USING Z TO SUPPORT THE DESIGN OF INTERACTIVE, SAFETY-CRITICAL SYSTEMS

C.W. Johnson
The Department of Computing Science,
The University of Glasgow, Glasgow,
Scotland, G12 8QQ.
E-mail: johnson@dcs.glasgow.ac.uk, Telephone: +44 141 339 8855 ext. 6053
Fax: +44 141 330 4913.

ABSTRACT

Mathematically based, specification techniques are increasingly being recruited to support the development of safety-critical systems. Formal notations, such as Z and VDM, provide precise and concise means of representing a design without forcing commitment to implementation strategies during the early stages of development. Unfortunately, interface requirements are not normally considered within formal specifications. This threatens user-centred design. A prime objective in the use of formal methods is to minimise the modifications that are necessary once a specification has been refined towards implementation. Usability considerations, therefore, run the risk of being relegated to an after-thought in the development process. This paper argues that temporal and presentation issues must be represented within formal specifications of interactive systems.

Keywords: *safety-critical systems; user interface design; formal specification, Z.*

1 Introduction

Formal methods provide a precise and concise means of representing design requirements. They avoid some of the ambiguity that can arise in natural language specifications. They are supporting the development of a range of applications. NASA are applying mathematical specification techniques to assess the design of their Advanced Orbiting System [34]. Formal methods are being widely used in the standardisation of communications protocols for distributed applications [16]. Unfortunately, usability requirements are typically not considered within formal specifications. This has important implications for the development of many systems. The Commission of the European Community [17], the Japanese Fifth Generation Initiative [45] and United States' Presidential Task Forces [37] all cite human intervention as a primary factor in the cause and exacerbation of accidents in the process industries.

1.1 Time

Some notable attempts have been made to apply formal methods to support interface development. Harrison and Thimbleby recruit an algebraic notation to specify high-level requirements for interactive

systems [14]. Took has developed a formal framework for the representation of graphical interfaces [44]. Sufrin and He have exploited the Z schema calculus to represent interaction with a text-editor [42]. An important limitation of this work is that it has not considered how temporal properties can be explicitly represented within specifications of interactive systems. Temporal properties can determine the success or failure of an interactive system [26]. Hysteresis effects can reduce the accuracy and reliability of sensing technology; delays in system responses lead to frustration and error [1, 26]. Safety can also be jeopardised if there is insufficient time for operators to perceive and understand the warnings presented by a system [24]. A number of suggestions have been made for representing time within formal notations [2, 7, 10, 30]. This paper argues that interval temporal logic can be used to represent timing requirements that determine the success or failure of many interactive systems.

1.2 Presentation

Delisle and Garlan argue that presentation details should not be considered during the initial stages of development [8]. Formal specifications must be independent of any particular interface; they should not consider whether a particular function is provided through a menu item, a command line or a hard-wired button. Characterising a system in terms of mathematical or logical relationships strips away the complexity of implementation details, such as device handling, that obscure important properties of an application. In contrast, this paper argues that presentation details must be considered during systems development. The introduction of device and presentation details can, however, threaten the tractability of an abstract specification. It is a non-trivial task to represent and reason about an interface in terms of bitmaps and low-level input events. This paper describes techniques that avoid this problem and yet still provide some impression of the 'look and feel' of a human-machine interface. Structured representations, based around regions, can be used to represent graphical displays at many different levels of abstraction. Input from a range of physical devices, such as mice or tracker balls, can be represented by generic events,

1.3 Outline Of The Paper

This section has argued that usability must be considered during formal development. Section 2 introduces the reactor feedwater example that will be used to illustrate the rest of this paper. Section 3 exploits the Z notation to describe some high level properties of an interface to this system. Section 4 shows how Z can be extended in order to represent temporal properties of interaction. Section 5 argues that this approach can be developed to represent graphical presentation techniques within a formal specification. Section 6 demonstrates that input abstractions can be linked to graphical structures. Section 8 identifies the conclusions that can be drawn from this work and proposes directions for further research.

2 An Application And A Notation

A high level specification for a feedwater control system in nuclear power generation is used as an example for the remainder of this paper. This application exhibits many of the problems that complicate the design of interactive control system. It is complex and dynamic. Feedwater control systems have posed a significant challenge for both systems designers [39] and human factors specialists [25].

Our choice of formal notation is justified by the observation that Z is one of the most widely used specification languages [4]. It is model based. Designers represent the internal data structures

that compose the state of the system. Operations are then defined to represent the ways in which those data structures can be manipulated. Brevity prevents a more complete introduction to the language, further details are provided in a number of textbooks [9, 40]. Informal readings of the formal specifications will be provided throughout the remainder of the paper.

