a counter-example. In this case we simply suggest a situation where $strat_2$ completes and does not result in goal being achieved. We can do this by considering the size of the sets involved, if the final size of *ed* is not equal to the size of $(ed_0 \cup pd)$ then *ed* cannot equal $(ed_0 \cup pd)$ and $strat_2$ cannot be correct with respect to goal.

The program fragment *setSeries* adds an arbitrary set of events to *ed* and there is no guarantee that this arbitrary set will be the same size as *pd*. Assume that the set added is larger than *pd*, this will only result in $strat_2$ finishing correctly so long as the subsequent loop can delete events from *ed*. It cannot. The loop merely replaces events in *ed* with corrections: it does not decrease (or increase) the size of *ed*. As the size of the final value of *ed* may be incorrect, the final value of *ed* may also be incorrect, and therefore the proof fails.

6.6 Summary

We have shown abstract models of the strategies and sketched out proofs showing whether or not they necessarily result in the user achieving his goal. Full proofs are presented elsewhere (Butterworth *et al.,* 2000). The proofs are stronger than the claims we made with the operational models, where we only tested the possibility of the user achieving his goal. The abstract nature of the models used in this section facilitates this stronger reasoning. However, it comes at the cost of less explicit assumptions about the details of user behaviour and knowledge, as discussed more fully below.

7. Conclusions

From the analyses of both the operational and abstract models we can draw the conclusion that the device does not support the user well in performing the en masse strategy: it places considerable load on the user and may be error prone. This provides some explanations for the empirical finding that users often choose the one by one strategy in preference to the en masse strategy, even though the one by one strategy requires more input to the device. Each analysis told us different (but consistent) things about the problem. Other analyses using different basic frameworks might have told us yet other different things.

Our aim has been to investigate the benefits and drawbacks of these styles of analysis, which we can now do in detail. The differences between the model-based analyses are summarised in Table 2 and described in more detail below.

	Operational models	**Abstract models**
The tractability of evaluation methods	The model can be simulated automatically or by arguing that a particular behaviour trace is legal or not with respect to the model. Operational models can be used to argue the existence of certain legal behaviours. A finite set of behaviours can be reasoned about.	An abstract model can be used to prove that general properties of behaviour are consistent with the model. In particular, we can use proof techniques to show universal properties of behaviour. Infinite sets of behaviour can be reasoned about.
Inspectability of cognitive assumptions	The link to theory is clear – there is (approximately) a one-to-one translation between the underlying theory and the user model.	The act of abstracting over assumptions and the act of integrating these assumptions with the device model means that those assumptions become implicit in the model.
Level of abstraction	An operational model is no more abstract than the level of abstraction of the underlying theory.	An abstract model can be considerably abstracted in both the device model and user assumptions.
The reuse of models	The theory incorporated into a operational model is general and can be ported from one specific model to another. Different models are implemented by defining different knowledge and goals.	The assumptions retained in an abstract model are specific not only to that example, but also to the questions that are asked of that model. They may be inappropriate for other examples.

Table 2. A comparison between operational and abstract models.

7.1 The tractability of evaluation methods

We showed how we can both hand-simulate and run an implementation of the operational model, which may show the existence of legal traces that do or do not achieve the goal of an interaction. However, an operational model cannot easily be used to ask exhaustive questions about the space of legal behaviours. We can answer questions such as ‘can a rational user get to a goal state?’ or ‘can a rational user make errors?’, but it is harder to answer ‘will the modelled user always reach a goal state?’, i.e. do all legal behaviours reach a goal? To do so we need to employ proof techniques.

We showed that abstract system models can check these more exhaustive properties by showing that the models are (or are not) correct 'implementations' of higher level properties.

Much of the benefit of the analysis of the operational model comes not from showing the legality of given behaviours, but from the act of setting out the knowledge that the user needs to perform the task. The main conclusion of the operational model analysis – that the en masse strategy is more demanding on the user (in terms of the complexity of the operations) – can be reached without having to go to the effort of actually running the model. The act of laying out and defining the system models gets the analyst thinking about the system from a more user centred perspective and many potential knowledge-based usability problems can be ironed out without having to go to the expense of simulating or proving properties about the models (Hall, 1990; Blandford, Buckingham Shum and Young, 1998). The challenge in developing and evaluating an abstract model is ensuring that it is valid, and that it is designed to answer interesting questions.

Predictive system modelling approaches suffer from a double bind in how they are perceived by practitioners, as pointed out by Gray, John and Atwood (1993): 'If models predict results that designers consider 'intuitive' then the models are perceived to be of little value. On the other hand, if models predict results that are counter intuitive, why, in the absence of empirical data, should they be believed?' In choosing the example used in this paper our main motivation was to select an example for which there were known to be usability difficulties and that would explore the different issues raised by simple models of rational behaviour and their roles within interactive system evaluation. We are not specifically trying to show something about the system that an analyst may not already know; however, the level of analysis we have conducted here improves the understanding of the system and the issues involved in improving usability, taking a simple empirical result (that people rarely use the en masse strategy even if they know about it and even though it is usually more efficient) and demonstrating the differences in knowledge needs and in reliability of the result for the user. To put this another way: the modelling presented here is not predictive, in the full sense discussed by Gray *et al*; rather, it provides some plausible explanations for understanding a reported phenomenon. As shown, two of the analyses produce finding that are consistent with the empirical findings, while the third (the efficiency analysis) does not. Such simplified models are inadequate for the detailed predictions and cognitive explanations that are exemplified by the work of Gray (2000), Byrne and Bovair (1997), Byrne (2001) or Kieras, Wood and Meyer (1997); however, they have a role to play in supporting an analyst exploring the behavioural and design consequences of different user knowledge and user strategies, in the style discussed by Fields (2001).

7.2 Inspectability of assumptions

One of the motivations behind the PUM approach, on which this modelling is based, is that it encourages system designers to express explicitly the assumptions that they are making about users. Simply stated, the designer is required to define user knowledge that would allow the user model to interact successfully with the proposed device. This knowledge should be expressed clearly and can then be validated.

