# Chapter 6
# Combining Models for Interactive System Modelling

**Judy Bowen and Steve Reeves**

**Abstract** Our approach for modelling interactive systems has been to develop models for the interface and interaction which are lightweight but with an underlying formal semantics. Combined with traditional formal methods to describe functional behaviour, this provides the ability to create a single formal model of interactive systems and consider all parts (functionality, user interface and interaction) with the same rigorous level of formality. The ability to convert the different models we use from one notation to another has given us a set of models which describe an interactive system (or parts of that system) at different levels of abstraction in ways most suitable for the domain but which can be combined into a single model for model checking, theorem proving, etc. There are, however, many benefits to using the individual models for different purposes throughout the development process. In this chapter, we provide examples of this using the nuclear power plant control system as an example.

## 6.1 Introduction

Safety-critical interactive systems are software or hardware devices (containing software) operating in an environment where incorrect use or failure may lead to loss, serious harm or death, for example banking systems, ATMs, medical devices, aircraft cockpit software, nuclear power plant control systems and factory production cells. Avoiding such errors and harm relies on the systems being developed using robust engineering techniques to ensure that they will behave correctly and also that they can be used successfully.

Developing suitable interfaces for safety-critical systems requires two things. First, they must be usable in their environments by their users—i.e. they must be

J. Bowen (✉) · S. Reeves
University of Waikato, Hamilton, New Zealand
e-mail: jbowen@waikato.ac.nz

S. Reeves
e-mail: stever@waikato.ac.nz

B. Weyers et al. (eds.), *The Handbook of Formal Methods in Human-Computer Interaction*, Human-Computer Interaction Series,
DOI 10.1007/978-3-319-51838-1_6

developed using a sound user-centred design (UCD) process and following known HCI principles. Secondly, we must be able to verify and validate the user interface and interaction with the same rigour as the underlying functionality. While we can (we hope) assume the former, the latter is harder and requires us to develop suitable techniques which not only support these requirements but which will also be useful (and used) by the interface developers of such systems.

We have developed modelling techniques for the user interface (UI) and interactivity of a system which take as a starting point typical informal design artefacts which are produced as part of a UCD process, e.g. prototypes (at any level of fidelity), scenarios and storyboards. In addition to the interface modelling techniques, we also have mechanisms for combining these models with more traditional functional specifications (which deal with the requirements for the system behaviour) in order to be able to reason about the system as a whole.

In the rest of this chapter, we provide details of the models and notations we use to describe the different parts of an interactive system. We also discuss how these can be combined into a single model to give a single, formal 'view' of the entire system. This cohesive model allows us to consider important properties of the system (which generally involves proving safety properties and ensuring the system as specified will behave in known, safe ways at all times) which encompasses aspects of the UI and interaction as well as functional behaviour. At the same time, however, the individual component models used to create this single model have their own benefits. They give us the ability to consider different aspects of the system (either specific parts or different groups of behaviours, for example) using different levels of abstraction or different modes of description to suit the domain. Essentially, they provide us with a set of options from which we can select the most appropriate model for a given use. Because these component models are developed as part of the design process and form part of the overall system model, we essentially get this 'for free' (that is without additional workload).

We use the nuclear power plant control system as an example to show how these models can be used independently, as well as in combination, to consider different properties of interest during a development process.

## 6.2 Related Work

In early years, formal methods were developed as a way of specifying and reasoning about the functionality of systems which did not have the sorts of rich graphical user interfaces provided by today's software. Some formal methods were used to reason about interaction properties, e.g. (Jacob 1982; Dix and Runciman 1985), but as user interfaces evolved and became more complex, and the importance of their design became increasingly obvious, the disciplines of HCI and UCD evolved to reflect this. However, these two strands of research—formal system development and UI design research—remained primarily separate and the approaches used within them were also very different. On the one hand were formal languages and notations based on mathematical principles used to formally reason about a system's behaviour via specification, proof, theorem proving, model checking, etc., while on the other were design processes targeted at usability based on psychological principles and involving shared, informal design artefacts, understanding end-users and their tasks, evaluation of options and usability, etc.

This gap between the formal and informal has been discussed many times, notably as far back as 1990 by Thimbleby (1990). Numerous approaches have been taken over the years to try and reduce the gap between the two fields, particularly as the need to reason about properties of UIs has become increasingly necessary due to the prevalence of interactive systems in general and the increase in safety-critical interactive systems in particular. We can generalise key works in this area into the following categories:

- development of new formal methods specifically for UIs (Puerta and Eisenstein 2002; Courtney 2003; Limbourg et al. 2004)
- development of hybrid methods from existing formal methods and/or informal design methods. (Duke and Harrison 1995; Paternò et al. 1995)
- the use of existing formal methods to describe UIs and UI behaviour (Harrison and Dix 1990; Thimbleby 2004)
- replacing existing human-centred techniques with formal model-based methods. (Hussey et al. 2000; Paternò 2001; Reichart et al. 2008)

These, and other similar works, constitute a concerted effort and a step forward in bringing formal methods and UI design closer together. However, in many cases, the resulting methods, models and techniques continue to either retain the separation of UI and functionality in all stages of the development process, or seek to integrate them by creating new components within the models which combine elements of both in a new way (Duke et al. 1994, 1999).

When we first began to consider the problem and investigate and develop modelling techniques for interactive systems, we had a number of criteria, including a desire to model at the most natural level of granularity (describe the existing components as they appear) as well as come up with an approach that could fit with both formal and HCI methodologies. In contrast to other approaches, our starting point is that of design artefacts (of both interface and system) which may be developed separately, and perhaps at different times, during the development life cycle. Unlike more recent work such as (Bolton and Bass 2010), we do not include models of user behaviour or consider the UI in terms of the tasks performed or user goals. Rather, we model at a higher level of abstraction which enables us to consider any available behaviours of the system via the UI rather than constraining this to expected actions of the user based on predefined goals.

So, just as the interfaces we design must be suitable for their systems and users, the models we use to reason about these interactive systems must also be usable and useful to their users (designers, formal practitioners, etc.). Rather than having an expectation that UI designers should throw away their usual, and necessarily informal,design processes in favour of a new formal approach, we are of the opinion that these should be incorporated into our approach rather than being replaced by it.

We start with the assumption that traditional, informal user-centred design practice has been used, as usual, to produce many artefacts relating to the 'look and feel' of the system. These will include design ideations such as prototypes, scenarios, and storyboards and will be the result of collaboration between designers and end-users, as well as other stakeholders. Keeping this point in mind is very important when it comes to understanding the modelling techniques that we discuss in this chapter.