3 Representing Interaction In Z

This section argues that formal notations can be used to strip out the implementation details that obscure important properties of interactive systems. For example, the feedwater control system can be described in terms of a number of parameters:

$$PARAMETER_IDS = \{cntmnt_sump_level, pump_a_press, ...\}$$

The *cntmnt_sump_level* describes the depth of fluid in a containment sump. The *pump_a_press* represents the output pressure produced by pump A. An important benefit of introducing parameters in this manner is that designers can represent a system without being forced to consider low level details during the early stages of development. Decisions about the sampling rate and tolerance of sensors can be postponed until more is known about a potential application. These details can be considered as development progresses [18].

It is possible to identify a number of operations that might be performed on the elements of $PARAMETER_IDS$. Some operations can be identified from system requirements. Sensing devices must be deployed in order to access the *value* of particular parameters. High and low values are used to define a range of permissible values for each parameter; *max* and *min*. Others operations relate both to application requirements and user tasks. For instance, automated systems and operators may request a new target value for a parameter; *set*. The final class of operations relate primarily to user interface requirements. Textual names can be associated with $PARAMETER_IDS$; *name*. This mapping might be used to ensure that a consistent labelling scheme is used whenever information is presented about a particular process parameter. Names are composed of $CHARACTERS$; literals such as $a, .., z, 0, .., 9, A, .., Z$ and punctuation marks. $TEXT$ is the set of character sequences:

$$TEXT = seq\ CHARACTERS$$

We adopt the convention of enclosing $TEXT$ sequences within quotation marks.

The following Z schema describes the feedwater system:

Feed_System_Application

$$set, value, max, min : PARAMETER_IDS \rightarrow \mathbb{Z}$$

$$name : PARAMETER_IDS \rightarrow TEXT$$

$$\forall p : PARAMETER_IDS \bullet max(p) \geq value(p) \geq min(p)$$

Even at this level of abstraction it is possible to characterise important properties of an interactive system. For instance, the operator could respond to a steam rupture accident by entering a target value, *new_setting*?, for the temperature, *param*?, of the primary coolant system. The following schema shows that designers can use Z to describe the effect of operator intervention without specifying the devices that provide input such as *param*? and *new_setting*?. Later sections will show how more detailed information about input events can be introduced as development progresses.

Set_Parameter

$\Delta Feed_System_Application$

$param? : PARAMETER_IDS$

$new_setting? : \mathbb{Z}$

$set' = set \oplus \{param? \mapsto new_setting?\}$

$value' = value$

$max' = max$

$min' = min$

$name' = name$

Assuming that an operator provides valid input to alter the setting of a process parameter, the system must then ensure that the parameter assumes the new value. In other words, the readings for a variable must reach the level specified by the new setting:

Change_Parameters

$\Delta Feed_System_Application$

$param? : PARAMETER_IDS$

$value'(param?) = set(param?)$

$set' = set$

$max' = max$

$min' = min$

$name' = name$

Having represented high-level input requirements within a formal specification, it is important to show that designers can represent displays. This can be done by introducing a set of formats. For instance, *sump_levels* could include information about the contamination of liquid accumulating within the drainage system. Similarly, the *steam_circuit* format presents information about the pressure at various points within the system:

$$DISPLAY_FORMATS = \{sump_levels, steam_circuit, ...\}$$

Designers must be able to represent and reason about the information content of displays. In terms of a Z specification, the information content of a particular display can be characterised by those parameters that it presents. A relation, *display_params*, can be introduced between system parameters and the display formats that contain information about those parameters. Similarly, a *display_text* relation can be introduced to represent those formats that include items of text. Finally, the *image* presented to an operator can be described in terms of a set of display formats. This set is intended to capture the information displayed by various presentation devices, without considering unstructured bitmaps and 'raw' devices during the early stages of the development process.

Feed_System_Display

$$display_params : PARAMETER_IDS \leftrightarrow DISPLAY_FORMATS$$

$$display_text : TEXT \leftrightarrow DISPLAY_FORMATS$$

$$image : \mathbb{P}\, DISPLAY_FORMATS$$

These abstractions can be used to represent potential presentation techniques. For instance, the operators of many safety-critical applications cannot reconfigure the information content of their displays. It is not possible for them to move items to new locations on a screen nor is it possible to move items onto new screens. This is intended to ensure that information is presented in a consistent manner [15]. It is possible to navigate between different formats by requesting new screens of information. A request for information about a particular parameter, *param*?, results in the presentation of a *format* that includes information about that parameter:

Update_Display_Format

$$\Delta Feed_System_Display$$

$$param? : PARAMETER_IDS$$

$$\exists format : image' \bullet (param?, format) \in display_params'$$

$$display_params' = display_params$$

$$display_text' = display_text$$

An important benefit of the Z schema calculus is that operations on the state of an application can be separated from operations which only affect the display. Designers can use this to focus upon particular issues during the development of an interactive control system. For example, *Set_Parameter* represented the setting of a new target value without describing the ways that operators view the effects of their commands. *Update_Display_Format* represented the presentation of a parameter without describing situations in which its users are informed of changes to an application. The Z schema calculus provides a means of representing the interaction between an application and its display. For instance, designers could require that users view information about a parameter that has a new target setting. The *Set_Parameter* and *Update_Display_Format* operations can be combined to express this requirement. The input of a target *setting*? for a parameter, *param*?, can be tied to the presentation of its value in a display *format*:

$$Set_And_Display \mathrel{\widehat=} Set_Parameter \wedge Update_Display_Format$$