In addition to presenting user knowledge for a particular scenario of use, we have also laid out general assumptions about how users deal rationally with their beliefs. These assumptions, based on existing theories, can be further debated and validated – or, indeed, extended or modified.

The abstract models presented here abstract not only over the beliefs that the user requires in order to successfully interact, but also over the operational mechanism itself. Hence the assumptions made in an abstract model are largely implicit, although the process of abstraction itself can be made inspectable. The structure of the model will be determined by the purposes to which it is put, rather than by a desire to make the contents of that model inspectable.

7.3 Level of abstraction

An operational model inherits its level of abstraction from the underlying assumptions. Typically the assumptions are expressed in an operational manner; it deals with the mechanisms that produce rational behaviour. An operational model is therefore limited in how abstract it can be. The assumptions we use were originally posited for use in systems that simulate human problem solving. These systems have been implemented, so much of the theory behind these assumptions is expressed in a form that is useful for implementing working systems. One of the aims of our work has been to 'lift' the assumptions away from the implementational issues and try to express them clearly and abstractly, very much in the style of Newell's (1982) work on the knowledge level. One of the main advances in the work reported here is to drop the need to reduce non-determinism in the model. Previously much effort was put into cataloguing selection heuristics which refined the notion of rationality so that the models would produce a limited, finite space of legal behaviours. In particular, we first started working on abstract models because there were many cases where implementing an operational model seemed to be 'missing the point': the point was not that there needed to be a full simulation of behaviour (the user performs an operation on this data item, then on that one then…), but that there was an emergent pattern of behaviour, and what was interesting was whether that pattern could be relied upon to achieve the goal. Abstract models are better suited to supporting this abstract reasoning than operational ones.

7.4 The re-use of models

The simplicity of the process of instantiating the general framework to an operational model means that the assumptions captured in an operational model can be easily ported into a different example. Indeed the general framework is by definition intended to cover a wide variety of interactive systems, and the implemented model presented here has been 'run' on several different examples. The assumptions in an abstract model are largely implicit, so it is difficult to extract those assumptions and apply them to other examples. Indeed, abstract models are likely to be tailored not only to a particular example, but also to a particular question about that example. The abstract models are of little or no value in addressing usability questions other than those for which they are formulated. However the simplicity of the abstract models means that we may be able to use them to identify similarities between the usability questions asked of them and those asked of usability analyses already performed.

7.5 Discussion

As discussed earlier, HCI draws from both the natural and social sciences and also engineering, design and fine art. Researchers within disciplines operate within research paradigms that determine what questions are considered interesting, and what methods can appropriately be applied to address those questions. HCI is inherently multidisciplinary, and therefore draws on methods from a range of disciplines; the questions of interest concern the quality of interactions between users and computer systems. Long and Dowell (1989) propose that these questions focus primarily on one or other agent to the interaction, with Human Factors focusing on the users within the interactive system and 'Software Engineering' focusing on the computer systems. Barnard and Harrison (1989) argue for a more neutral view of focusing on the interaction in its own right as the object of study. The models presented in this paper illustrate one approach to doing this.

An interaction cannot be understood simply as the separate behaviours of the contributing agents, but has emergent properties that relate to the interaction as an entity in its own right. The operational model forces an explicit representation of the knowledge requirements on the user, given particular behavioural assumptions. The abstract model focuses attention on the interaction as

an entity in its own right, and therefore leads towards a theory of interactions, abstracting away from the details of the agents involved in that interaction.

The research methods employed in this work are based in software engineering and the mathematical techniques that underpin it. However, a particular knowledge-level theory is also expressed within the formalisation, so that software engineering and human factors are brought together and reasoned about within a common framework. Such an approach is necessarily limiting: by creating abstractions, we ignore much of the richness of natural human interaction. However, this apparent weakness of mathematics can also be a strength: as argued by Duke, Barnard, Duce and May (1998), "several key advances in understanding complex problems in computing have come about through the development of mathematical abstractions". HCI presents some of the most complex problems in computing. The different mathematical abstractions presented in this paper exemplify alternative ways of reasoning about these complex problems. We have demonstrated some of the tradeoffs that are being made in adopting a particular level of abstraction, and illustrated what kinds of reasoning each supports (the operational model focusing primarily on knowledge and the abstract model on a safety proof that a particular kind of error will *not* be made given the stated user assumptions).

The stronger proof technique implemented in the abstract proof approach delivers stronger verification results than the operational model approach (e.g. about all possible cases as opposed to individual ones), though based on more assumptions. The result in each case is that *if* the initial assumptions are correct *then* the system is (or is not) error-free. However, as illustrated particularly by the efficiency calculation, findings need to be informed by, and related back to, empirical findings in order to validate them. Modelling discards information – the amount dependent on the degree of abstraction. Successful abstraction occurs if the information lost is not critical to the question under consideration. Validation against empirical results is a way to increase confidence that the assumptions made are appropriate for the question considered. The findings of individual analyses can be considered valid insofar as they are consistent with the user data.

Put another way: the efficiency analysis is interesting because it highlights something that users 'should' do, but don't. In contrast, the operational and abstract models provide plausible partial explanations of why users behave the way they do. The importance of formal analysis here is not to give absolute statements of correctness or otherwise of a system – such absolute statements are not attainable. Rather, in the approach presented, they give a way to probe issues of importance raised as part of the empirical studies or earlier formal analysis.

For example, in terms of the validity of the modelling, one criticism might be that users do not always behave rationally; this is so. However, that does not preclude the usefulness of the approach. It would be unwise to design computer systems that *cannot* be used by rational users, so we would argue that it is a minimal requirement on design that they should be usable by people who have the necessary knowledge that they can plausibly have obtained. Put another way: unless users are really lucky in their choice of actions, real user behaviour is likely to be less successful than that modelled here. Bounded rationality, slips, etc. will all contribute to ineffective behaviour; the PUM approach involves the analyst articulating what knowledge users need to perform effectively, and exploring the consequences of plausible incompleteness or misconceptions (Reason, 1990). In terms of Newell's (1982) principle of rationality, from which the PUM principles are refined: "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action". What is rational for a given agent depends on its knowledge and goals. Thus if such an analysis shows that a system "can be used" then little can be deduced of the use of the real system by real users in general. However, if the analysis suggests it cannot be used, an issue that warrants further investigation has been raised, that ought to be investigated further with reference to empirical data.

