Using Formal Models to Design User Interfaces
A Case Study

Judy Bowen
Department of Computer Science
University of Waikato
Hamilton, New Zealand
+ 64 7 838 4021

jab34@cs.waikato.ac.nz

Steve Reeves
Department of Computer Science
University of Waikato
Hamilton, New Zealand
+ 64 7 838 4021

stever@cs.waikato.ac.nz

ABSTRACT
The use of formal models for user interface design can provide a number of benefits. It can help to ensure consistency across designs for multiple platforms, prove properties such as reachability and completeness and, perhaps most importantly, can help incorporate the user interface design process into a larger, formally-based, software development process. Often, descriptions of such models and examples are presented in isolation from real-world practice in order to focus on particular benefits, small focused examples or the general methodology. This paper presents a case study of developing the user interface to a new software application using a particular pair of formal models, presentation models and presentation interaction models. The aim of this study was to practically apply the use of formal models to the design process of a UI for a new software application. We wanted to determine how easy it would be to integrate such models into our usual development process and to find out what the benefits, and difficulties, of using such models were. We will show how we used the formal models within a user-centred design process, discuss what effect they had on this process and explain what benefits we perceived from their use.

Categories and Subject Descriptors
D.2.2 [**Software**]: Software Engineering – *design tools and techniques, user interfaces.*

General Terms
Design, Human Factors, Verification.

Keywords
Formal methods, user-centred design, software design process.

1. INTRODUCTION
We approach software design with the intention of building correct, robust, usable systems. This means that for the functionality of our software we use formal methods, such as developing a specification in a formal language like Z [11] or B

[1], model-based testing [24], proof of correctness [26], refinement [8, 25] etc. This all goes towards ensuring that we build software with the correct functionality that does the right thing all of the time.

For the design of the user interfaces (UIs) to our software we rely on user-centred design (UCD) techniques [15, 16]. These may include such things as ethnographic studies [9], task analysis methods like HTA [21] or GOMs [5], prototyping [16] and user testing to ensure that the requirements of the user are central to the design, and that the users remain involved at all stages of the design process. By this we aim to ensure our UIs are both usable and complete.

Many different approaches have been taken to formalising aspects of UI design and a number of different methods and models exist for this purpose. These include work based on models such as interactors [7, 18], using formal languages such as Z [2, 10], and techniques such as model-testing [17]. Recently an increasing number of approaches use XML-like languages to create abstract views of the UI which allows them to be considered independently of any implementation [12, 19].

Some of these methods are designed to replace existing ways of developing UIs and may come with their own development environments, such as the UsiXML based tool GrafiXML [13]. Others seek to absorb the UI design process into an existing formal approach, perhaps by trying to formalize the requirements of the UI along with the system [6]. For our case study we were interested in using an approach that complemented our existing UCD approach rather than requiring us to change it. For this reason we decided to design our UI using presentation models and presentation interaction models (PIMS) [4]. These models are designed to capture information about informal design artefacts, such as prototypes, and are then used to check for correctness properties within the UI as well as to help guide the design.

The original intention behind the development of these models was to allow UI design in general, and informal design artefacts in particular, to be incorporated into a formal design process. As such, their use is not intended to make the task of UI design a formal process itself, rather the aim is to look at existing UI development methods and find ways of incorporating these into a formal software development process. The models are used to formally describe attributes of the sorts of artefacts produced during a UCD process (such as storyboards, prototypes etc.) in a manner which enables them to be incorporated into a specification and refinement software design process (in particular using the specification language Z [11]). While this is an interesting and, in our opinion, worthwhile reason for using such models, we were also interested in finding out what other

benefits their use may provide to the UI design. We were particularly interested to find out if their use assisted with designing *better* UIs (by *better* we mean UIs with desirable properties such as those described by Shneiderman [22] and which present fewer difficulties, and a better user experience, to users during usability testing and beyond). We hoped to discover whether the act of developing such models has any effect upon the design (does it in some way guide the development of the UI) and would the development of the models highlight previously undetected problems in the design even prior to testing the models within the formal process or undertaking user testing?

1.1 Terminology

In order to be consistent with previous work describing and using presentation models and PIMS [2, 3], and to distinguish between the abstractness of the UI models and actual implementations of such models, we will use the following terminology in this paper:

The main views of the UI will be called 'windows' while temporary and modal windows will be called 'dialogues'. All active controls described in the UI designs will be referred to as 'widgets'. We will describe the actions of these widgets as 'behaviours' and talk about widgets 'having certain behaviours' or 'invoking certain behaviours' to mean that when implemented they cause these actions/events to occur. When we describe the presentation model of a design we will refer to the component parts as 'PModels' and within our modular presentation interaction model we will refer to 'component PIMs' to mean each of the individual parts of the model.