Schemas can also capture requirements that relate to both an application and its display. For instance, users must be warned if they attempt to set a value which is either too low or too high for a particular parameter. The input must be validated against the current range of admissible values for the application. The warning must be included within the current image of the system:

High_Set

$\Xi Feed_System_Application$

$\Delta Feed_System_Display$

$param? : PARAMETER_IDS$

$new_setting? : \mathbb{Z}$

$new_setting? > max(param?)$

$\exists format : image' \bullet$
$\quad (\text{“}Setting\ of\ \text{”} \frown name(param?) \frown \text{“}\ high\text{”}, format) \in display_text'$

$display_params' = display_params$

$display_text' = display_text$

Low_Set

$\Xi Feed_System_Application$

$\Delta Feed_System_Display$

$param? : PARAMETER_IDS$

$new_setting? : \mathbb{Z}$

$new_setting? < min(param?)$

$\exists format : image' \bullet$
$\quad (\text{“}Setting\ of\ \text{”} \frown name(param?) \frown \text{“}\ low\text{”}, format) \in display_text'$

$display_params' = display_params$

$display_text' = display_text$

Ok_Set_Parameter

$Set_Parameter$

$max(param?) \geq new_setting? \geq min(param?)$

The previous schemas illustrate how designers can use Z to identify the different outcomes that can arise when operators attempt to alter a parameter setting. By examining each different case, it is possible to devise appropriate warnings. In the examples shown above, these were items of $TEXT$. Later sections will show how graphical presentation details can be introduced into formal specifications of interactive systems. For now it is sufficient to observe that the three alternative cases can be used to build a single operation that specifies the possible dialogues that can result from an attempt to alter a parameter setting:

$Validate_Set_Parameter \mathrel{\widehat=}$
$(Ok_Set_Parameter \land Update_Display_Format) \lor Low_Set \lor High_Set$

Having described some of the benefits that Z offers for the development of interactive systems, it is important to identify its limitations. In particular, it provides limited support for the specification of temporal properties. Designers might use composition within the schema calculus to describe sequences of operations. The input of a valid setting can be composed with a change in the value of a parameter. The composition $\fatsemi$ operator requires that the setting is validated and the parameter information displayed before the new target is achieved by the system:

$$Achieve_Valid_Setting \mathrel{\widehat=} (Ok_Set_Parameter \land Update_Display_Format) \fatsemi Change_Parameters$$

Unfortunately, the $\fatsemi$ operator cannot capture all of the temporal properties that affect the usability of an interactive control system. Potential delays can arise between the issuing of input and consequent changes in the value of a parameter. In consequence, designers might require that the *value* of a process attribute should be presented until the new setting is reached. Similarly, it is frequently important to synchronise the activities of operators so that they do not interfere in the tasks of their colleagues. In order to support this designers must carefully consider the sequencing and duration of the display formats that are presented to multiple operators. Operational composition does not capture these temporal relationships.

4 Temporal Logic

A number of attempts have been made to represent temporal properties within Z specifications [2, 7, 10, 11, 30]. The following section applies one of these approaches to support the design of interactive control systems.

Interval temporal logic extends predicate calculus to include the following operators: $\Diamond$ (read as 'eventually'); $\bigcirc$ (read as 'next'); $\Box$ (read as 'always') and $\mathcal{U}$ (read as 'until'). Previous work by the author has exploited executable versions of this notation to derive prototypes from high level designs of interactive control systems [19, 23]. Temporal logics also provide well developed proof techniques through Kripke semantics [32]. Duke and Smith show how this notation can be introduced into Z specifications [10]. The semantics of their notation can be confusing; temporal operators are applied to both schemas and predicates. This complicates the task of interpreting the resulting formulae. In contrast, the following section provides an alternative operational semantics for the introduction of temporal operators within the Z notation. Readers who are more interested in the application rather than the formal basis of the notation are encouraged to move to the *Low_Set_Response* schema.