Similarly if the efficiency analysis had agreed with the empirical data then it would have been of little interest. It is of interest precisely because it fails to explain the actual behaviour observed. It highlights that further more detailed analysis is needed on that point and that a simple explanation of user behaviour based on optimality cannot be sufficient. Formal methods are costly to apply. They can only be realistically used when targeted at specific problems. It is a sensible use of resources to employ different techniques, in an integrated way, to probe issues that have been highlighted as important either from empirical studies or other analyses. Any individual method is unlikely to provide the whole story, but each has the potential to yield valuable insights into possible explanations for observed behaviour.

In summary, this work applies a mathematical paradigm (specification and proof) to the study of interaction. The focus has been on understanding the tradeoffs between provability and inspectability for systems that include a human agent. We do not expect such mathematical techniques to be used routinely in interactive system design (however see Good and Blandford (1999), where practical discount techniques closely derived from the theoretical ideas expressed here are shown to have value in a 'real world' design context); their contribution is, rather, to express issues without ambiguity. Our aim has been to demonstrate particular, complementary styles of reasoning and to illustrate what contributions mathematical modelling can make to reasoning about interactive system design. For the diary case study, the various models have each provided different insights that have overall yielded a fuller picture of the issues surrounding the use of one feature of a diary system. In doing so, we have compared the various styles of modelling to also gain a better understanding of the strengths and limitations of each.

Acknowledgements

This work was supported by EPSRC Grant GR/L00391. We are grateful to David Duke, Harold Thimbleby and Richard Young for useful discussions on the ideas presented here and to anonymous referees for constructive feedback on earlier versions of this paper.

References

ABOWD, G., WANG, H-M. AND MONK, A. (1995) A formal technique for automated dialogue development. In *Proc. DIS'95*. ACM. 219 – 226.

ADAMS, A. & BLANDFORD, A. (2002) Acceptability of Medical Digital Libraries. *Health Informatics Journal*. 8(2), 58-66. Sheffield Academic Press. ISSN 1460-4582.

ANDERSON, J. R. (1993) *Rules of the Mind,* Hillsdale, NJ: LEA.

BARNARD, P.J. AND HARRISON, M.D. (1989). Integrating Cognitive and System Models in Human Computer Interaction. In A. SUTCLIFFE AND L. MACAULEY, Eds. *People and Computers V, Proceedings of HCI'89,* 87-103. Cambridge: CUP.

BARNARD, P. J., MAY, J., DUKE, D. & DUCE, D. (2000), Systems, Interactions and Macrotheory. *ACM Transactions on Computer-Human Interaction*, 7.2, 222-262.

BEARD, D. & PALANLAPPAP, M. WITH HUMM, A., BANKS, D., NAIR, A. & SHAN, Y-P. (1990) A visual calendar for scheduling group meetings. In *Proc. CSCW'90*. 279-290

BLANDFORD, A. E., BUCKINGHAM SHUM, S. AND YOUNG, R. M. (1998) Training software engineers in a novel usability evaluation technique. *International Journal of Human-Computer Studies*, 45(3), 245-279.

BLANDFORD, A. E. & GREEN, T. R. G. (2001) Group and individual time management tools: what you get is not what you need. *Personal and Ubiquitous Computing*. 5.4. 213-230.

BLANDFORD, A. E. & YOUNG, R. M. (1993). Developing runnable user models: Separating the problem solving techniques from the domain knowledge. In J. ALTY, D. DIAPER AND S. GUEST, Eds. *People and Computers VIII, Proceedings of HCI'93, Loughborough,* 111-122 Cambridge: Cambridge University Press.

BLANDFORD, A. E. & YOUNG, R. M. (1995) 'Separating user and device descriptions for modelling interactive problem solving'. In K. Nordby, P. Helmersen, D J Gilmore, and S Arnesen, Eds: *Human-Computer Interaction: Interact'95*. pp. 91-96. Chapman and Hall

BLANDFORD, A. E. & YOUNG, R. M. (1996) Specifying user knowledge for the design of interactive systems. *Software Engineering Journal*. 11.6, 323-333.

BUTTERWORTH, R. J., BLANDFORD, A. E. & DUKE, D. J. (1998) The role of formal proof in modelling interactive behaviour. In P. Markopulos & P. Johnson (Eds.) *Proc. Design, Specification and Verifications of Interactive Systems '98*. pp. 87-101. Wien: Springer.

BUTTERWORTH, R., BLANDFORD, A. & DUKE, D. (1999). Using formal models to explore display based usability issues. *Journal of Visual Languages and Computing*, 10. pp. 455-479.

BUTTERWORTH, R., BLANDFORD, A. & DUKE, D. (2000) Demonstrating the cognitive plausibility of interactive system specifications. *Formal Aspects of Computing*, 12. pp. 237-259.

BYRNE, M. (2001) ACT-R/PM and menu selection: applying a cognitive architecture to HCI. *International Journal of Human–Computer Studies*. 55. 41-84.

BYRNE, M. D. & BOVAIR, S. (1997) A working memory model of a common procedural error. *Cognitive Science*. 21.1, 31-61.

CARD, S.K., MORAN, T.P. & NEWELL, A. (1983). *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Associates: Hillsdale, NJ.

CARROLL, J. M., (ed) (2003) *HCI Models, Theories and Frameworks*. Morgan Kaufmann, San Francisco.

CURZON, P. & BLANDFORD, A. E. (2001) Detecting multiple classes of user error. In *Engineering for Human-Computer Interaction, 8th IFIP International Conference, EHCI 2001*, M. R. LITTLE & L. NIGAY ,Eds, pp 57-71, Lecture Notes in Computer Science, 2254, Springer.

CURZON, P. & BLANDFORD, A. (2002) From a Formal User Model to Design Rules, In P. FORBRIG, B. URBAN, J. VANDERDONCKT & Q. LIMBOURG, Eds. *Interactive Systems. Design, Specification and Verification, 9th Int Workshop*. pp 19-33. Lecture Notes in Computer Science, 2545.