In the same way that we assume the interface has been developed using well-known and appropriate design methodologies, we similarly assume that the design of the functional behaviour of the system has likewise followed a rigorous development process which leads to the development of formal artefacts such as a system specification or some other formal model of the system's behaviour. Given that we are interested in safety-critical systems, where erroneous behaviour can lead to injury or even death, this assumption seems a reasonable one.

In order to be able to reason about all parts of the system and not just the functional behaviour, we need a way of considering the informal design artefacts with the same level of formality as the functional specification. To do this, we create formal models of these design artefacts. By creating a formal representation of the informal design artefacts (in essence creating a different abstraction of the UI which happens to be in a formal notation), we gain the ability to combine the two representations (system and UI) into another model which gives a single view of the system as a whole. This then allows us to reason about the interactive system in a way which captures all of the information (both UI and system) and perform activities such as model checking and theorem proving to show that all parts of the system will have the properties we desire (Bowen and Reeves 2008, 2013).

We next give an overview of the different models used in our process and then go on to provide examples of these in use for the nuclear power plant example.

## 6.3 Background

Our starting point for the modelling process is to consider the two main components of the system (functionality and interactivity—which includes the interface) separately. This separation of concerns is, in some sense, an artificial division. While it is often the case that one group within a design team may focus on appearance and look and feel of the interface while another focusses on core functionality, we are not aiming to reflect the divisions and complexities that exist within design groups. Rather, the separation allows us to consider different parts of the system in different, and appropriate, ways, and provides the basis for our use of different levels of abstraction to describe different components within that system. Such separation is a common approach taken in this type of modelling, although it can be done at differing levels of granularity. For example in Chap. 5, we see functional behaviours described as components which are then associated with related widgets and composed using channels, whereas Chap. 14 uses a layered approach where different parts of the system and model are described in different layers.

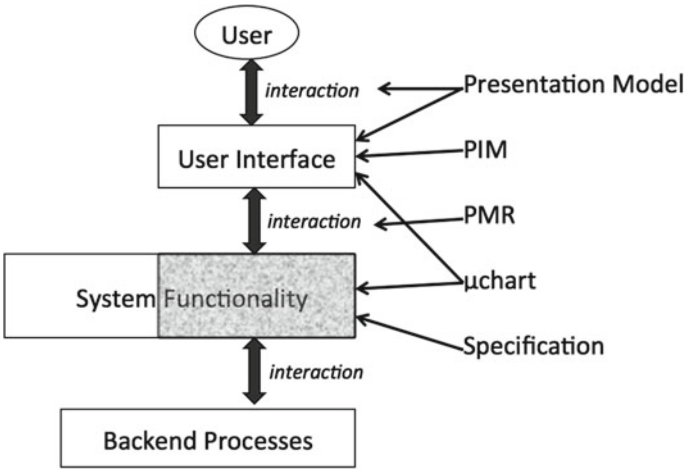

**Fig. 6.1** Overview of model components

We rely on a combination of existing languages and models to specify the functionality. Typically, for us, this means creating a Z specification (ISO/IEC 13568 2002; Henson et al. 2008) and/or $\mu$charts (Reeve 2005) to reason about functional and reactive behaviours, although any similar state-based notation could be substituted for Z. The ProZ component of the ProB tool[1] is used for model checking the specification, or we can use Z theorem provers such as Proofpower.[2] These allow us to ensure that the system not only does the right thing (required behaviour) but also does not do the wrong thing (behaviour that is ruled out), irrespective of the circumstances of use. Figure 6.1 gives an overview of how each of the models relates to the system under consideration.

The presentation model describes the user interface elements that are present but not the layout of the interface. It also describes the types of interaction each widget exhibits (which suggests how a user interacts) and labels the behaviour associated with that widget. The presentation interaction model (PIM) also describes the user interface but at a higher level of abstraction which hides the general behaviours and widgets of the interface and focusses on the navigation that is possible through the different modes/windows, etc.

The presentation model relation (PMR) relates some of the labels of widget behaviours to operations in the specification, and as such gives a very high-level view of the interaction between system and UI. Not all functionality is expressed via the user interface (the user can only directly access certain behaviours) and hence the split in the system functionality box where the grey sections are those functions which do relate to user actions.

[1]http://stups.hhu.de/ProB/.

[2]http://www.lemma-one.com/ProofPower/index/index.html.

$\mu$Charts is used as an alternative notation for PIMs, and so describes the same properties of the user interface, but can also be used to describe aspects of system functionality to show cause and effect behaviours. The specification on the other hand describes only the system functionality in terms of what can be achieved and abstracts away all details of how it might be achieved.

Finally, the models can be combined into single representation which gives a low-level description of the what and how of the interface and system and their interaction. We do not explicitly model the user at all in our approach nor any back-end processes such as database connectivity and networks. In the next section, we describe each of the models in more detail using examples from the nuclear power plant example.

### *6.3.1 Presentation Model*

This is a behavioural model of the interface of a system described at the level of interactive components (widgets), their types and behaviours. We group the widgets based on which components of the interface (e.g. windows and dialogues) they appear in (or in the case of modal systems, which modes they are enabled in). The presentation model can be derived from early designs of interfaces (such as prototypes and storyboards), final implementations, or anything in between. As such they can be produced from the sorts of artefacts, interface designers are already working with within a user-centred design process. The presentation model does not in any sense replace the design artefacts it describes. Rather, we use it as an accompanying artefact that provides a bridge between informal designs and formal models. So the presentation model describes the 'meaning' of the interface (or interface design) in terms of its component widgets and behaviours, but the layout and aesthetic attributes are contained in the visual artefacts that the presentation model is derived from.

It is important to appreciate that we do not require widgets to be (only) buttons or checkboxes or menu items, etc. Widgets are any 'devices' through which interaction can occur. For example, sensors attached to parts of a physical system would be widgets: as the physical system evolves, the sensors would note this evolution and, in response, the system would move between states. In this sort of system, we might simply describe a collection of sensors together with the behaviours (both functional and interactive) that their triggering or their readings cause in the system. Thus, our idea of 'interface' is very general.