Before introducing temporal operators into Z, it is important to understand their syntax and semantics outside of the schema calculus. The following paragraphs present a temporal logic proposed by Manna and Pnueli [31]. The syntax of the interval temporal logic language uses elements of a set of symbols to represent variables, constants, propositions, functions and predicates. This set can be partitioned into global and local symbols. Global symbols have a uniform interpretation; they do not change their value or meaning from state to state. In contrast, local symbols may assume different meanings and values from state to state. The set of symbols can also be partitioned into different sorts. These correspond to different domains. For instance, the following symbols are associated with the domain of natural numbers:

$$0, 1, 2, 3, 4, 5, ..., +, -, \mathrm{div}, \times, ..., \geq, \leq$$

For each sort there may be constants that are interpreted over the associated domain. There can also be variables which assume values from that domain, function symbols that represent functions over

the domain and predicate symbols which represent predicates over that domain. The logic contains the set of boolean connectives:

$$\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow, \Leftarrow$$

The logic also contains the first-order quantifiers $\exists$ and $\forall$ together with the temporal operators:

$$\bigcirc, \Box, \Diamond, \mathcal{U}$$

The first three of these are unary operators, whilst $\mathcal{U}$ is a binary operator. Terms are built from the application of functions and predicates to constants and variables. Applications must satisfy the sort and arity restrictions associated with each of the symbols. It should also be noted that if t is a term then so is $\bigcirc t$. Formulas are constructed from the application of boolean connectives, temporal operators or quantifiers to terms.

The semantics of the logic can be described in terms of a model (I, α, δ). This consists of a global interpretation, I, a global assignment, α, and a sequence of states, δ. I specifies a domain corresponding to each sort and assigns concrete elements of that domain to symbols. α assigns a value over an appropriate domain to global free variables and propositions. $\delta = s_0, s_1, ...$ is an infinite sequence of states where each s_i assigns values to local free variables and propositions. The following provides an inductive definition of the truth value of temporal formula w in a model (I, α, δ). I is implicitly assumed, the value of a subformula or term, τ, is denoted by $\tau \mid^{\alpha}_{\delta}$

- For $\bigcirc$ application: $\bigcirc w \mid^{\alpha}_{\delta} = w \mid^{\alpha}_{\delta^1}$. Here the value of $\bigcirc w$ in $\delta = s_0, s_1, ...$ is given by the value of w in the shifted sequence $\delta^1 = s_1, s_2, ...$
- For $\Box$ application: $\Box w \mid^{\alpha}_{\delta} = true$ iff for every $k \geq 0, w \mid^{\alpha}_{\delta^k} = true$. w is true for all suffix sequences of δ.
- For $\Diamond$ application: $\Diamond w \mid^{\alpha}_{\delta} = true$ iff there exists $k \geq 0, w \mid^{\alpha}_{\delta^k} = true$. w is true in at least one suffix of δ.
- For $\mathcal{U}$ application: $w_1 \mathcal{U} w_2 \mid^{\alpha}_{\delta} = true$ iff for some $k \geq 0, w_2 \mid^{\alpha}_{\delta^k} = true$ and for all $i, 0 \leq i < k, w_1 \mid^{\alpha}_{\delta^i} = true$.

It should be noted that the $\bigcirc$ does not specify a real-time interval. This problem can be avoided by linking all state transitions to a clock, $\forall\, i(s_i, s_{i+1}) \in \delta$. Ladkin has developed similar techniques for grounding Allen's convex interval logic in real-time [27]. In contrast, this section focuses on the integration of Z and temporal logic operators. The semantics of the logic are defined in terms of sequences of states. For our example system, the state is derived from the application and the display:

$$State \mathrel{\widehat=} Feed_System_Display \wedge Feed_System_Application$$

Sequences of states are not normally represented in Z beyond the before and after conditions which are described within operation schemas. In order to introduce temporal logic into the specification language, a relation must be defined between the before and after states of an operation:

$$OP \subseteq State \leftrightarrow State$$

The *State* schema can now be extended to include a successor relation between states of the system:

Temporal_State

$State$

$successor : OP$

$State \in \mathrm{dom}\, successor$

The successor function constructs the sequence of states which is represented by $\delta = s_0, s_1, ...$ within the semantics of the temporal logic. We introduce a set of well formed formulae in our temporal logic, determined by the syntax described above:

$[TEMPORAL_WFF]$

An evaluation function, *eval*, returns the value of well formed formulae within a state indexed by its position in a sequence or history of interaction. Temporal operators can now be introduced as follows:

History

$hist : \mathrm{seq}\ Temporal_State$

$eval : TEMPORAL_WFF \times Temporal_State \rightarrow BOOLEAN$

$\forall i : \mathbb{Z} \mid 0 \leq i < \#hist \bullet successor\ hist(i) = hist(i+1)$

$\forall w \in TEMPORAL_WFF; \forall i : \mathbb{Z} \mid i \geq 0 \bullet eval(\neg\, w, hist(i)) \Leftrightarrow eval(w, hist(i)) = false$

$\forall w \in TEMPORAL_WFF; \forall i : \mathbb{Z} \mid i \geq 0 \bullet eval(\bigcirc w, hist(i)) \Leftrightarrow eval(w, hist(i+1))$

$\forall w, w_1 \in TEMPORAL_WFF; \forall i : \mathbb{Z} \mid i \geq 0 \bullet eval(w \mathcal{U} w_1, hist(i)) \Leftrightarrow$
$\quad \exists j : \mathbb{Z} \mid i \leq j \leq \#hist \bullet eval(w_1, hist(j)) = true \wedge$
$\quad \forall l : \mathbb{Z} \mid i \leq l < j \bullet eval(w, hist(l)) = true$