DOHERTY, D. J., CAMPOS, J. F. & HARRISON, M. D. (2000) Representational Reasoning and Verification. *Formal Aspects of Computing*, 12. pp. 260-277.

DUKE, D. J., BARNARD, P.J., DUCE, D.A. & MAY, J. (1998) Syndetic Modelling. *Human-Computer Interaction*. 13, 337-394

DUKE, D.J., & HARRISON, M.D. (1993) Abstract Interaction Objects. *Computer Graphics Forum* 12(3). 25-36. NCC/Blackwell. Proc. Eurographics'93.

FIELDS, R. (2001) *Analysis of erroneous actions in the design of critical systems*. DPhil Thesis. University of York. Technical Report YCST 2001/09.

FIELDS, B., WRIGHT, P. & HARRISON, M. (2000) Analysing Human-Computer Interaction as distributed cognition. *Human-Computer Interaction Journal*. 15. 1-41.

GÄRDENFORS, P. (1988) *Knowledge in flux: modelling the dynamics of epistemic states*. MIT Press.

GOOD, J. P. & BLANDFORD, A. E. (1999) Incorporating Human Factors Concerns into the Design and Safety Engineering of Complex Control Systems. In J. Noyes & M. Bransby (Eds.) *People in Control: An International Conference on Human Interfaces in Control rooms, Cockpits and Command Centres*, IEE Conference Publication Number 463, Institution of Electrical Engineers, London, 1999. ISBN number 0 85296 715 2. Pages 51 - 56.

GRAY, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science*, 24(2), 205-248.

GRAY, W., JOHN, B & ATWOOD, M. (1993) 'Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance', *Human-Computer Interaction*, 8. pp 237-309.

GRAY, W., YOUNG, R. M. & KIRSCHENBAUM, S. (1997) Introduction to this special issue on cognitive architectures and Human–Computer Interaction. *Human-Computer Interaction*. 12. 301-309.

HALL, A. (1990) *Seven Myths of Formal Methods* IEEE Software, September. 11-19.

HARRISON, M. AND THIMBLEBY, H., Eds. (1990) *Formal Methods in Human-Computer Interaction,* 97-127. Cambridge: CUP.

HOLLAN, J. D., HUTCHINS, E. L. & KIRSH, D. (2000) Distributed cognition: toward a new foundation for human-computer interaction research. *ACM Transactions on CHI,* 7.2, 174-196.

HOLLNAGEL, E. (1998) *Cognitive Reliability and Error Analysis Method (CREAM)*. Oxford : Elsevier Science.

HUTCHINS, E. *Cognition In The Wild*. MIT Press, Cambridge, MA. (1995)

JOHN, B. & KIERAS, D. (1996) The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on CHI*, 3, 320-351.

KELLEY, J. F. & CHAPANIS, A. (1982) How professional persons keep their calendars: implications for computerisation. *Journal of Occupational Psychology*. 55. 241-256.

KIERAS, D. E., WOOD, S. D. & MEYER, D. E. (1997) Predictive Engineering Models Based on the EPIC Architecture for a Multimodal High-Performance Human-Computer Interaction Task. *ACM Transactions on CHI*, 4, 230-275.

KIERAS, D. & POLSON, P. (1985). An Approach to the Formal Analysis of User Complexity. *International Journal of Man-Machine Studies*, 22, 365-394.

KINCAID, C. M., DUPONT, P. B. & KAYE, A. R. (1985) Electronic Calendars in the Office: An Assessment Of User Needs And Current Technology. *ACM Transactions on Office Information Systems*. 3.1. 89-102.

KLEIN, G. A. (1999) *Sources of Power: How people make decisions*. Cambridge, MA: The MIT Press.

LAMPORT, L. (1994) The temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*. 16(3). 872–923.

LONG, J. & DOWELL, J. (1989). Conceptions of the discipline of HCI: Craft, Applied Science and Engineering. *Proc. HCI'89,* 9-32 Cambridge: Cambridge Univ. Press.

MOHER, T.G. & DIRDA, V. (1995) Revising mental models to accommodate expectation failures in human-computer dialogues. In P. PALANQUE & R. BASTIDE, Eds, *Design, Specification and Verification of Interactive Systems '95*. pp.76-92. Wien : Springer.

MORGAN, C. (1990) *Programming from specifications*. Prentice Hall.

NEWELL, A. (1982) The knowledge level *Artificial Intelligence*, **18**, 87-127.

NEWELL, A. (1990) *Unified Theories of Cognition*, Harvard University Press, Cambridge, MA.

ON Technology (1995) *Meeting Maker (v.3) User's Guide*.

PALANQUE, P. & BASTIDE, R. (1995) Petri net based design of user-driven interfaces using the Interactive Cooperative Objects formalism. In F. PATERNO', Ed, *Interactive Systems: Design, Specification and Verification*. Springer Verlag. 383-400.

PALEN, L. (1999) Social, Individual & Technological Issues for Groupware Calendar Systems. *Proc. CHI'99* 17-24. ACM Press.

PATERNO', F. (1993) Definition of Properties of User Interfaces Using Action-Based Temporal Logic. *Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering*, pp. 314-318.

PAYNE, S. J. AND GREEN, T.R.G. (1986). Task-Action Grammars: a model of mental representation of task languages. *Human-Computer Interaction,* **2,** 93-133.

PAYNE, S.J. (1991). Display-based action at the user interface. *International Journal of Man-Machine Studies,* **35,** 275-289.

PAYNE, S. J. (1993) Understanding calendar use. *Human-Computer Interaction*. 8. 83-100.

PNUELI, A. (1992) System specification and refinement in temporal logic. In *Lectures Notes on Computer Science*, Vol. 652. Springer Verlag. 1-38.

POLLOCK, J. (1993). The Phylogeny of Rationality. *Cognitive Science,* **17,** 563-588.

REASON, J. (1990) *Human Error*. Cambridge : Cambridge University Press.

RITTER, F. & YOUNG, R. (2001) Embodied models as simulated users: introduction to this special issue on using cognitive models to improve interface design. *Int. J. Human-Computer Studies*. 55. 1-14.