Each window, dialogue, mode or other interactive state of the system is described separately in a component presentation model (*pmodel*) by way of its component widgets which are described using a triple:

```
widget name, widget type, (behaviours)
```

The full interface presentation model is then simply the collection of the *pmodels*, and describes all behaviours of the interface and which widgets provide the behav-

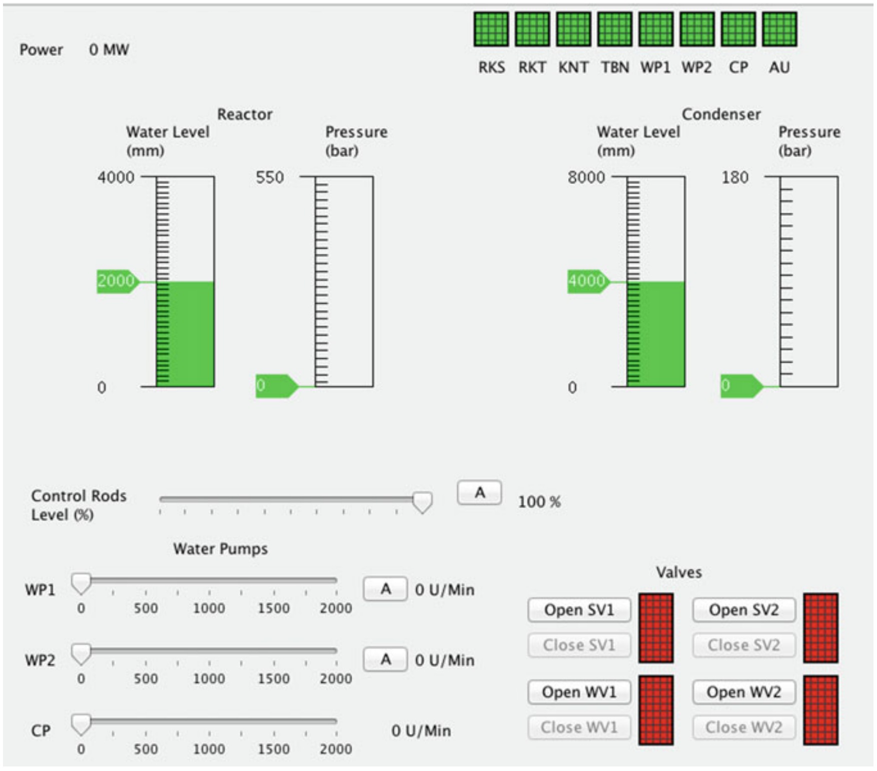

**Fig. 6.2** Nuclear plant control example interface

iours. Behaviours are split into two categories, interactive behaviours (I-behaviours) are those which facilitate navigation through the interface (opening and closing new windows, noting the change of a sensor state, etc.) or affect only presentational elements of the interface, whereas system behaviours (S-behaviours) provide access to the underlying functionality of the system (the grey part in Fig. 6.1).

The syntax of the presentation model is essentially a set of labels which we use to meaningfully describe the attributes of the widgets. Consider the interface provided as part of the nuclear power plant example, which we repeat in Fig. 6.2. Each of the widgets is described by a tuple in the presentation model, so for example the power output label at the top left of the display is:

```
PowerDisplay, Responder, (S_OutputPower)
```

The category assigned is that of ‘Responder’ as this widget displays information to the user in response to some inner stored value (which keeps track of the current power level). As such, the behaviour it responds to is a system behaviour which outputs whatever that power level currently is and hence has the label 'S_OutputPower'. The WP1 slider on the other hand is described as follows:

```
WP1Ctrl, ActionControl, (S_IncWaterPressure1,
                          S_DecWaterPressure1)
```

The category 'ActionControl' indicates it is a widget which causes an action to occur (i.e. the user interacts with it to make something happen), which in this case is to change the value of the water pressure either up or down. Again, these are behaviours of the system and so are labelled as S-behaviours.

Once we have labelled all of the widgets in the UI, we have an overview of all the available behaviours that are presented to a user by that UI. We subsequently give a formal meaning to these labels via the presentation interaction model (PIM) and presentation model relation (PMR) which we describe next.

### *6.3.2 Presentation Interaction Model*

The presentation interaction model (PIM) is essentially a state transition diagram, where *pmodels* are abstracted into states and transitions are labelled with I-behaviours from those *pmodels*. As such, the PIM gives a formal meaning to the I-behaviours as well as provides an abstract transition model of the system's navigational possibilities. The usual 'state explosion' problem associated with using transition systems or finite state automata to model interactive systems is removed by the abstraction of *pmodels* into states, so the size of the model is bounded by the number of individual windows or modes of the system. While the presentation model describes all available behaviours, the PIM describes the availability of these via the user's navigation. For example, in a UI with multiple windows, the user may be required to navigate through several of these windows to reach a particular behaviour. The nuclear power plant example of Fig. 6.2 has only a single, static UI screen, and as such, the PIM is a single-state automaton. We show later how this changes when we extend the example to have multiple windows constraining behaviour in the case of the emergency scenarios.

A PIM can also be used to consider aspects such as reachability, deadlock and the complexity of the navigational space (via the lengths of navigational sequences). Considered formally, the role of the PIM is to inform us what the allowable sequences of Z operations are for the interactive system that we are modelling. This allows us to make the Z definitions somewhat simpler since we do not have to introduce an elaborate system of flags and consequent preconditions in order to disallow the use of Z operations when the interactivity does not allow them: the PIM handles all of this, and what Z operations are allowed at any point in the interactivity is given by which widgets have behaviours (that are given by the Z operations) at that point.

### *6.3.3 Presentation Model Relation*

Just as the PIM gives meaning to the I-behaviours of the presentation model, the presentation model relation (PMR) does the same for the S-behaviours. These behaviours represent functional behaviours of the system, which are specified in the formal specification. The PMR is a many-to-one relation from all of the S-behaviours in a presentation model to operations in the specification. This reflects the fact that there are often multiple ways for a user to perform a task from the UI, and therefore, there may be several different S-behaviours which relate to a single operation of the specification.

So, in order to understand what an S-behaviour label represents, for example the behaviour label *S_IncWaterPressure1* from the presentation model tuple above, we identify from the PMR the name of the operation in the Z specification that it represents

$$S\_IncWaterPressure1 \mapsto IncreaseWaterPressure$$

This tells us that in the formal specification is an operation called 'IncreaseWater-Pressure' which specifies what effect this operation has on the system. The specified operation then gives the meaning to this behaviour label. We give a larger example of the PMR for the nuclear power plant example later.

### *6.3.4 Specification*

The formal specification of the system provides an unambiguous description of the state of the system and the allowable changes to that state provided by the operations. Many different formal languages exist for such a specification (e.g. VDM, Z, B, Event-B and Object-Z to name but a few), but for our approach, we rely on the Z specification language which is based on set theory and first-order predicate logic. In Z, we typically give a description of the system being modelled which is based on what can be observed of it, and then the operation descriptions show how (the values of) what is observed change. The operations are guarded by preconditions which tell us under what circumstances they are allowed to occur (i.e. based on given values of observations or inputs) and the postcondition defines which observations change and which do not when the operation occurs as well as describing any output values.

The specification of the nuclear power plant example, therefore, is concerned with observations relating to the items such as reactor pressure and water levels, condenser pressure and water levels, power output, speed of the pumps and status of the valves. A Z specification can be used with theorem provers to prove properties over the entire state space of the system (for example to show that a certain set of observations is never possible if it describes an undesirable condition) and can also be used with model checkers to examine the constrained state space for things such as safety conditions.