$\forall w \in TEMPORAL_WFF; \forall i : \mathbb{Z} \mid i \geq 0 \bullet eval(\Diamond w, hist(i)) \Leftrightarrow eval(true \mathcal{U} w, hist(i))$

$\forall w \in TEMPORAL_WFF; \forall i : \mathbb{Z} \mid i \geq 0 \bullet eval(\Box w, hist(i)) \Leftrightarrow eval(\neg \Diamond \neg\, w, hist(i))$

Temporal logic can be recruited to represent high level properties of interactive control systems. In our example, users might be required to provide input in the next interval after a warning has been displayed for a parameter underflow. During later stages of development, designers could restrict this immediate response to those *PARAMETER_IDS* that require the most urgent intervention:

Low_Set_Response

$History$

$param? : PARAMETER_IDS$

$input_text? : TEXT$

$text_to_num : TEXT \rightarrow \mathbb{Z}$

$\exists format : image'; setting : TEXT \bullet$
$\quad (\text{“}Setting\ of\ \text{”} \frown name(param?) \frown \text{“}\ low\text{”}, format) \in display_text \Rightarrow$
$\quad\quad \bigcirc(input_text? = name(param?) \frown setting \wedge$
$\quad\quad\quad text_to_num(setting) > min(param))$

The previous schema is extremely deterministic in the sense that the user is required to provide input immediately after a warning has been issued. Given divided attention and a noisy working environment it may only be possible to ensure that the operator eventually responds to the warning. The $\Diamond$ operator can be used to represent such liveness properties:

Live_Low_Set_Response

$$
\begin{array}{l}
History \\
param? : PARAMETER_IDS \\
input_text? : TEXT \\
text_to_num : TEXT \rightarrow \mathbb{Z} \\
\hline
\exists\, format : image';\ setting : TEXT \bullet \\
\quad (\text{“}Setting\ of\ \text{”} \frown name(param?) \frown \text{“}\ low\text{”}, format) \in display_text \Rightarrow \\
\quad\quad \Diamond(input_text? = name(param?) \frown setting \wedge \\
\quad\quad\quad text_to_num(setting) > min(param))
\end{array}
$$

The use of an interval notation provides designers with the means of representing temporal properties without being forced to specify real-time requirements. This is important because in most situations it is not possible to specify the exact moment when an operator will respond to a warning. Weakening the $\bigcirc$ requirement with the $\Diamond$ operator does, however, raise further development issues. For example, a user may not respond appropriately if a warning is removed shortly after the underflow occurs. The message must be displayed until the operator establishes a setting higher than the parameter's minimum acceptable value. This requirement can be specified using the $\mathcal{U}$ operator:

Warning_Until_Low_Set_Response

$$
\begin{array}{l}
History \\
param? : PARAMETER_IDS \\
input_text? : TEXT \\
text_to_num : TEXT \rightarrow \mathbb{Z} \\
\hline
\exists\, format : image';\ setting : TEXT \bullet \\
\quad (\text{“}Setting\ of\ \text{”} \frown name(param?) \frown \text{“}\ low\text{”}, format) \in display_text\ \ \mathcal{U} \\
\quad\quad (input_text? = name(param?) \frown setting \wedge \\
\quad\quad\quad text_to_num(setting) > min(param))
\end{array}
$$

The logic notation can also be used to describe the temporal relationships that must hold between an application and the display presented to system operators. Section 3 used the *Change_Parameters* schema to describe the effect of operator input upon the setting of a process parameter. Given transmission delays and the physical properties of many applications it may not be possible to guarantee that a change in setting will have an immediate effect upon the monitored value of a process variable. As in *Live_Low_Set_Response*, the $\Diamond$ operator can be used to represent such delays:

Delay_Change_Parameters

$History$

$param? : PARAMETER_IDS$

$max(param?) \geq set(param?) \geq min(param?) \Rightarrow$
$\quad \Diamond(value(param?) = set(param?))$

Interface designers must carefully consider the consequences of such delays upon system operators. De Keyser argues that users develop a range of coping strategies which help them to combat the uncertainty that can be created by a lag between input and its effect [24]. Users make predictions about the evolving behaviour of a system over time. They might, therefore, attempt to estimate the moment when the value reaches the setting of a process variable. These predictions are supported if operators are provided with sufficient feedback to inform them about the impact of their intervention. In other words, the value of a parameter must be displayed until it has reached its setting. This would provide the contextual information which is necessary in order for operators to determine whether their commands are having the desired effect:

Display_Delay_Change_Parameters

$Delay_Change_Parameters$

$Feed_System_Display$

$\exists format : image \bullet$
$\quad (param?, format) \in display_params \ \mathcal{U} \ \ (value'(param?) = set(param?))$

The introduction of temporal logic operators into the Z notation provides a means of representing important properties of an interactive system. Temporal logic does not, however, provide a panacea for the development of safety critical, interactive systems. In particular, Norman argues that "inappropriate feedback" is the primary problem facing the development of safety-critical systems [36]. It is difficult to assess the value of the feedback described in previous schemas because they provide little detail about the way in which information is presented.