RUSHBY, J. (1999) Using model checking to help discover mode confustions and other automation surprises. In D. Javaux (Ed.) *Proc. 3rd Workshop on Human Error, Safety and System Development*.

RYAN, M. (1992) *Ordered presentations of theories: a hierarchical approach to default reasoning*. PhD Thesis, Imperial College, London.

SALVUCCI, D. D. & LEE, F. J. (2003) Simple cognitive modeling in a complex cognitive architecture. *Proc. ACM CHI 2003*. 265 - 272

SIMON, H.A. (1987): Bounded rationality. In: J. Eatwell, M. Millgate & P. Newman (eds.): *The New Palgrave: A Dictionary of Economics*. London and Basingstoke: Macmillan

THIMBLEBY, H., CAIRNS P. & JONES M. (2001) Usability Analysis with Markov Models, ACM Transactions on Computer Human Interaction, 8(2) 99–132.

TVERSKY, A. & KAHNEMAN, D. (1992) Advances in prospect theory: cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, 5. 297 – 323.

VICENTE, K. (1999) *Cognitive Work Analysis*. Mahwah, NJ : Lawrence Erlbaum.

WHARTON, C., RIEMAN, J., LEWIS, C. & POLSON, P. (1994) The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R. Mack (Eds.), *Usability inspection methods* (pp. 105-140) New York: John Wiley.

YOUNG, R. M., GREEN, T. R. G., & SIMON, T. (1989) Programmable user models for predictive evaluation of interface designs. In *Proceedings of CHI '89*. ACM, New York.

Appendix A: the core of the Lisp implementation of the principles of rationality

The function that follows is the core of the Lisp implementation that produces the running simulation of the operational model. The full Lisp code is available from www.uclic.ucl.ac.uk/annb/puma/ .

```
;user-do is the core of the implementation, implementing most of the principles of
;behaviour. The Principle of Belief Updating is implemented in the function
;'visible-effects'
(defun user-do (user-state)
;print out trace information to show how the device and user states change over time
  (terpri)
  (terpri)(princ "device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+")(terpri)
  (princ device-state)
  (terpri)(princ "user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - ")(terpri)
  (princ user-state)
  (terpri)(princ "-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+")(terpri)
;
;if the goal of the currently committed action has in fact been achieved, then
;drop the commitment.
; (part of Principle of Commitment Performance)
  (cond ((review-commitment user-state)
         (terpri)
         (princ "dropping a commitment; have already achieved the goal of ")
         (princ (commitp user-state))
         (user-do (rem-comm user-state)))
;
;if you've reached the goal state then quit
; (Principle of Goal Driven Behaviour)
        ((desired-reached user-state) (princ "desired reached") user-state)
;
;if there are no goals and no commitments, adopt goals
        ((and (null (list-extract 'goals user-state))
              (null (commitp user-state)))
         (terpri)(princ "adopting goals ")
         (user-do (review-goals user-state)))
;
;if there are goals but no commitments, consider ways to address goals
; (Principle of Commitment)
        ((and (list-extract 'goals user-state)
              (null (commitp user-state)))
         (terpri)(princ "considering ways to address goals ")
         (princ (list-extract 'goals user-state))
         (user-do (user-commit
                   (user-might user-state))))
;
;if you can do the operation right now, then do it!
; (Principle of Immediate Performance)
        ((precond-true (commitp user-state) (list-extract 'knows-ds user-state))
         (terpri)(princ "preconditions satisfied, so doing it!")
         (user-do (visible-effects (issue-command user-state))))
;
;if not, then sub-goal on it and adopt new commitments
; (part of Principle of Commitment Performance)
        ((commitp user-state)
         (terpri)(princ "subgoaling")
        (user-do (user-commit
                   (user-might (sub-goal user-state)))))
;
;and if you can't go any further then give up!
        (t (princ "goal failed") user-state)))
```

Appendix B: Implemented device model

The device model, together with the initial device state and the functions that determine what is visible to the user on the display at any time, are as follows. For simplicity, the function 'newseries' is implemented with a start event, start time and interval, and adds precisely 11 events, rather than with base, interval and finish as presented in the hand simulation (section 5).

As shown here, 'world knowledge' about events being almost equal and times being similar are set in the device-state, and treated as being visible to the user.

This device model works with all versions of the user knowledge (Appendix C).

```
;================================
;below are the functions that correspond to the device description of
;the electronic diary modelled by Blandford & Butterworth (Jan 2003)
;
;initialise the device state
(defun reset-state nil
  (setq device-state '((firstvisible 30)(lastvisible 40)(is-visible 33))))
(reset-state)
;
;calculate what is visible on the display
(defun visibility-test (rel)
  (cond ((equal (car rel) 'ined) (test-visible (caddr rel)))
        ((equal (car rel) 'is-visible) (test-visible (cadr rel)))
        ((member (car rel) '(firstvisible lastvisible)) t)
        (t nil)))
(defun test-visible (t1)
  (cond ((and (< (car (list-extract 'firstvisible device-state)) t1)
              (> (car (list-extract 'lastvisible device-state)) t1)) t)
        (t nil)))
;
;define the effects of user actions on the device state.
;the definition of newseries is necessarily somewhat simplified here...
(defun scrollto (t1)
  (update-value (list 'firstvisible (- t1 5)))
  (update-value (list 'lastvisible (+ t1 5)))
  (update-value (list 'is-visible t1)))
;
(defun newevent (e1 t1)
  (add-value (list 'ined e1 t1)))
;
(defun makecorrection (e1 t1 e2 t2)
  (delete-value (list 'ined e1 t1))
  (add-value (list 'ined e2 t2)))
;
(defun delextra (e1 t1)
  (delete-value (list 'ined e1 t1)))
;
(defun newseries (e1 t1 interval)
  (setq interval 10)
  (add-value (list 'ined e1 t1))
  (add-value (list 'ined e1 (+ t1 interval)))
  (add-value (list 'ined e1 (+ t1 (* 2 interval))))
  (add-value (list 'ined e1 (+ t1 (* 3 interval))))
  (add-value (list 'ined e1 (+ t1 (* 4 interval))))
  (add-value (list 'ined e1 (+ t1 (* 5 interval))))
  (add-value (list 'ined e1 (+ t1 (* 6 interval))))
  (add-value (list 'ined e1 (+ t1 (* 7 interval))))
  (add-value (list 'ined e1 (+ t1 (* 8 interval))))
  (add-value (list 'ined e1 (+ t1 (* 9 interval))))
  (add-value (list 'ined e1 (+ t1 (* 10 interval)))))
```

Appendix C: implemented user models

All three implemented user models are presented here. For clarity, the minimum information needed to perform as outlined is included in each model. In these models, each week is arbitrarily split into ten time intervals, with Wednesdays 3-4pm being at times 33, 43, 53, etc. and later times on Wednesdays being 34, 44, 54, etc. A seminar in Meeting Room 1 is encoded as EventA; other events are encoded as EventB and EventC (see Figure 6).