### 6.3.5 *μCharts*

In addition to the Z specification, we also use the visual language, $\mu$Charts (Reeve 2005; Reeve and Reeves 2000a, b) (a language used to model reactive systems). PIMs can also be represented as $\mu$charts, which provide additional benefits over a simple PIM (including the ability to compose specific sets of behaviours in different charts via a feedback mechanism and embed complex charts into simple states in order to 'hide' complexity) (Bowen and Reeves 2006a).

$\mu$Charts is based on Harel Statecharts (Harel 1987) and was developed by Philipps and Scholz (Scholz 1996; Philipps and Scholz 1998). $\mu$Charts was subsequently extended by Reeve (2005), and we use his syntax, semantics and refinement theory. $\mu$Charts has a simpler syntax than Statecharts (and in some sense can be considered a 'cut-down' version) and it also has a formal semantics. $\mu$Charts and Statecharts differ in terms of synchrony of transitions (we imagine a clock ticking and a step happening instantaneously at each tick), step semantics and the nature of the labels on transitions. Labels on transitions in $\mu$charts (note, we refer to the language as $\mu$Charts and the visual representations as $\mu$charts) are of the form $guard/action$, where guards are predicates that trigger a transition if they are true, and also cause actions to take place. For example, if a guard is simply a signal $s$, then the presence of $s$ in the current step makes the guard true (think of the guard as being the predicate 'the signal $s$ is present'). An example of an action is 'emit the signal $t$ in the current step'. Guards are evaluated and actions happen instantaneously in the same single step; thus, the emission of a signal from one transition which is a guard to another transition results in both transitions occurring in the same step.

The $\mu$Charts language includes several refinement theories which in turn gives us refinement theories for PIMs. The trace refinement theory for $\mu$Charts is particularly useful as it can be abstracted into a much more lightweight refinement theory for interfaces based on contractual utility (Bowen and Reeves 2006b). The semantics of $\mu$Charts is given in Z, and there is a direct translation available (via an algorithm and tool) from a $\mu$chart to a Z specification (Reeve and Reeves 2000b) and this in turn means we have an algorithm and means to turn a PIM into a Z specification (Bowen and Reeves 2014).

### 6.3.6 *Combining the Models*

The models of functionality (specification) and interactivity (presentation model and PIM) are already coupled via the PMR. This gives us a model which combines the conventional use of Z to specify functionality together with the more visually appealing use of charts for the interactivity.

However, we can also combine the models in a way that leads to a single model, all in Z, of the entire system. This gives us the ability to, for example, create a single model of all parts of an interactive system (i.e. interactivity and underlying functionality) that can then be used to prove safety properties about that system which might relate to functional constraints, interface constraints or both (Bowen and Reeves 2013). It might also be used as the basis for refinement, ultimately into an implementation, via the usual refinement theories for Z (Derrick and Boiten 2014; Woodcock and Davies 1996).

We do this single-model building by using the Z semantics of $\mu$Charts and by expressing the PMR as a Z relation. This is then combined with the formal specification of the functionality of the system, giving a single model where the transitions representing the PIM are used to constrain the availability of the operations. So if an S-behaviour given in the presentation model is only available in one state of the UI, this is represented in the new combined Z specification as a precondition on the related operation. Recently, we have also given a simplified semantics for the creation of a single specification from the models to reflect models of modal devices, where the PIMs typically do not use the full expressiveness of $\mu$Charts (Bowen and Reeves 2014).

## 6.4 The Nuclear Power Plant Case Study

We now consider the case study and give concrete examples of the modelling techniques described above. From the general overview of the functionality of the nuclear power plant given in the case study (along with several assumptions to fill in the gaps), we can generate a Z specification of the desired functionality. This Z specification gives us a description of the observable states of the system (i.e. its state space) as a Z state schema, along with the operations that can change these states (i.e. move us around the state space). As such, it formally specifies behaviours that can be exhibited by the system.[3] Note that we have simplified the value types for pressure and water levels to natural numbers for convenience of specification.

The observable states of the system can be captured by observing all of the parameters that are described in the case study brief along with any known constraints. For example, we know that for the state of the reactor to be considered 'safe', then the reactor pressure must be no more than 70 bar while the core temperature remains below the maximum value of 286 °C. So, we would have to be able to observe these values, and ensure that they are within the required limits, in order to be able to say that we are describing an allowed state of the system.

The state space of the system is given by a state schema like *ReactorControl* below. It declares names and types for all the observations that can be made of parts (or parameters) of the system, and it places constraints on some or all of those observations in order to ensure that the state space contains only safe states. Of course, this is a design decision; it is equally valid to have a state space that includes unsafe values for some observations, as long as this aspect is handled elsewhere in the spec-

[3]Of course, in order to ensure this is actually true, we must also consider the preservation of these properties in the final implementation, but we will not go into a discussion about refinement here.

ification (e.g. perhaps there are specialised error-handling operations that come into play once we enter a state which is allowed by the model but which is unsafe according to the requirements).

The Z snippets below show some of the definitions used to model the system.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{ReactorControl}} \\ \hline
\mathit{sv}1 : \mathit{Valve} \\
\mathit{sv}2 : \mathit{Valve} \\
\mathit{cpUMin} : \mathbb{N} \\
\mathit{pressure} : \mathbb{N} \\
\mathit{temp} : \mathbb{N} \\
\mathit{wv}1 : \mathit{Valve} \\
\mathit{wv}2 : \mathit{Valve}\ \mathit{outputMW} : \mathbb{N} \\
\mathit{waterlevelMM} : \mathbb{N} \\
\mathit{wp}1\mathit{UMin} : \mathbb{N} \\
\mathit{rodposition} : \mathbb{N} \\ \hline
\mathit{pressure} \leq 70 \\
\mathit{temp} < 286 \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{LowerRods}} \\ \hline
\Delta \mathit{ReactorControl} \\ \hline
\mathit{rodposition} > 0 \\
\mathit{rodposition}' = \mathit{rodposition} - 1 \\
\mathit{wp}1\mathit{UMin} = \mathit{wp}1\mathit{UMin}' \\
\mathit{sv}1' = \mathit{sv}1 \\
\mathit{sv}2' = \mathit{sv}2 \\
\mathit{wv}1' = \mathit{wv}1 \\
\mathit{wv}2' = \mathit{wv}2 \\
\mathit{cpUMin}' = \mathit{cpUMin} \\
\mathit{waterlevelMM}' = \mathit{waterlevelMM} \\
\mathit{outputMW}' = \mathit{outputMW} \\ \hline
\end{array}$$

The *ReactorControl* schema is the state schema showing the observable states of the system (via the parameters or observable values within the system), and *LowerRods* is an operation schema that changes a state—in this case, the position of the rods is the only part of the overall state that changes.