5 Presentation

The complexity of modern display techniques can be illustrated by the following description of the Main Control Room within the U.K.'s Sizewell B reactor:

> "Displays which are considered to be of major significance, or which are frequently used, can be accessed using single dedicated keys from the keyboard. In addition to direct access and access from targets, it is possible to obtain pop-up menus by operating the <menu> key with the cursor on a particular portion of the screen. The operator can then cycle numerically through all of the displays in a given plant area, or access related displays which are not process mimics, such as alarm lists, related trends or histograms, and tabular information." [15]

Designers must be able to represent and reason about such techniques during the development of safety critical interfaces. Previous sections have represented the information presented to system operators

in terms of the sequences of characters that form $TEXT$s. This approach provides an extremely poor impression of the 'look and feel' of an interactive system. It cannot easily be used to describe the graphical presentation techniques that are being recruited to support the operation of production processes.

Formal methods can support the design of graphical interfaces for safety-critical systems [3, 44]. It is important, however, that the introduction of presentation details should not threaten the tractability of an abstract specification. For instance, bitmaps would quickly increase the complexity of a graphical specification. It would be difficult to break an image down into its component parts. In consequence, designers would be forced into a one step refinement between graphical abstractions, such as display formats, and bitmap data structures. Alternatively, designers might introduce graphical requirements by examining higher level properties of the images that are presented to system operators. An image might be described in terms of its X and Y dimensions. Natural numbers of pixels could be used to represent this information. Such detail would again introduce a high degree of device dependence. Requirements can only be satisfied by hardware that supports the pixel coordinates used within a specification. In contrast, we define the dimensions of an image to be positive real values which are calculated relative to the dimensions of the screen. For instance, if the screen is 1.0×1.0 units in size then an image which occupies one quarter of the screen would have dimensions 0.5×0.5. During the later stages of development, these coordinates must be translated onto the pixel space supported by a particular presentation device. It is not necessary to specify an upper bound upon display dimensions because it is perfectly feasible for an image to be larger than the screen. Only part of the display would be visible and the operator must be provided with facilities for navigating around the rest of the image. In order to represent this coordinate scheme, we introduce a set of real numbers greater than or equal to zero; $\mathcal{R}_0$. It would make little sense to have an image of negative dimensions. We assume here that $\geq$ is defined between 0.0 and elements of the reals, $\mathcal{R}$:

$$\mathcal{R} = \{r : \mathcal{R} \mid r \geq 0.0\}$$

$$COORDINATES = \mathcal{R}_0 \times \mathcal{R}_0$$

Having defined the coordinate space, it is necessary to consider ways in which the content of an image can be represented within Z specifications. Complex images can be described in terms of their component parts. This supports the gradual introduction of graphical details as development progresses. The informational requirements identified for system operators frequently change during development; parts of a display can be altered to reflect those changes. This approach also provides a convenient means of developing multi-user interfaces. Designers can construct displays around those parts of an image which are presented to a number of operators. There are further human factors benefits. Rasmussen and Lind argue that structured presentation formats help operators to learn the layout of their plant [38]. Figure 1 illustrates how the image of a *steam_circuit* display format can include a background region, condensers, feedwater pumps, turbines, etc. The condenser image can be further decomposed into sub-parts containing lines, squares or circles. In our specification, all parts of an image will have a unique identifier, elements of $REGION$:

$$[REGION]$$

A display format can be described in terms of its region hierarchy. At the top there is a root. The root can have children regions that may be the parents of further regions:

Display

$root \subset REGION$

$root_of_format_structure : DISPLAY_FORMATS \rightarrow root$

$child : REGION \leftrightarrow REGION$

$parent : REGION \nrightarrow REGION$

$\forall r : root \bullet r \notin \text{ran } children \wedge r \notin \text{dom } parent$

Each region has properties, such as size and position, attributes, such as font and colour, and a behaviour, such as whether or not it is selectable. Fonts could be decomposed into the individual lines that form the image of each letter. This would be a non-trivial task and it is difficult to identify any benefits from this additional level of graphical decomposition. In contrast, the following sets are introduced:

$FONTS = \{times_roman_7pt, courier_11pt, ...\}$
$COLOURS = \{cyan, red, blue, black, ...\}$
$LINES = \{start, finish : COORDINATES\}$

Colours are represented by a set of identifiers that must be mapped to the display attributes of particular presentation devices. Saturation, luminosity and hue may be determined during the later stages of development. Lines are represented by the coordinates that describe their end points. Curves can be represented in terms of line segments, sequences of *COORDINATES* or characteristic functions. The following schema represents the colour and other attributes of graphical regions:

Display_Attributes

Display

$Line_coordinates : \mathbb{P}\, LINES$

$lines : REGION \nrightarrow Line_coordinates$

$font : REGION \rightarrow FONTS$

$colour : REGION \rightarrow COLOURS$

$dimension : REGION \rightarrow COORDINATES$

$position : REGION \rightarrow COORDINATES$

$text : REGION \nrightarrow TEXT$

$selectable : REGION \rightarrow BOOLEAN$

$movable : REGION \rightarrow BOOLEAN$

...