1.2 Paper Outline

In the next section we will describe the software application being designed for this case study. Although our focus is on the design of the UI we will give a brief description of the requirements for the application as a whole before concentrating on the user requirements for the system. We will describe the process used to design the UI and present some of the prototypes and other informal design artefacts developed as part of this process. We will then show how the presentation models and PIMs were derived from these. We will outline the benefits we obtained from using these models, show how we were able to identify problems within the design using them, and describe how we subsequently fixed these problems. We will also discuss problems we encountered when using the models and suggest some possible solutions to these. Finally we will present some conclusions based on our experiences of designing a UI in this way.

2. CASE STUDY DESCRIPTION

The software being designed for this case study is somewhat self-referential in that it is an editor for presentation models and PIMs. The tool, called PIMed, will be used to support designers who wish to use presentation models and PIMs in their work. In this paper we focus on the requirements for the UI to the system but in this introduction to the example we will briefly outline the major requirements for the system as a whole and refer to the system requirements within this paper only as necessary.

2.1 System Requirements

The editor is to be used to create, view, edit and print presentation models and PIMs. Data should be preserved between uses of the application so that we can incrementally build up a collection of these models. Presentation models and

PIMs can either be created independently from each other, or presentation models already in the system can be used to create PIMs with the pair then becoming linked.

Information stored in the application should reflect the hierarchical and component-based nature of the models. It should, therefore, allow for declarations to be entered and stored independently from completed models and used as required. Similarly, the described presentation models should exist as both detailed, complete models in their own right as well as components within conjunctions to build up other models. It should, in this manner and also all other respects, follow the syntax and semantics given in [3].

2.2 User Requirements

We began by discussing these requirements with potential users of the system. Typically in a UCD process we may have several such "brain-storming" sessions with different groups of users, using such things as post-it notes and whiteboards to ensure a collaborative and interactive discussion process. Following this we then identified the key user requirements and started to consider how the functional requirements of the system should be represented as user requirements within the UI.

Once this initial consultation was completed we were able to create a set of top-level user requirements for the UI, which are:

- o   Create new presentation model
- o   View existing presentation model
- o   Print presentation model
- o   Edit presentation model
- o   Create new PIM from presentation model
- o   Create new PIM from scratch
- o   View existing PIM
- o   Print PIM
- o   Edit PIM

Each of these represents a key task identified by the users. The next step was to break these down further into hierarchical tasks, so, for example, the requirement to create a new presentation model is broken into the following task hierarchy:

- • Create new presentation model
    - o Add declarations
        - ▪ Add model names
        - ▪ Add widget names
        - ▪ Add behaviour names
    - o Add widget
        - ▪ Select widget name
        - ▪ Set category
        - ▪ Add behaviours
    - o Compose presentation models
        - ▪ Select presentation models
        - ▪ Link models

This hierarchical view allowed us to start to consider what the different parts of the UI may be (in terms of windows and dialogues) and what sort of actions the users will perform with the UI in order to perform these sub-tasks (*e.g.* entering text, selecting options *etc.*) Each of the initial requirements we presented was expanded into its own task hierarchy and these were then used as the basis to begin the design of the UI.

3. UI PROTOTYPES

With the information we had gathered from the previous steps we now had enough information to begin developing paper prototypes for each part of the UI. From the task analysis we had identified the need for three main parts to the UI: the 'Main' navigation window; a 'View Presentation Model' window; and a 'View PIM' window. The prototypes for the 'Main' window and 'View PIM' window are shown in Figures 1 and 2.
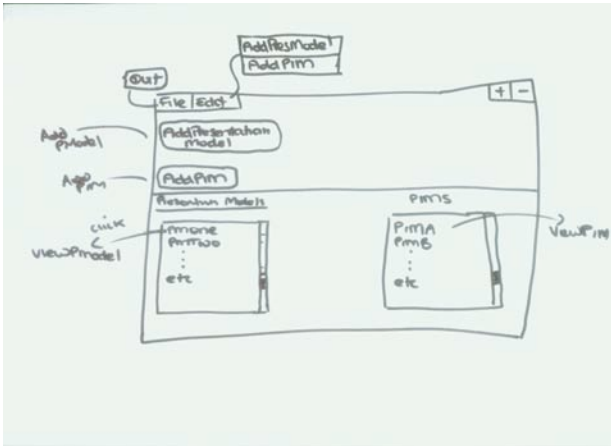

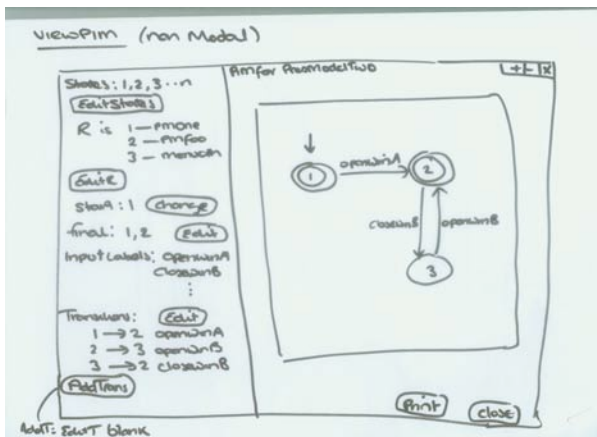**Figure 1. Design for Main window.**

Figure 2. Design for ViewPIM window.