Once we have a complete specification of all allowable behaviours, we can then use a model checker such as the ProZ plug-in for ProB to investigate the specification to ensure that it behaves as expected. For example, if we have specified what it means for the system to be in a stable state, then we should be able to show that when these conditions are not satisfied then the system moves into one of the three error control states—abnormal operation, accident or SCRAM.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Stable}} \\
\hline
\Xi \mathit{ReactorControl} \\
\hline
\mathit{cpUMin} = 1600 \\
\mathit{waterlevelMM} = 2100 \\
\mathit{outputMW} = 700 \\
\hline
\end{array}$$

$$\mathit{Status} \mathrel{\widehat{=}} \mathit{Stable} \lor \mathit{Abnormal} \lor \mathit{Accident} \lor \mathit{SCRAM}$$

In the same way as we describe the system observations for *Stable*, we also describe the *Abnormal, Accident* and *SCRAM* states so that *Status* is then defined as the disjunction of these possible states.

While this high-level view is essential in enabling us to perform the sorts of proofs we require for a safety-critical system, we may wish to consider subsets of behaviour in more detail. We could always, of course, add to the specification to include all of the details needed to consider these, but it is true that the more we reduce the level of abstraction, the more unreadable (and potentially unwieldy) the specification becomes.

Suppose we wish to consider some specific procedures of the power plant which combine several operations. For example, when the plant starts up or shuts down, there are a series of required steps that must occur along with requirements of values of certain parameters (i.e. conditions on observable values) that enable the required steps to proceed. We could investigate the ‘start-up’ and ‘shutdown’ steps using model checking with the Z specification as above; however, in order to more easily consider the user inputs to control these procedures (which are not included in the specification), we might instead create a $\mu$chart which shows the required input levels and reactions that occur in these processes.

Figure 6.3 shows the $\mu$chart of the‘start-up’ procedure for the power plant. It provides a different (and more visual) abstraction of the system than the Z specification might be able to and is more expressive in terms of the reactive properties (the Z is intended to describe what is and is not possible and abstracts away such reactive and event-driven behaviour deliberately), but at the same time (via its Z semantics) retains all of the useful properties of a formal specification. In particular, the Z specification, while it tells us precisely what the state space is and what operations can move us around the state space, says nothing about the *sequencing* of operations that are allowed or possible. This is, in part, the role of the $\mu$chart. For example, the fact that a valve must be open before it can be closed is best handled by the chart. It could formally be handled via setting up flags and preconditions for Z operations in terms of those flags, but this is an example of the unwieldiness of one language over another: picking the right language for each part of a job is part of the skill of a modeller, and part of the elegance that comes from experience. The fact that the modelling *could* all be done in Z, say, is no reason to actually do it all in Z. If there is a better language for certain parts of the model, in this case the use of a chart for the sequencing of operations, then the elegant solution is to do the modelling in that way.

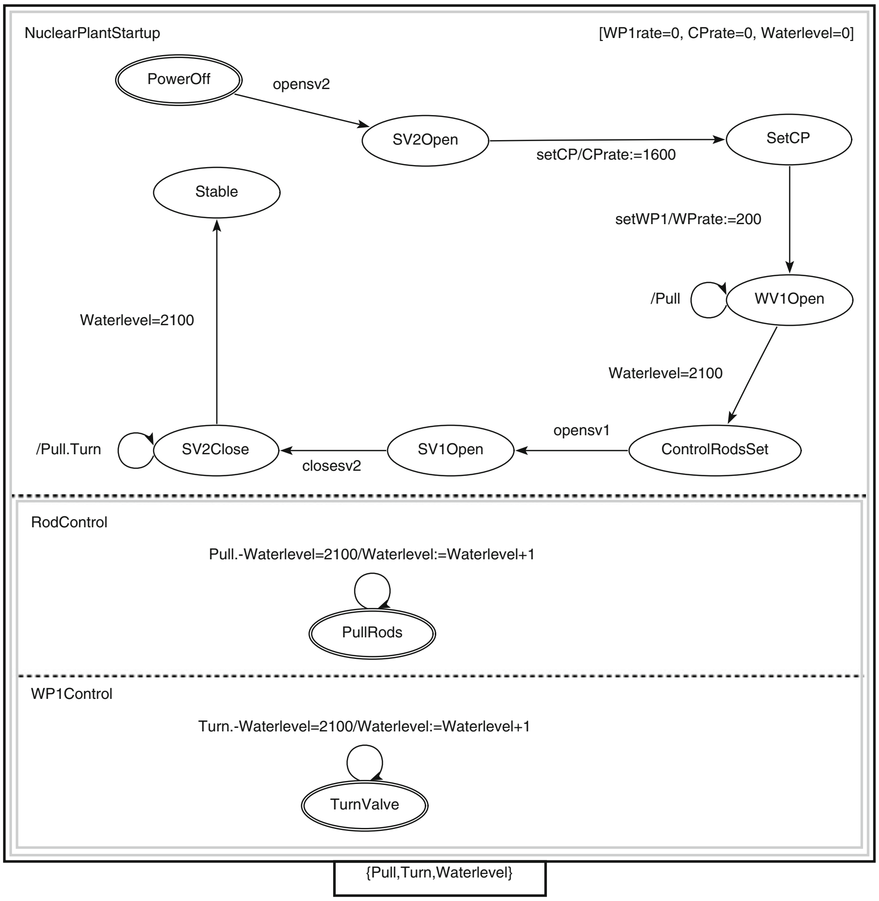

**Fig. 6.3** Chart of start-up procedure

The model consists of three atomic $\mu$charts (which are, in this case, just like simple finite-state machines) composed in parallel, which means that they each react in step with each other and signals listed in the feedback box at the bottom of the chart are shared instantaneously between all charts at each step. This modularity is another of the advantages of using $\mu$Charts as we can explicitly model relationships between independent components and behaviours. For example, in this chart, we can see that once the system is in the *SV2Close* state, it will remain there until the water level reaches the required value, and at each step, it outputs signals (from the self-loop transition) *Pull* and *Turn*.[4] These signals are then shared with the *RodControl* and

[4]Note that these two transitions happen at *every* clock tick since their guards are true, denoted by convention by the absence of any guard.

*WP1Control* charts which lead to transitions which ultimately affect the water level until the required value is reached.

The transitions between the various states the system goes through are guarded by required values on key indicators (such as water level and power output) as well as user operations (such as opening and closing valves). In this model, we still do not distinguish between user operations, system controlled operations and functional monitoring of values. In this way, we still abstract from a user's view of the system as we are most interested here in ensuring that the correct outcomes are reached depending on the values and that the components interact properly as shown by the feedback mechanism of the composed charts.