$\forall r : REGION \bullet movable(r) \Rightarrow selectable(r)$

The previous schema provides a framework which can be used to describe part of the image shown in Figure 1 as follows:

Steam_Circuit_Display

$Display_Attributes$

$steam_circuit : DISPLAY_FORMATS$

$steam_circuit_region, condenser_a,$
$pump_a_icon, pump_a_pressure_region : REGION$

$steam_circuit \in image'$

$root_of_format_structure(steam_circuit) = steam_circuit_region$

$child(steam_circuit_region) = condenser_a$

$position(condenser_a) = (0.8, 0.4)$

$dimension(condenser_a) = (0.05, 0.05)$

$selectable(condenser_a) = true$

$colour(condenser_a) = green$

$child(steam_circuit_region) = pump_a_pressure_region$

$child(steam_circuit_region) = pump_a_icon$

...

This schema describes part of the image of the steam circuit display format, it does not describe its behaviour. The static, declarative, representation must be linked back to the specification of the underlying application. Without such a link, displays might be developed that include information which cannot be derived from process sensors. Similarly, it would be possible to propose presentation formats that omit necessary process data. These problems could be avoided by considering the process parameters which are to be represented by a particular image:

Present_Steam_Circuit_Display

$History$

$\Delta Feed_System_Display$

$Steam_Circuit_Display$

$num_to_text : \mathbb{Z} \rightarrow TEXT$

$pump_a_press, pump_b_press : PARAMETER_IDS$

$(pump_a_press, steam_circuit) \in display_params$

$num_to_text(value(pump_a_press)) \in text(pump_a_pressure_region)$

$(pump_b_press, steam_circuit) \in display_params$

...

It is important not to under-emphasise the role of such schemas within the formal design process. They capture requirements that relate to both human factors and systems engineering. For instance, the details of functions such as *num_to_text* must be determined through collaboration between software

developers and interface designers. Task analysis provides means of assessing the precision which operators require during a range of different plant conditions. Systems engineering provides the source data from which this information is derived. Problems can arise if task analysis identifies an informational requirement which is not supported by application functionality. Conversely, it may be difficult to develop appropriate displays if human factors engineers are not informed of design changes which effect the parameters that characterise the behaviour of production processes.

The integration of temporal logic and Z can be used to represent changes in the appearance of an image. For instance, if the reading derived for the output pressure of pump A exceeds its permitted maximum then the region displaying that reading is red until it returns to a normal value. In the following schema the requirement that the colour of the pressure region is not red ensures that its appearance changes once the error is detected. The $\mathcal{U}$ operator is intended to synchronise the presentation of the warning with the duration of the error:

Present_Steam_Circuit_Error_Display

$$
\begin{array}{l}
Present_Steam_Circuit_Display \\
\hline
(colour(pump_a_pressure_region) \neq red \wedge \\
value(pump_a_press) > max(pump_a_press)) \Rightarrow \\
\bigcirc (colour(pump_a_pressure_region) = red \ \mathcal{U} \\
max(pump_a_press) \geq value(pump_a_press) \geq min(pump_a_press))
\end{array}
$$

The change in the image of the pump pressure region might be intended to elicit operator intervention. Users should respond to the error by providing input to shut the pump down. This could also act as confirmation that the warning has been observed. The following schema uses an *input_text* to represent operator intervention:

Present_Steam_Circuit_Error_Shutdown

$$
\begin{array}{l}
Present_Steam_Circuit_Display \\
\\
input_text? : TEXT \\
\hline
(colour(pump_a_pressure_region) \neq red \wedge \\
value(pump_a_press) > max(pump_a_press)) \Rightarrow \\
\bigcirc (colour(pump_a_pressure_region) = red \ \mathcal{U} \\
(max(pump_a_press) \geq value(pump_a_press) \geq min(pump_a_press)) \vee \\
input_text? = \text{“}shut\ pump\ A\text{”})
\end{array}
$$

6 Input

It is not sufficient to represent input in terms of *TEXT*. Graphical displays frequently exploit trackerballs, mice and touch-sensitive devices in addition to keyboards and hard-wired buttons. The choice of input media profoundly affects the usability of many safety-critical systems. For instance, Galer and Yap have investigated the costs and benefits of different input devices for the users of intensive care systems [12]. Some operators suffered from high error rates when using thumb wheels, mice were difficult to use in cluttered clinical environments. The same observations have been made about process control applications [35]. Previous schemas have not shown how input from a range of devices can be represented within a specification.