In addition to these three main windows, there are also a number of sub-windows and dialogues (mostly modal). In total, the UI consists of 27 different windows and dialogues. For each of these we developed a prototype with various levels of annotation.

The prototypes give an idea of what the UI may look like by suggesting possible widgets and layouts. More importantly, they also give an idea of how the UI can be used. So, for example, with the prototype given in Figure 1, we can discuss with a user what happens when they interact with the different widgets and they can see how they would perform different tasks using the widgets provided. Figure 3 shows some of the prototypes that were developed for the UI spread out in order to facilitate a more complete view of the UI for the users. When we discuss the prototypes with users we can also show them how we get from one part of the UI to another by moving between the different designs and sliding the prototypes around the table (we could also achieve this by developing interaction

storyboards). As we will discuss in the next section, this information about interactive behaviour and navigation around the UI, which typically forms part of the discussions between designers and users and drives the design process, is precisely the information that is described by the formal models.
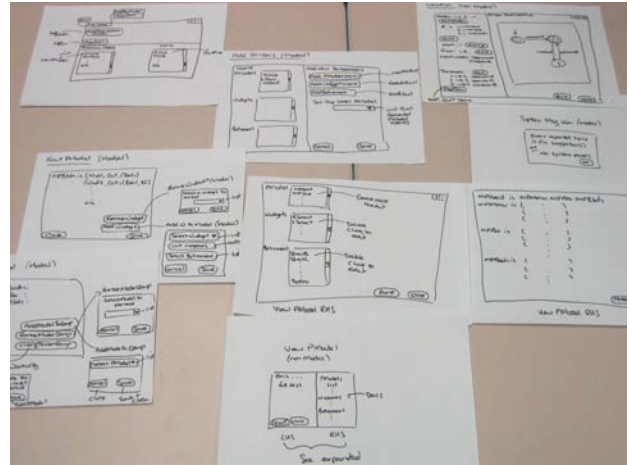

Figure 3. A selection of the paper prototypes.

Typically at this stage of the design process we would begin to show the prototypes to the users and discuss the designs to gather feedback, and we would then use this to update and refine the prototypes. We might also begin to develop computer-based prototypes of the designs that could be incrementally updated and which would form the basis of the implemented UI. This work would take place independently from the system design team (responsible for the underlying system functionality) who would be working on the formal specification of the system to ensure correctness and completeness of the specified system prior to undertaking a refinement process that would lead to the development of the system.

Rather than continue to follow our usual design process however, our next step was to start to develop the formal models of the prototypes. We describe this in detail in the next section.

4. FORMAL MODELS
4.1 Building the Presentation Model

The first part of formalising the designs was the development of a presentation model of the PIMed UI prototypes. This was done by creating component models (PModels) for each of the windows and dialogues in the design and then combining these to create a description of the overall UI.

A presentation model is a formal description of a UI design which is intended to describe the controls of that UI (in an abstract manner which includes their name and category) and their behaviour. The behaviours of the widgets of the UI may either be UI behaviours (i.e., things that control the UI's appearance or navigation such as window resizing or opening a new window) or system behaviours (things that interact with the underlying system functionality such as saving or retrieving information). As such, a presentation model reflects the way UIs may be designed or built by describing the UI in terms of the widgets which are used in UI development toolkits and which may form the basis of either a computer-based design or a prototype.

For example, the PModel for the 'View PIM' window design shown in Figure 2 is as follows:

ViewPIM is (MaxWin, ActCtrl, (UI_MaxWindow))
(MinWin, ActCtrl, (UI_MinWindow))
(CloseWin, ActCtrl, (UI_CloseViewPIM)
(PIMFrame, MulValResp, (EditState, EditTrans,
EditStart, EditFinal, EditR))
(PrintButt, ActCtrl, (Print))
(CloseButt, ActCtrl, (UI_CloseViewPIM))
(StateList, StatusDisplay, ())
(RList, StatusDisplay, ())
(SState, StatusDisplay, ())
(FSList, StatusDisplay, ())
(ILabelList, StatusDisplay, ())
(TransList, StatusDisplay, ())
(EdStateButt, ActCtrl, (UI_OpenEdStateWin))
(EdRButt, ActCtrl, (UI_OpenEdRWin))
(EdSSButt, ActCtrl, (UI_OpenEdSSWin))
(EdFSButt, ActCtrl, (UI_OpenEdFSWin))
(EdTransButt, ActCtrl, (UI_OpenEdTransWin))
(AddTransButt, ActCtrl, (UI_OpenEdTransWin))