Using the Z semantics of $\mu$Charts, and a tool called ZooM[5] which generates the Z specification which expresses the meaning of a $\mu$chart, we can go on to model check this component of behaviour to ensure that the system progresses correctly through the start-up procedure (and similarly shutdown) only when the correct preconditions/postconditions are met. Already having these two different, but interrelated, views gives us a consistent mechanism for viewing parts of the system in different ways.

Once we are satisfied that the system will behave correctly as described, we must also ensure that the users can perform the required operations and that at the very least the interface provides the necessary controls (we do not talk about the issue of usability of the interface in this chapter—recall our comment in the introduction about the usual artefacts being available from the UCD process—however, it is of course equally important in ensuring that the system can be used): so we take any design artefacts we have for the user interface to the control system and create presentation models, PIMs and PMR, as described previously.

For the nuclear power plant control system, we start with the initial design shown in Fig. 6.2. The following presentation model and PMR snippet give an example of the models derived from this. There is no PIM at this stage as we are dealing with a single fixed 'window' which has no navigational opportunities for the user and so, as mentioned previously, it is trivially a single-state automata.

**Presentation Model**

```
PowerDisplay, Responder, (S_OutputPower),
RWaterLevelDisplay, Responder, (S_OutputReactorWaterLevel),
RPressureDisplay, Responder, (S_OutputReactorPressure),
ControlRodCtrl, ActionControl, (S_RaiseControlRods,
    S_LowerControlRods),
WP1Ctrl, ActionControl, (S_IncWaterPressure1,
    S_DecWaterPressure1),
WP2Ctrl, ActionControl, (S_IncWaterPressure2, S_DecWaterPressure2),
CPCtrl, ActionControl, (S_IncCPressure, S_DecCPressure),
SV1Open, ActionControl, (S_OpenSV1),
```

[5]http://sourceforge.net/projects/pims1/files/?source=directory.

SV1Close, ActionControl, (S_CloseSV1),
SV1Status, Responder, (S_OutputSV1Status)

**PMR**

S_OutputPower $\mapsto$ OutputPowerLevel
S_OutputReactorWaterLevel $\mapsto$ OutputReactorWaterLevel
S_OutputReactorPressure $\mapsto$ OutputReactorPressure
S_RaiseControlRods $\mapsto$ RaiseRods
S_LowerControlRods $\mapsto$ LowerRods
S_IncWaterPressure1 $\mapsto$ IncreaseWaterPressure
S_DecWaterPressure1 $\mapsto$ DecreaseWaterPressure
S_OpenSV1 $\mapsto$ OpenSV1
S_CloseSV1 $\mapsto$ CloseSV1
S_OutputSV1Status $\mapsto$ OutputSV1Status

For brevity, we do not include all of the status lights and valve controls (e.g. for valves SV2, WV1 and WV2), but the reader can assume they are described in the same manner as the SV1 controls and status display.

The presentation model can be used to ensure that all of the required operations are supported by the user interface, while the PMR ensures that the UI designs are consistent and complete with respect to the functionality of the system. For example, if we have some S-behaviours of the presentation model which do not appear in the PMR, then we know that the interface describes functionality that is not included in the specification and we must therefore address this incompleteness.

We can also use these interface models to help derive alternative (restricted) interfaces for use in error conditions when the user may have only partial control of the system, or when they have no control due to SCRAM mode. Initially, a presentation model of the alternative interfaces provides information about what operations are (and more crucially, are not) available for the user. Subsequently, we can use the refinement theory based on $\mu$Charts trace refinement (Bowen and Reeves 2006b) to examine alternatives and prove that they are satisfactory. The visual appearance of the alternative interfaces may be entirely different from the original (although of course we would want as much correspondence between interfaces for the same system as possible to avoid user confusion). The presentation models of the different interfaces allow us to compare behaviours (via the refinement theory), irrespective of the appearances.

The interface in Fig. 6.2 allows the user full control of all aspects of the system, as it occurs when it is in a stable mode. However, if it moves into one of its error states, then the user has a reduced amount of control (or none in SCRAM mode when the system functions in a totally automated fashion). We might propose changes to the interface to support this restricted control and provide feedback to the user about what is happening and what they can, and cannot, do. Figure 6.4 shows a suggested design change to the nuclear power plant for when the control system is in 'Abnormal' mode, and partial automated-only control is in place.

The presentation model for the interface will then differ from that of the original example as the three pump controls are now displays rather than controls (they show

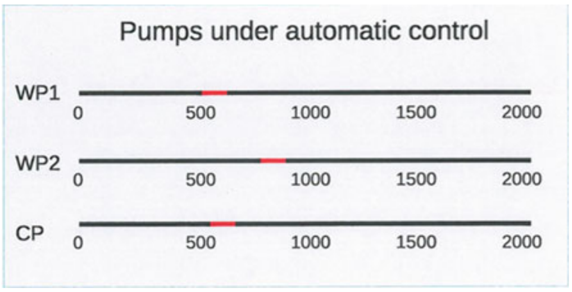

**Fig. 6.4** Restricted interface for pump controls

the user what the automated system is doing rather than enabling the user to make changes themselves). The widgets are still associated with the same behaviours, but instead of generating these behaviours (as action controls do), they now respond to them. In addition, we include the automated behaviour (which drives the mode switch), described as a 'SystemControl', which leads to the interface behaviours of changing the display.

WP1Ctrl, Responder, (S_IncWaterPressure1, S_DecWaterPressure1)
WP2Ctrl, Responder, (S_IncWaterPressure2, S_DecWaterPressure2)
CPCtrl, Responder, (S_IncCPressure, S_DecCPressure)
Status, SystemControl, (S_Stable, I_Stabilised)

Similarly we add the automated behaviour to the original presentation model to make explicit the automation which switches into abnormal mode.

Status, SystemControl, (S_Abnormal, I_AbnormalOperation)

It is important to ensure that this new interface provides exactly the right controls and restrictions to the user, and also that they are only present in the relevant error states (i.e. that a user is not restricted when the system is stable).

For each of the possible error states, Abnormal, Accident and SCRAM, we can provide different interfaces which provide only the correct levels of user interaction. Figure 6.5 shows the PIM for the new collection of interfaces, including those of the other error modes (Accident and SCRAM), although we do not discuss their designs here.

The *Stable* state is considered the initial state (indicated by the double ellipse) and the transitions indicate possible movements between states. The *SCRAM* state is a deadlock state, in that there are no possible transitions out of this state. This is correct for this system as the user cannot interact with the system in this mode and the only possible behaviour for the system is to go into a safe shutdown mode.