First, the model for the one-by-one strategy…

```
;this is the model for a user with an electronic diary, modelling knowledge needed for
;only adding events one at a time
;
(setq user-knows
      '((relations (ined event time)(almostequal event event)
                   (neartime time time))
        (properties (firstvisible time)(lastvisible time)(is-visible time)(is-start
time))
        (variables (t1 t2 e1 e2 interval))
       (newevent
         (arguments e1 t1)
         (precond (is-visible t1))
         (tracked (add ined e1 t1))
         (relevant (ined e1 t1)))
        (scrollto
         (arguments t1)
         (relevant (is-visible t1)))
       ))
;;
(setq desired-state '((ined eventA 33)(ined eventB 44)
                      (ined eventA 63)(ined eventA 73)(ined eventA 83)
                      (ined eventC 94)(ined eventA 103)(ined eventA 123)
                      (ined eventA 133)))
```

The model for the en masse strategy with errors (modelling the user who uses the paper diary as the cue for which events to correct) is as follows. This includes statements of things the user needs to know that are not visible on the display. Changes from the one-by-one model are marked in bold.

```
;this is the model for a user with an electronic diary, excluding knowledge about
;how to delete events
;
(setq user-knows
      '((relations (ined event time)(almostequal event event)
                   (neartime time time))
        (properties (firstvisible time)(lastvisible time)(is-visible time)(is-start
time))
        (variables (t1 t2 e1 e2 interval))
        (makecorrection
         (arguments e1 t1 e2 t2)
         (relevant (ined e2 t2))
         (precond (is-visible t1))
         (tracked (add ined e2 t2)(delete ined e1 t1))
         (filter (almostequal e1 e2)(neartime t1 t2)(ined e1 t1)))
        (newevent
         (arguments e1 t1)
         (precond (is-visible t1))
         (tracked (add ined e1 t1))
         (filter (neartime t1 t2)(almostequal e1 e2)(not ined e2 t2))
         (relevant (ined e1 t1)))
        (scrollto
         (arguments t1)
         (relevant (is-visible t1)))
        (newseries
         (arguments e1 t1 interval)
         (tracked (add ined e1 43)(add ined e1 53)
                  (add ined e1 63)(add ined e1 73)(add ined e1 83)
                  (add ined e1 93)(add ined e1 103)(add ined e1 113)
```

```
                (add ined e1 123)(add ined e1 133))
         (precond (is-visible t1))
         (filter (neartime t1 t2)(almostequal e1 e2)(not ined e2 t2)(is-start t1))
         (relevant (ined e1 t1)))
        (knows-not-visible (is-start 33)
                           (almostequal eventA eventA)(almostequal eventA eventB)
                           (almostequal eventB eventA)(almostequal eventC eventA)
                           (almostequal eventA eventC)(almostequal eventA eventD)
                           (almostequal eventD eventA)
                           (neartime 33 33)(neartime 43 44)(neartime 44 43)
                           (neartime 53 53)(neartime 63 63)(neartime 94 93)
                           (neartime 73 73)(neartime 83 83)(neartime 93 94)
                           (neartime 103 103)
                           (neartime 113 113)(neartime 123 123)(neartime 133 133))
       ))
;;
(setq desired-state '((ined eventA 33)(ined eventB 44)
                      (ined eventA 63)(ined eventA 73)(ined eventA 83)
                      (ined eventC 94)(ined eventA 103)(ined eventA 123)
                      (ined eventA 133)))
```

Finally, the knowledge needed to correctly complete the en masse strategy is as follows:

```
;this is the model for a user with an electronic diary, including knowledge about
;how to delete events
;
(setq user-knows
      '((relations (ined event time)(almostequal event event)
                   (neartime time time))
        (properties (firstvisible time)(lastvisible time)(is-visible time)(is-start
time))
        (variables (t1 t2 e1 e2 interval))
        (makecorrection
         (arguments e1 t1 e2 t2)
         (relevant (ined e2 t2))
         (precond (is-visible t1))
         (tracked (add ined e2 t2)(delete ined e1 t1))
         (filter (almostequal e1 e2)(neartime t1 t2)(ined e1 t1)))
        (newevent
         (arguments e1 t1)
         (precond (is-visible t1))
         (tracked (add ined e1 t1))
         (filter (neartime t1 t2)(almostequal e1 e2)(not ined e2 t2))
         (relevant (ined e1 t1)))
        (scrollto
         (arguments t1)
         (relevant (is-visible t1)))
        (delextra
         (arguments e1 t1)
         (precond (is-visible t1))
         (relevant (not ined e1 t1))
         (tracked (delete ined e1 t1)))
        (newseries
         (arguments e1 t1 interval)
         (tracked (add ined e1 43)(add ined e1 53)
                  (add ined e1 63)(add ined e1 73)(add ined e1 83)
                  (add ined e1 93)(add ined e1 103)(add ined e1 113)
                  (add ined e1 123)(add ined e1 133))
         (precond (is-visible t1))
         (filter (neartime t1 t2)(almostequal e1 e2)(not ined e2 t2)(is-start t1))
         (relevant (ined e1 t1)))
(knows-not-visible (is-start 33)
                           (almostequal eventA eventA)(almostequal eventA eventB)
                           (almostequal eventB eventA)(almostequal eventC eventA)
                           (almostequal eventA eventC)(almostequal eventA eventD)
                           (almostequal eventD eventA)
                           (neartime 33 33)(neartime 43 44)(neartime 44 43)
                           (neartime 53 53)(neartime 63 63)(neartime 94 93)
                           (neartime 73 73)(neartime 83 83)(neartime 93 94)
```

```
                          (neartime 103 103)
                          (neartime 113 113)(neartime 123 123)(neartime 133 133))
        ))
;;
(setq desired-state '((ined eventA 33)(ined eventB 44)
                     (ined eventA 63)(ined eventA 73)(ined eventA 83)
                     (ined eventC 94)(ined eventA 103)(ined eventA 123)
                     (ined eventA 133)(not ined eventA 53)(not ined eventA 113)))
```