All of the widget and behaviour names used were declared first at the start of the presentation model (following the syntax given in [3]). Each of the 27 window and dialogue prototypes are described in their own component presentation models and then the UI for the entire system is described as the concatenation of these. This is done in a modular fashion, so at the top level we have a description of the overall UI as:

PIMed is Main : ViewPIM : ViewPModel

and then each of these is subsequently built up of component presentation models, as in:

Main is MainWin : AddPMDecs : AddPIM
AddPMDecs is AddPMDecsWin : AddPMNExst :
AddWNExst : AddBExst

with each individual window described in full in its own PModel as in the 'View PIM' example above.

While we were creating the presentation models we found several things within our designs which required clarification or changes. The presentation model captures the 'narrative' of the design, that is, it describes the behaviour of the UI based on interaction with the widgets provided by that UI. On returning to the designs to create the presentation models we found that there were some examples where we were not entirely clear what the behaviour should be for a particular widget.

For example, in our design of the 'View Presentation Model' window we had annotated one of the controls, the list of PModel names, with 'double click to edit', but when we came to explain this in the presentation model we were not sure if this behaviour should be '*EditPModel*' or '*EditPModelName*', that is, we hadn't fully thought through the behaviour of this widget during the design stage. This is analogous to describing the design to a user and not being able to explain what would happen if they interacted with a particular widget as we haven't thought it through fully. Having to be explicit about the behaviour in the presentation model meant that we had to consider this properly before we could continue. This was also true of the widget name list and behaviour name list in the same window, which were similarly annotated and which similarly we hadn't fully considered.

We also found that several of the windows were unnecessarily complicated when we came to model them. The act of describing them formally made this clear and it also enabled us

to simplify them. This led to the removal of three of the dialogues totally (as we found the behaviour was either fully duplicated elsewhere or did not require a separate dialogue) and the simplification of several others. The amended designs had all the functionality of the originals but were less complicated.

4.2 Using the Presentation Models

The main purpose given for developing presentation models is to allow informal UI designs to be considered as part of a formal software design process. However, it is also possible to use the models to test designs for certain desirable properties, such as consistency and responsiveness. Before using the models to check for correctness in respect of the formal description of the system as a whole we therefore decided to use them to test for these UI properties.

To check for consistency within the UI design we analysed the behaviours and controls described. Generally, we would consider that the model represents a consistent UI if controls with the same name exhibit the same behaviour, and conversely if behaviours which are the same are exhibited by controls with the same name. This suggests a UI where a user knows what common widgets will do because they have seen similarly labelled widgets previously (such as *Print, Save,* etc.) and are not surprised by their behaviour.

One thing we did discover during this analysis was that there was a difference in the way the 'View Presentation Model' and the 'View PIM' windows could be closed. The 'View Presentation Model' window has a button called 'Close' located at the bottom of the window, whereas the 'View PIM' window has both a 'Close' button and an icon button which also performs the 'Close' behaviour. As these two windows (along with the 'Main' window) are the main windows of the UI (the user will spend most time viewing these windows and moving between them) it is likely that they would expect them to work in similar ways. Being able to close one of the windows in one way but not the other is an example of an inconsistency likely to prove annoying: a user gets used to using the icon button to close the window but finds it is not always available so becomes frustrated with the UI. We fixed this problem by the addition of an icon button to the 'View Presentation Model' window. As we were examining the prototypes for these two designs in light of this problem we also realised that both the 'Close' and 'Print' buttons were in different places on these two windows, we therefore also updated the designs so that these buttons were in the same place so that they would be quicker and easier for a user to find.

The reactivity of a UI design can be determined by looking at the categories of the widgets used. The categories of a presentation model come from the widget category hierarchy derived in [2]. There are three top-level categories: Display, Event Generator and Event Responder. These are then further broken down into sub-categories as shown in Figure 4.
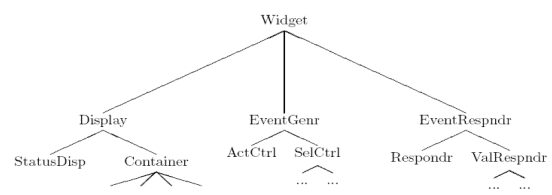

Figure 4. Part of widget category hierarchy.

Actual implementable widgets, such as radio buttons or value sliders, can be found at the leaf nodes in the hierarchy tree and the intention is that *any* widget (including custom made widgets or new widgets) can be easily positioned within the hierarchy.