There are several types of properties we may wish to consider once we have these designs and their associated models. First, we should ensure that the combination of new interface models still adhere to the original requirements. Second, as stated

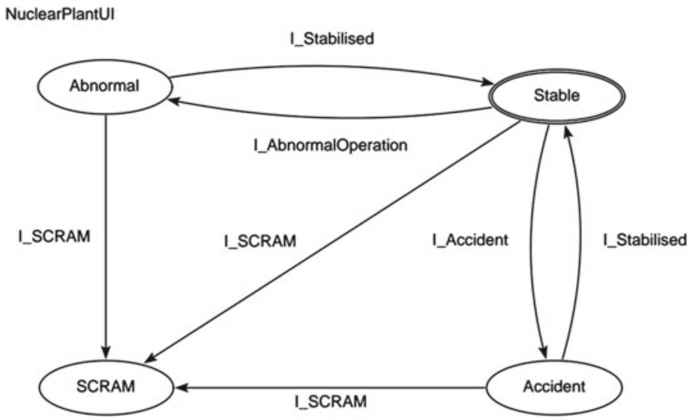

**Fig. 6.5** PIM of interface modes

above, we should be sure that the correct level of control/restriction is provided in each instance.

Typically, when we make changes to the interface or interaction possibilities of our system during the modelling stage, we would use refinement to ensure the adherence to original requirements. However, what we are doing with the new interfaces is to restrict behaviour, so it is not the case that each of the different (new) modes refines the original, but rather that the total interface model (the concatenation of the four *pmodels*) refines the original. We reiterate that we retain all of the visual designs of layout, etc. for both the original as well as the new UIs so that we can always refer back to these to understand more about the actual appearance. This will, of course, be crucial when we come to evaluate the usability aspects of the UIs. In terms of the final considerations of refinement, however, we rely on the models alone.

Refinement for interface and interaction properties described in presentation models and PIMs is based on the notion of contractual utility (Bowen and Reeves 2006b) and can be described by relations on the sets of behaviours of the models in a simple way, while being formally underpinned by the trace refinement theory of $\mu$Charts. Given two arbitrary interfaces, $A$ and $C$, the requirements for the two types of behaviours are as follows:

$$\mathrm{UI}_A \equiv_{SBeh} \mathrm{UI}_C$$
$$\mathrm{I\_Beh}[\mathrm{UI}_A] \subseteq \mathrm{I\_Beh}[\mathrm{UI}_C]$$

where I_Beh[P] is a syntactic function that returns identifiers for all I-behaviours in P.

We call the first interface (from Fig. 6.2) ‘Original’ and the new interfaces (a combination of ‘Original’ with the addition of the automation and ‘Abnormal’) ‘New’. We use the syntactic functions to extract behaviours to create the following sets:

I_Beh[Original] = { }
S_Beh[Original] = {S_OutputPower, S_OutputReactorWaterLevel, S_OutputReactorPressure, S_RaiseControlRods, S_LowerControlRods, S_IncWaterPressure1, S_DecWaterPressure1, S_IncWaterPressure2, S_DecWaterPressure2, S_OpenSV1, S_OpenSV2, S_OutputSV1Status}
I_Beh[New] = {I_AbnormalOperation, I_Stabilised}
S_Beh[New] = {S_OutputPower, S_OutputReactorWaterLevel, S_OutputReactorPressure, S_RaiseControlRods, S_LowerControlRods, S_IncWaterPressure1, S_DecWaterPressure1, S_IncWaterPressure2, S_DecWaterPressure2, S_OpenSV1, S_OpenSV2, S_OutputSV1Status, S_Stable, S_Abnormal}

The requirement on the I-behaviours permits the addition of new behaviours, and so, this is satisfied. However, notice the inclusion of the S-behaviours for the automation to switch the system between states of the interface (S_Stable and S_Abnormal). These are now implicit behaviours of the interface and so must also be included, but these additional behaviours break the requirement on equality between sets of S-behaviours of the interfaces. If we consider this further, we can understand why this is a requirement. Our new interface depends on behaviours to switch between modes under different states of the system which were not part of the original description; as such, there is no guarantee that this behaviour is described anywhere in the specification. We can see from the PMR that there is no relation between these operations and the specification, so we have added functionality to the interface that may not (yet) be supported by system functionality. This is an indication that we need to also increase behaviours in the functional specification which will allow these to then be supported, or we need to ensure that we can relate any existing specified operations (via the PMR) to the new S-behaviours. In fact, we have already discussed earlier how we can describe the different states of the system (Stable, Abnormal, SCRAM, etc.) in the specification, so we must now ensure that the identification of these states along with the necessary behaviour is also described.

Now we can investigate the behaviours of the component *pmodels* (of each different mode of use) via translation of the PIM (to Z) and its corresponding specification to ensure that these correspond to permissible user behaviours in each of the system states. We discuss the combination of models which provides this next.

### *6.4.1 Benefits of Combining the Models*

In the previous sections, we have given examples of the use of the individual models to consider some properties of the nuclear power plant control system which might be of interest during the development process. There are some things, however, which require a combined model of both UI and functionality in order to formally consider. Suppose we want to ensure that when the system is in *Abnormal* mode, the user cannot alter the water pressure (as this is one of the elements under automatic control in this mode). Using the ProB model checker, there are several different ways to perform such analysis, and we have found that describing the properties as LTL formulae and then checking these are a useful mechanism as ProB provides counterexamples consisting of a history list of operations performed when a formula check fails (Bowen and Reeves 2014).

However, the functional specification alone cannot be used for this. If we perform some analysis to show that when the status of the system is *Abnormal*, the operations to increase or decrease the water pressure are not enabled we find that this is not true. Of course, this is exactly as it should be; although the user cannot change the water pressure, it can still be changed (via automation) and our functional model correctly describes this.

In order to consider the possible effects of user interactions, therefore, we need to combine the interface models with the specification. We start by declaring types for the states of the PIM and the transition labels.

$$\begin{array}{l} State ::= Stable \mid Abnormal \mid Accident \mid SCRAM \\ Signal ::= I\_AbnormalOperation \mid I\_Accident \mid I\_SCRAM \mid I\_Stabilised \end{array}$$

Next, we give a description of the UI which consists of a single observation which shows the state the UI is in (one of the states of the PIM) and create an operation schema for each of the transitions which describes the change in state that occurs on a given signal, for example:

$$\begin{array}{|l} \multicolumn{1}{l}{TransitionStableAbnormal} \\ \hline \Delta UI \\ signal? : Signal \\ \hline signal? = I\_AbnormalOperation \\ currentState = Stable \\ currentState' = Abnormal \\ \hline \end{array}$$

This describes how the observation of ‘currentState’ changes when the transition from ‘stable’ to abnormal’ occurs, which requires the ‘I_AbnormalOperation’ input signal to be present.