Appendix D: traces of interaction with the instantiated Lisp model

In the following, all text in Courier font is taken directly from the trace; text in Times – like this sentence – provides additional explanation. '…' is used to denote trace text deleted for the sake of brevity.

1: The one-by-one strategy:

This trace of the one-by-one strategy was taken from a version of the program that omits knowledge about various relationships, such as events being almost equal and times being nearly the same, that are not used in this strategy.
The initial device and user states:

```
device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((firstvisible 30) (lastvisible 40) (is-start 33) (is-visible 33))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (firstvisible 30) (lastvisible 40) (is-start 33) (is-visible 33)))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

adopting goals

device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((firstvisible 30) (lastvisible 40) (is-start 33) (is-visible 33))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (firstvisible 30) (lastvisible 40) (is-start 33) (is-visible 33))
 (goals (ined eventa 33) (ined eventb 44) (ined eventa 63) (ined eventa 73)
  (ined eventa 83) (ined eventc 94) (ined eventa 103) (ined eventa 123)
  (ined eventa 133)))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

considering ways to address goals …
the choice is: ((newevent eventa 33 (ined eventa 33))
                (newevent eventb 44 (ined eventb 44))
                (newevent eventa 63 (ined eventa 63))
                (newevent eventa 73 (ined eventa 73))
                (newevent eventa 83 (ined eventa 83))
                (newevent eventc 94 (ined eventc 94))
                (newevent eventa 103 (ined eventa 103))
                (newevent eventa 123 (ined eventa 123))
                (newevent eventa 133 (ined eventa 133)))
(newevent eventa 33 (ined eventa 33)) y/n? y
```

Each of the choices above represents an alternative trace through the space of possible interactions. By consistently choosing the first option presented, the analyst is selecting the strategy of entering events in chronological order.

```
preconditions satisfied, so doing it!
issued command (newevent eventa 33)
adopting goals
the choice is: ((newevent eventb 44 (ined eventb 44))
                (newevent eventa 63 (ined eventa 63))
                (newevent eventa 73 (ined eventa 73))
                (newevent eventa 83 (ined eventa 83))
                (newevent eventc 94 (ined eventc 94))
                (newevent eventa 103 (ined eventa 103))
                (newevent eventa 123 (ined eventa 123))
```

```
                (newevent eventa 133 (ined eventa 133)))
(newevent eventb 44 (ined eventb 44)) y/n? y
```

the precondition for this action is not satisfied, so the modelled user has to adopt the new goal of making the target time visible, which has no preconditions and can therefore be done immediately.

```
subgoaling
the choice is: ((scrollto 44 (is-visible 44)))
preconditions satisfied, so doing it!
issued command (scrollto 44)
preconditions satisfied, so doing it!
issued command (newevent eventb 44)
[…]
issued command (newevent eventa 133)


device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((ined eventa 133) (is-visible 133) (lastvisible 138) (firstvisible 128)
 (ined eventa 123) (ined eventa 103) (ined eventc 94) (ined eventa 83)
 (ined eventa 73) (ined eventa 63) (ined eventb 44) (ined eventa 33)
 (is-start 33))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (ined eventa 133) (is-visible 133) (lastvisible 138)
  (firstvisible 128) (ined eventa 123) (ined eventa 103) (ined eventc 94)
  (ined eventa 83) (ined eventa 73) (ined eventa 63) (ined eventb 44)
  (ined eventa 33) (is-start 33))
 (committed))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
desired reached
```

After all nine events have been entered, the system outputs the final state of user knowledge about the device state. As can be seen here, the user's knowledge about the device state is accurate in this case.

2: The en masse strategy (incorrect behaviour version):

As noted above (section 5), the user has to know more for this version – in particular, about events and times being similar.

The initial state is as follows:

```
device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((firstvisible 30) (lastvisible 40) (is-start 33) (almostequal eventa eventa)
 (almostequal eventa eventb) (almostequal eventb eventa)
 (almostequal eventc eventa) (almostequal eventa eventc)
 (almostequal eventa eventd) (almostequal eventd eventa) (neartime 33 33)
 (neartime 43 44) (neartime 44 43) (neartime 53 53) (neartime 63 63)
 (neartime 94 93) (neartime 73 73) (neartime 83 83) (neartime 93 94)
 (neartime 103 103) (neartime 113 113) (neartime 123 123) (neartime 133 133)
 (is-visible 33))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (firstvisible 30) (lastvisible 40) (is-start 33)
  (almostequal eventa eventa) (almostequal eventa eventb)
  (almostequal eventb eventa) (almostequal eventc eventa)
  (almostequal eventa eventc) (almostequal eventa eventd)
  (almostequal eventd eventa) (neartime 33 33) (neartime 43 44) (neartime 44 43)
  (neartime 53 53) (neartime 63 63) (neartime 94 93) (neartime 73 73)
  (neartime 83 83) (neartime 93 94) (neartime 103 103) (neartime 113 113)
  (neartime 123 123) (neartime 133 133) (is-visible 33)))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

As before, the modelled user adopts goals and identifies actions that satisfy those goals. The user knows about entering events one by one, so in this case the analyst has to choose the second alternative presented to enter a new series:

```
adopting goals
considering ways to address goals …
the choice is: ((newevent eventa 33 (ined eventa 33))
```