Widgets which come under the *Action Control* or *Event Generator* categories are those which the user uses to initiate actions upon the UI and the system, whereas *Event Responders* and *Status Displays* are those which respond to events from the underlying system. The higher the percentage of widgets which allows the user to initiate actions, the more responsive we consider the UI to be, and we consider responsiveness to be a desirable UI property.

Analysis of the presentation model for our designs showed us that 70% of the widgets allowed the user to initiate actions while 30% were reactive to the system. This indicates a UI with a high level of user responsiveness.

The final stage of testing using the presentation models involved returning to the system specification to ensure that the behaviours described in the models are correct with respect to the specification. That is, does the UI correctly describe all of the required behaviours and so represent the same system as that given in the formal specification? This is the point at which we are able to integrate our formal and informal processes. We now have models of the designs that can be used in conjunction with the specification of the system.

There are different ways to determine whether a UI is correct with respect to the system specification. In [3] an example is given of converting the presentation model into a Z description (giving a specification of the design) and then using standard refinement simulation techniques to show that the UI design refines the specification. For smaller, less safety-critical, systems it is also possible to manually inspect the system specification and extract the operations which are to be made available to the user via the UI (using the information from gathering user requirements and task analysis in conjunction with the system requirements). We can then develop a relation between UI operations of the specification and behaviours of the presentation model to ensure that all requirements are met. For our case study we followed the latter procedure.

As an example of this, the specification for our system relating to the PIM editing functionality contains the following Z schemas:

$$UserOp\_EditPIM \; \widehat{=}$$
$$UpdateStateList \; \vee \; UpdateStartState \; \vee \; UpdateFinalState \; \vee$$
$$UpdateTransitions \; \vee \; UpdateRelation$$

$$\begin{array}{|l}
\underline{UpdateStartState}\rule{0pt}{0pt}\hspace{3cm} \\
\Delta PIMDescription \\
newss : STATE \\
\hline
startstate' = newss \\
finalstate' = finalstate \\
allstates' = allstates \\
transitions' = transitions \\
relation' = relation
\end{array}$$

We then include in our relation a mapping between this system operation and the corresponding UI behaviour:

$$UpdateStartState \rightarrow EditStart$$

We extend the relation to include all system operation and UI behaviour pairs.

We found two problems during this process. First, we discovered that one of the dialogues we had included in our initial design list (developed from the task analysis) had somehow been overlooked during the design stage and there was no prototype for it, and therefore no presentation model. This was discovered because there were a number of operations in the specification which had no related behaviour in the presentation model. Second, we discovered a problem with the 'Print' operations. The system specification described two print operations, one which allowed for the printing of a viewed presentation model and one which allowed for the printing of a viewed PIM. The presentation model however has a single behaviour relating to printing which is called just 'Print'. We could see from this that the designs were not precise enough in this respect and the 'Print' behaviour should in fact be split into two separate behaviours, 'Print presentation model' and 'Print PIM'.

This completed the testing of the presentation model. We amended the designs based on our discoveries and updated the presentation model accordingly. We were then able to move on to the development of the next formal model, the PIM.

4.3 Building the PIM

A PIM consists of a 6 tuple (Q, $\sum$, δ, q_0, F, R) where Q is a set of states, $\sum$ is a set of input labels, δ is a transition function, q_0 is the start state, F is a set of accepting states and R is a relation between states and presentation models. A PIM is used to show the dynamic behaviour of the UI which allows the user to switch between different views, windows and dialogues. In the PIM each component PModel within the presentation model is represented by at least one state and transitions between states are labelled with the behaviour which causes that transition.

There are several different ways of visualizing PIMs. For our case study we decided to use the visual representation based on the µCharts language [20]. This provided us with a visual representation that was modular and easy to view, and which made the modality of windows and dialogues explicit. The syntax of µCharts allows us to embed multiple states within a single state (called decomposition) creating an abstraction of the model which is small and easy to view by itself, and which enables us to examine the decomposed states in more detail as required.

Using our updated presentation model we developed the PIM by creating the visual representation first, based on what we knew about the UI behaviour and modality of the design from both the presentation model and the designs themselves.