The final step is to create a combined schema for the system and UI (so we include the schema descriptions for each into a single schema), and then, for each of the operation schemas, we add a precondition which gives the required state of the UI, i.e. shows when the operation is available to a user. Now when we model check the specification and check the condition of whether a user can change the water pressure when the system is in an abnormal state, we find that they cannot, as none of the user operations which change the water pressure are enabled if the system state is abnormal.

In order to create a fuller picture, we should include all of the automation considerations. We can do this by describing automatic control in exactly the same way as we have shown above. That is, we create the alternate set of models including automated interface behaviours (as if they were user controls) and then we can include this with the combined model. This new model then enables us to show that the correct levels of manual (human user) or automatic control occur in all of the different states of the system.

## 6.5 Conclusion

In this chapter, we have described the different models we use for reasoning about interactive systems. Using the nuclear power plant control system as an example, we have shown how the models can be used independently as the differing expressive natures of the languages involved mean that they are individually suitable for different tasks in the process of verifying and validating safety-critical interactive systems. We have also given an example of combining the models into a single formal description, which allows us to ensure correctness of the interactivity and interaction in combination with the functionality.

Although the type of interface used in the nuclear power plant control example consists of standard desktop system controls (buttons, sliders, etc.), this is not a requirement for our methods. Any type of interaction control (speech, touch, gesture, sensor, etc.) can be modelled in the same way as we can abstract them in the presentation models as event generating (action controls) or responding (responders) as shown here.

## References

Bolton ML, Bass EJ (2010) Formally verifying human-automation interaction as part of a system model: limitations and tradeoffs. Innov Syst Softw Eng A NASA J 6(3):219–231

Bowen J, Reeves S (2006a) Formal models for informal GUI designs. In: 1st international workshop on formal methods for interactive systems, Macau SAR China, 31 October 2006. electronic notes in theoretical computer science, Elsevier, vol 183, pp 57–72

Bowen J, Reeves S (2006b) Formal refinement of informal GUI design artefacts. In: Australian software engineering conference (ASWEC'06). IEEE, pp 221–230

Bowen J, Reeves S (2008) Formal models for user interface design artefacts. Innov Syst Softw Eng 4(2):125–141

Bowen J, Reeves S (2013) Modelling safety properties of interactive medical systems. In: 5th ACM SIGCHI symposium on engineering interactive computing systems, EICS'13. ACM, pp 91–100

Bowen J, Reeves S (2014) A simplified Z semantics for presentation interaction models. In: FM 2014: formal methods—19th international symposium, Singapore, pp 148–162

Courtney A (2003) Functionally modeled user interfaces. In: Interactive systems. design, specification, and verification. 10th international workshop DSV-IS 2003. Lecture notes in computer science, LNCS. Springer, pp 107–123

Derrick J, Boiten E (2014) Refinement in Z and object-Z: foundations and advanced applications. Formal approaches to computing and information technology, 2nd edn. Springer

Dix A, Runciman C (1985) Abstract models of interactive systems. Designing the interface, people and computers, pp 13–22

Duke DJ, Harrison MD (1995) Interaction and task requirements. In: Palanque P, Bastide R (eds) Eurographics workshop on design, specification and verification of interactive system (DSV-IS'95). Springer, pp 54–75

Duke DJ, Faconti GP, Harrison MD, Paternò F (1994) Unifying views of interactors. In: Advanced visual interfaces, pp 143–152

Duke DJ, Fields B, Harrison MD (1999) A case study in the specification and analysis of design alternatives for a user interface. Formal Asp Comput 11(2):107–131

Harel D (1987) Statecharts: a visual formalism for complex systems. Sci Comput Program 8(3):231–274

Harrison MD, Dix A (1990) A state model of direct manipulation in interactive systems. In: Formal methods in human-computer interaction. Cambridge University Press, pp 129–151

Henson MC, Deutsch M, Reeves S (2008) Z Logic and its applications. Monographs in theoretical computer science. An EATCS series. Springer, pp 489–596

Hussey A, MacColl I, Carrington D (2000) Assessing usability from formal user-interface designs. Technical report, TR00-15, Software Verification Research Centre, The University of Queensland

ISO, IEC 13568 (2002) Information technology-Z formal specification notation-syntax, type system and semantics. International series in computer science, 1st edn. Prentice-Hall, ISO/IEC

Jacob RJK (1982) Using formal specifications in the design of a human-computer interface. In: 1982 conference on human factors in computing systems. ACM Press, pp 315–321

Limbourg Q, Vanderdonckt J, Michotte B, Bouillon L, López-Jaquero V (2004) UsiXML: a language supporting multi-path development of user interfaces. In: 9th IFIP working conference on engineering for human-computer interaction jointly with 11th international workshop on design, specification, and verification of interactive systems, EHCI-DSVIS'2004, Kluwer Academic Press, pp 200–220

Paternò FM (2001) Task models in interactive software systems. Handbook of software engineering and knowledge engineering

Paternò FM, Sciacchitano MS, Lowgren J (1995) A user interface evaluation mapping physical user actions to task-driven formal specification. In: Design, specification and verification of interactive systems. Springer, pp 155–173

Philipps J, Scholz P (1998) Formal verification and hardware design with statecharts. In: Prospects for hardware foundations, ESPRIT working group 8533. NADA—new hardware design methods, survey chapters, pp 356–389

Puerta A, Eisenstein J (2002) XIML: a universal language for user interfaces. In: Intelligent user interfaces (IUI). ACM Press, San Francisco

Reeve G (2005) A refinement theory for $\mu$charts. PhD thesis, The University of Waikato

Reeve G, Reeves S (2000a) $\mu$-charts and Z: examples and extensions. In: Proceedings of APSEC 2000. IEEE Computer Society, pp 258–265

Reeve G, Reeves S (2000b) $\mu$-charts and Z: hows, whys and wherefores. In: Grieskamp W, Santen T, Stoddart B (eds) Integrated formal methods 2000: proceedings of the 2nd international workshop on integrated formal methods. LNCS, vol 1945. Springer, pp 255–276

Reichart D, Dittmar A, Forbrig P, Wurdel M (2008) Tool support for representing task models, dialog models and user-interface specifications. In: Interactive systems. Design, specification, and verification: 15th international workshop, DSV-IS'08. Springer, Berlin, pp 92–95

Scholz P (1996) An extended version of mini-statecharts. Technical report, TUM-I9628, Technische Univerität München. http://www4.informatik.tu-muenchen.de/reports/TUM-I9628.html

Thimbleby H (1990) Design of interactive systems. In: McDermid JA (ed) The software engineer's reference book. Butterworth-Heineman, Oxford, Chap, p 57

Thimbleby H (2004) User interface design with matrix algebra. ACM Trans Comput Hum Interact 11(2):181–236

Woodcock J, Davies J (1996) Using Z: specification, refinement and proof. Prentice Hall