```
                (newseries eventa 33 interval (ined eventa 33))
                (newevent eventb 44 (ined eventb 44))
                (newevent eventa 63 (ined eventa 63))
                (newevent eventa 73 (ined eventa 73))
                (newevent eventa 83 (ined eventa 83))
                (newevent eventc 94 (ined eventc 94))
                (newevent eventa 103 (ined eventa 103))
                (newevent eventa 123 (ined eventa 123))
                (newevent eventa 133 (ined eventa 133)))
(newevent eventa 33 (ined eventa 33)) y/n? n

(newseries eventa 33 interval (ined eventa 33)) y/n? y

preconditions satisfied, so doing it!
issued command (newseries eventa 33 interval)
```

Once the event series has been entered, the modelled user adopts further goals to correct incorrect events. In this case, the analyst only has one choice to make (which one to correct first), and from that point on the simulation runs its course:

```
adopting goals

considering ways to address goals ((ined eventb 44) (ined eventc 94))
the choice is: ((makecorrection eventa 43 eventb 44 (ined eventb 44))
                (makecorrection eventa 93 eventc 94 (ined eventc 94)))
(makecorrection eventa 43 eventb 44 (ined eventb 44)) y/n? y

subgoaling
the choice is: ((scrollto 43 (is-visible 43)))
preconditions satisfied, so doing it!
issued command (scrollto 43)
preconditions satisfied, so doing it!
issued command (makecorrection eventa 43 eventb 44)
[…]
issued command (makecorrection eventa 93 eventc 94)


device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((ined eventc 94) (is-visible 93) (lastvisible 98) (firstvisible 88)
 (ined eventb 44) (ined eventa 133) (ined eventa 123) (ined eventa 113)
 (ined eventa 103) (ined eventa 83) (ined eventa 73) (ined eventa 63)
 (ined eventa 53) (ined eventa 33) (is-start 33) (almostequal eventa eventa)
 (almostequal eventa eventb) (almostequal eventb eventa)
 (almostequal eventc eventa) (almostequal eventa eventc)
 (almostequal eventa eventd) (almostequal eventd eventa) (neartime 33 33)
 (neartime 43 44) (neartime 44 43) (neartime 53 53) (neartime 63 63)
 (neartime 94 93) (neartime 73 73) (neartime 83 83) (neartime 93 94)
 (neartime 103 103) (neartime 113 113) (neartime 123 123) (neartime 133 133))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (ined eventc 94) (is-visible 93) (lastvisible 98) (firstvisible 88)
  (ined eventb 44) (ined eventa 33) (ined eventa 53) (ined eventa 63)
  (ined eventa 73) (ined eventa 83) (ined eventa 103) (ined eventa 113)
  (ined eventa 123) (ined eventa 133) (is-start 33) (almostequal eventa eventa)
  (almostequal eventa eventb) (almostequal eventb eventa)
  (almostequal eventc eventa) (almostequal eventa eventc)
  (almostequal eventa eventd) (almostequal eventd eventa) (neartime 33 33)
  (neartime 43 44) (neartime 44 43) (neartime 53 53) (neartime 63 63)
  (neartime 94 93) (neartime 73 73) (neartime 83 83) (neartime 93 94)
  (neartime 103 103) (neartime 113 113) (neartime 123 123) (neartime 133 133))
 (committed))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
desired reached
```

In this version of the implementation, the user does know about the incorrect events (indicated in bold in the final user state above) but ‘ignores’ the fact that they are not in the desired state. A more sophisticated implementation, including a more sophisticated representation of user knowledge, would be required to model more sophisticated incorrect knowledge. A partial implementation of

this (which 'mocks up' the three-valued logic needed to implement this properly) is available on the web site (www.uclic.ucl.ac.uk/annb/puma/), but a full implementation is not currently available.

3: The en masse strategy (correct behaviour version):

Given the same set of initial choices as above, the analyst selects newseries:

```
[…]
 (newseries eventa 33 interval (ined eventa 33)) y/n? y
```

This time, the user model, identifies four differences and possible actions to correct them:

```
considering ways to address goals…
the choice is: ((makecorrection eventa 43 eventb 44 (ined eventb 44))
                (makecorrection eventa 93 eventc 94 (ined eventc 94))
                (delextra eventa 53 (not ined eventa 53))
                (delextra eventa 113 (not ined eventa 113)))
```

The modelled user proceeds to make corrects and delete events to reach the desired state:

```
device-state: -+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
((is-visible 113) (lastvisible 118) (firstvisible 108) (ined eventc 94)
 (ined eventb 44) (ined eventa 133) (ined eventa 123) (ined eventa 103)
 (ined eventa 83) (ined eventa 73) (ined eventa 63) (ined eventa 33) (is-start 33)
 (almostequal eventa eventa) (almostequal eventa eventb)
 (almostequal eventb eventa) (almostequal eventc eventa)
 (almostequal eventa eventc) (almostequal eventa eventd)
 (almostequal eventd eventa) (neartime 33 33) (neartime 43 44) (neartime 44 43)
 (neartime 53 53) (neartime 63 63) (neartime 94 93) (neartime 73 73)
 (neartime 83 83) (neartime 93 94) (neartime 103 103) (neartime 113 113)
 (neartime 123 123) (neartime 133 133))
user-state: - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
((knows-ds (is-visible 113) (lastvisible 118) (firstvisible 108) (ined eventc 94)
  (ined eventb 44) (ined eventa 33) (ined eventa 63) (ined eventa 73)
  (ined eventa 83) (ined eventa 103) (ined eventa 123) (ined eventa 133)
  (is-start 33) (almostequal eventa eventa) (almostequal eventa eventb)
  (almostequal eventb eventa) (almostequal eventc eventa)
  (almostequal eventa eventc) (almostequal eventa eventd)
  (almostequal eventd eventa) (neartime 33 33) (neartime 43 44) (neartime 44 43)
  (neartime 53 53) (neartime 63 63) (neartime 94 93) (neartime 73 73)
  (neartime 83 83) (neartime 93 94) (neartime 103 103) (neartime 113 113)
  (neartime 123 123) (neartime 133 133))
 (committed))
-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
desired reached
```