Figure 5 shows the top level view of the PIM for our application, and Figure 6 shows the detail of the ViewPIM state.
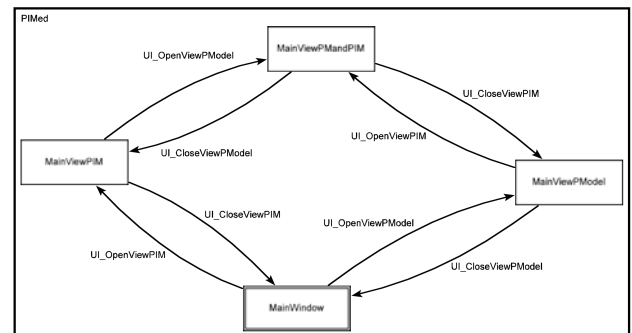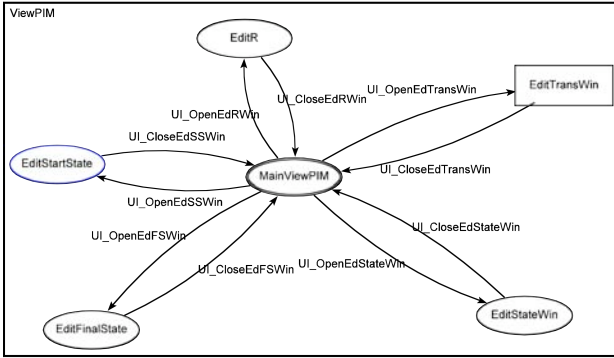

Figure 5. Top-level PIM.

Figure 6. PIM detail for ViewPIM.

Once again we found that the process of creating the formal model exposed some problems in the design. The completed PIM contained a state (representing the *EditPModelName* PModel) which was not linked to any other state in the PIM. This indicated that this dialogue (and therefore all behaviours within the dialogue) was unreachable within the design. This is an example of why the PIM is required as this sort of problem cannot be detected by the presentation model alone. To fix the problem we added another widget to the 'View Presentation Model' window to open this dialogue and updated our models accordingly. The new PIM contained no unlinked states.

Another problem we discovered was that our initial PIM was nondeterministic. The model of the 'View Presentation Model' window contained two transitions with the same label, as shown in Figure 7.
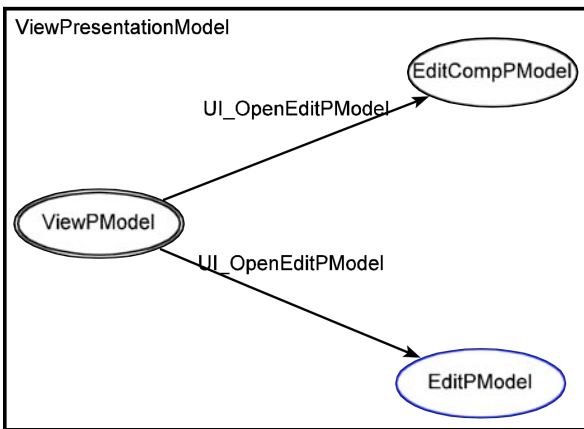

Figure 7. Non determinism in PIM.

This suggests that in the 'View Presentation Model' the same behaviour can lead to two different resulting states, that is, it is nondeterministic. When the user interacts with the widget that invokes this behaviour they cannot predict what will happen. This was not the intention of the design. In fact what should happen is that the *type* of PModel being accessed (either a composite presentation model or non-composite model) is determined, and then one of the two possible dialogues opens depending on that type. The behaviour of the widget should actually be to invoke a system behaviour which checks the type, and the system behaviour then invokes the correct UI behaviour. In this instance we can see that our design is underspecified

We updated the presentation model, so that the widget description changed from:

(PMNameList, SValSel, (UI_OpenEditPModel,

UI_OpenEditCompPModel))

to:

((PMNameList, SValSel, (CheckPMType))

However, once this change was made and we updated the PIM, we were then faced with a different problem as this part of the PIM now contained unlinked states as we show in Figure 8.
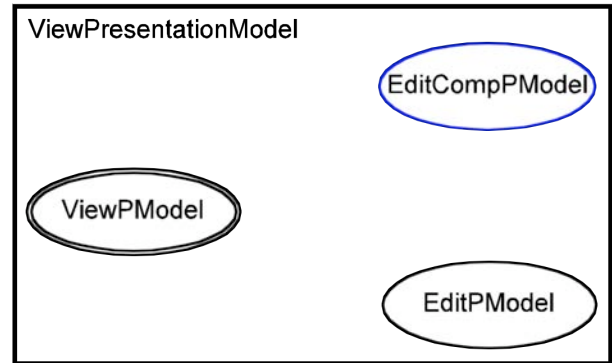

Figure 8. PIM with unreachable states.

Because the behaviour which opens this dialogue is now *hidden* within the system behaviour (*i.e.* is no longer a part of the presentation model) it appears that the states representing this dialogue are now unreachable. In fact what we really need is a combination of the system and UI behaviours, such that the presentation model description becomes:

((PMNameList, SValSel, (CheckPMType:UI_OpenEditPModel,

CheckPMType:UI_OpenEditCompPModel))

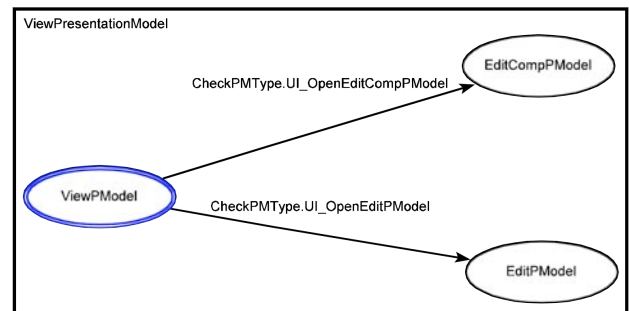which gives us the PIM of Figure 9.

Figure 9. Corrected PIM.

The syntax and semantics of presentation models [4] do not provide for composition of behaviours in this way. However, from this is example it is clear that there is a requirement for the language to support this either by joining behaviours, as we have done, or by making explicit system driven UI behaviour in some other way. We suggest that one way to achieve this is by extending the composition operator (:) defined for joining composite PModels, to include (:) for composition of behaviours.

4.4 Testing the PIM
Once the PIM was complete we were able to test for total reachability and deadlock within the modelled UI (total reachability means that we can get to any state in the PIM from any other state, it is therefore a stronger requirement than just

ensuring all states can be reached from the start). We performed the deadlock test using a manual walk-through procedure to check that in each of the individually modelled component PIMs we could either reach a designated final state, or (in the case of a decomposition) could return to the parent part of the PIM (the model containing the decomposed state). We found two states where deadlock occurred. Both of these states represented modal dialogues where widgets exhibiting 'close' behaviour had been omitted, so there was no way to return to the parent state and they were not themselves final states.

Once these errors had been fixed and the designs, presentation model and PIM updated, we found the design to be deadlock free. Using a manual walkthrough again we were then able to test for total reachability and found that the design was indeed totally reachable.

This concluded our testing of the UI designs using the formal models.

5. COMPLETING THE DESIGN
5.1 Finalising the UI and System
We had now reached a stage where we were sure that the designs we had created for the UI were both correct in terms of the system and user requirements as well as holding certain properties of both design and behaviour.

In order to complete the designs and move on to the implementation stage we now returned to our more traditional UCD design process involving user feedback and iterative updating of the designs. At the same time, the work on the underlying system specification continued so that an implementable system could be derived via the refinement process. Both of these activities are currently still ongoing.

Research describing presentation models and PIMs provides ways of ensuring (via different types of equivalences) that updates to the UI designs preserve the correctness properties previously determined [3] and it is our intention to follow these procedures so that our final application has been designed in accordance with the described process for these models.

5.2 Future Work
Once the UI design and system design processes are complete we will be ready to implement the final system. The system will then undergo usability and functionality testing to ensure that it is correct, robust and usable.

In terms of the formal development process we follow there is much more to be said about how we move forward to actually implement the designs described. Composing the system and UI and the concept of refinement and its application to UI development is an area which we are very interested in. We are currently pursuing several ideas in this area.

6. CONCLUSIONS
Our experience of using these two particular formal models for UI design (namely presentation models and PIMs) has shown us that not only is it possible to incorporate such models into our UCD development process for UIs, but that we can do so with little time/effort overhead and gain a number of benefits from doing so.

Although the intention behind the development of these models was to allow the UI design to be incorporated into a formal software development process in a way which did not change the nature of user-driven, user-centred design, we were more

interested to find out what, if any, benefits they might give to the actual UI design process itself.

We found that the act of modelling our designs made us consider certain aspects more carefully than we had when using just paper prototyping methods and we were able to pick up problems earlier in the design process than we would have otherwise. This proved to be one of the major benefits of using the models within our work. Early detection of errors means that when we reach the stage of usability testing we have already removed some potential problems, making it easier for testers to interact more fully with the system and find more deep-rooted problems than they would otherwise. Nielsen describes this as users being able to delve more deeply into applications if they are not overwhelmed by small problems [14]. In addition, fixing problems at the paper design stage is both quicker and cheaper than changing implemented, or partially-implemented, systems and we can immediately ensure that such changes do not introduce new problems.

From the perspective of the underlying system development, being able to consider UI aspects (by way of the models) at an early stage meant that we thought more carefully about how we specified operations that were required by the user so as to produce less overhead in the UI. For example, when describing operations to change data within representations of the models we gave more consideration to how we would integrate this with the UI functionality. While this can be seen as a benefit of any method which considers the UI within the formal process, in this case it does so without requiring the UI designers to follow the formal process, but rather considers the existing designs by way of the models.

In conclusion, we found that using the models was helpful and provided a benefit to the design process which justified their use. We did discover some problems with the models which required us to extend the original syntax, but this did not unduly hinder the process. It is clear that such models cannot (and should not) replace any of the tasks we generally undertake within a UCD process (nor is it intended they do so), but we are satisfied that they can provide a benefit over and above the stated aim of integrating a formal specification and refinement based approach to system development with a user-centred UI design approach.

7. REFERENCES
[1] Abrial, J. *The B-Book: Assigning Programs to Meanings*. Cambridge University Press, New York, 1996.

[2] Bowen, J. *Formal Specification of User Interface Design Guidelines*. MSc. Thesis, University of Waikato, Hamilton, NZ, 2005.

[3] Bowen, J., and Reeves, S. Formal refinement of informal GUI design artefacts. In *Proceedings of the Australian Software engineering conference (ASWEC 2006)*. (Sydney, 2006). 2006.

[4] Bowen, J., and Reeves, S. Formal models for informal GUI designs. In *First international workshop on Formal methods for interactive systems (FMIS 2006)*. (Macau SAR China, 31 October 2006). Electronic Notes in Theoretical Computer Science, Elsevier, Amsterdam, 2006.

[5] Card, S., Moran, P., and Newell, A. *The Psychology of Human Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1983.

[6] Courtney, A. Functionally modeled user interfaces. In *Proceedings of Design, specification and verification of*

interactive systems (DSV-IS 2003). Springer-Verlag, Berlin, 2003, 107-123.

[7] Duke, D., Fields, B., and Harrison, M. A case study in the specification and analysis of design alternatives for a user interface. *Formal Aspects of Computing, 2,* 11 (1999), 107-131.

[8] Henson, M., and Reeves, S. Investigating Z. *Journal of Logic and Computation 10,* 1 (2000), 1-30.

[9] Hughes, J., King, V., Rodden, T., and Anderson, H. Moving out from the control room: Ethnography in system design. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work. (CSCW 1994).* ACM Press, New York, NY, 1994, 429-439.

[10] Hussey, M., MacColl, I., and Carrington, D. *Assessing Usability from Formal User-Interface Designs,* Technical TR00-15, University of Queensland, 2000.

[11] ISO/IEC 13568. *Information Technology - Z Formal Specification Notation - Syntax, Type System and Semantics.* Prentice-Hall, 2002.

[12] Lepreux, S., Vanderdonckt, J., and Michotte, B. Visual design of user interfaces by (de)composition. In *Proceedings of Design, specification, and verification of interactive systems (DSV-IS 2006).* Springer-Verlag, Berlin, 2006, 150-170.

[13] Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., and Lòpez-Jaquero, V. USIXML: A language supporting multi-path development of user interfaces. In *Proceedings of EHCI/DS-VIS 2004.* Kluwer Academic, 2004, 200-220.

[14] Nielsen, J., and Landauer, T. A mathematical model of the finding of usability problems. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI 1993).* ACM Press, New York, NY, 1993, 206-213.

[15] Norman, D., and Draper, S. (eds). *User Centered System Design: New Perspectives on Human-Computer Interaction.* Hillsdale, NJ, Lawrence Erlbaum Associates, 1986.

[16] Norman, D. *The Psychology of Everyday Things.* Basic Books, New York, 1988.

[17] Paiva, A., Tillmann, N., Faria, J., and Vidal, R. Modeling and testing hierarchical GUIs. In *Proceedings of ASM 2005.* (Universite de Paris, Paris, 2005). 2005.

[18] Paternò, F., Sciacchitano, M., and Lowgren, J. A user interface evaluation mapping physical user actions to task-driven formal specification. In *Proceedings of Design, specification and verification of interactive systems (DSV-IS 1995).* Springer-Verlag, Berlin, 1995, 155-173.

[19] Puerta, A., and Eisenstein, J. XIML: A common representation for interaction data. In *Proceedings of the 7th international conference on Intelligent user interfaces (IUI 2002).* ACM Press, New York, NY, 2002, 214-215.

[20] Reeve, G. *A Refinement Theory for μCharts.* Ph.D Thesis, University of Waikato, New Zealand, 2005.

[21] Shepherd, A. Analysis and training in information technology tasks. In D. Diaper (ed.), *Task analysis for human-computer interaction.* Ellis Horwood, Chichester, 1989, 15-55.

[22] Shneiderman, B. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (3rd edition).* Addison Wesley Longman Inc, 1998.

[23] Snyder, C. *Paper Prototyping: The Fast and Easy Way to Design and Refine User Interfaces.* Morgan Kaufmann, 2003.

[24] Utting, M., and Legeard, B. *Practical Model-Based Testing: A Tools Approach.* Morgan-Kaufmann, 2007.

[25] Wirth, N. Program development by stepwise refinement. *Communications of the ACM, 14,* 4 (1971), 221-227.

[26] Woodcock, J., and Davies, J. *Using Z: Specification, Refinement and Proof.* Prentice Hall, New York, 1996.