Designers must consider the trade-offs that exist between tracker-balls, mice, joysticks and keyboards. This could be done by assessing input devices at the same level of detail as other sources of information within safety-critical applications [5]. For example, requirements could be expressed in terms of the data structures that are maintained by device drivers. Burton et al. show how this approach can be used to specify user interfaces built upon the Apple Macintosh Toolbox [6]. Such descriptions are appropriate for many stages in the development of safety-critical systems. They are, however, extremely device dependent. The complexity of representing input at this level of detail is likely to dissuade designers from assessing the costs and benefits of a range of devices.

Abstraction provides a means of avoiding device dependence during the early stages of development. Devices can be represented by the events that they generate. For instance, a mouse can be represented by an *on_select* event. This would be generated every time an operator pressed one of its buttons. The *on_select_up* event would be generated when the user released that button. The *on_CR* event would be generated every time the user entered a carriage return from the keyboard. These input events can be represented within a Z specification as follows:

$$EVENT = \{on_CR, on_select, on_select_up, on_move, ...\}$$

Following Took we associate events with graphical regions [44]. An *on_select* event can be associated with the region which is currently underneath the mouse pointer. An *on_CR* event can be associated with the region which is currently being edited. *Events_Display_Attributes* extends *Display_Attributes* to include an *input_event* function that returns the most recent event generated for a region:

Events_Display_Attributes

$$Display_Attributes$$

$$input_event : REGION \pfun EVENT$$

This schema can be used to introduce generic input requirements into a formal specification. For example, a select event can be associated with the *pump_a_icon* in the *steam_circuit* display format. Users might issue this input in order to shut down a malfunctioning pump. The effect of the operator intervention can be represented by the setting of the pump pressure to its minimum value:

Select_Steam_Circuit_High_Shut_Down

$$Present_Steam_Circuit_Display$$

$$Events_Display_Attributes$$

$$(colour(pump_a_pressure_region) \neq red \land$$
$$value(pump_a_press) > max(pump_a_press)) \Rightarrow$$
$$\bigcirc (colour(pump_a_pressure_region) = red \ \mathcal{U}$$
$$(max(pump_a_press) \geq value(pump_a_press) \geq min(pump_a_press)) \lor$$
$$(input_event(pump_a_icon) = on_select \land$$
$$set(pump_a_press) = min(pump_a_press)))$$

There is considerable potential for the use of formal methods in interface design. For instance, the increasing application of Computer Supported Cooperative Work (CSCW) systems is forcing designers to represent and reason about complex multi-user dialogues. The development of COLAB [41] and RTCAL [13] has emphasised the problems of synchronising multiple access to shared resources. In

our example, contention might arise if one user attempts to reduce the output pressure of a pump whilst another operator attempts to increase it. This leads to deadlock if neither can progress until the other abandons their attempt. In order to represent and reason about such problems, designers can introduce a set of users:

$$[USERS]$$

Subsets can be introduced to distinguish between operators who belong to particular work groups. This provides a means of describing constraints on those users who are permitted to intervene in safety-critical processes. Designers might do this by restricting certain input events to the users within a particular group. The following schema shows how input events can be associated with system operators:

Multi_User_Events_Display_Attributes

$$Display_Attributes$$

$$multi_user_input_event : USERS \times REGION \pfun EVENT$$

This schema and the temporal operators, introduced in previous sections, can be exploited to represent synchronisation requirements for multi-user systems. Voting mechanisms resolve contention by requiring mutual agreement before commands take effect. This democratic approach is the antithesis of 'primary user' systems in which input privileges are associated with a single participant who can dictate the course of interaction. The following schema specifies that when *user_1* issues a select event on the *pump_a_icon* the pressure of the pump does not reach its minimum value until *user_2* issues the same select event:

Select_Steam_Circuit_High_Shut_Down

$$Present_Steam_Circuit_Display$$

$$Multi_User_Events_Display_Attributes$$

$$\forall\, user_1, user_2 : USERS \bullet$$
$$multi_user_input_event(user_1, pump_a_icon) = on_select \wedge$$
$$\neg\, (value(pump_a_press) = min(pump_a_press))\ \mathcal{U}$$
$$multi_user_input_event(user_2, pump_a_icon) = on_select$$

This requirement could be satisfied by implementations that provided tracker-balls, cursor keys, mice. All of these devices can generate *on_select* events. Such device independence helps to avoid premature commitment to particular hardware platforms [43]. Implementation decisions can be postponed until late in the development cycle when the costs and benefits of a range of different input media have been assessed. This encourages designers to identify those devices that are most appropriate to the particular tasks and environments of system users.

7 Prototyping

We are concerned to validate the properties that are described in Z schemas. Design requirements expressed in mathematical notations provide the non-formalist with little idea of what it would be like to interact with a potential implementation. Prototypes provide a far better impression of the look and

feel of an interactive system. They can be shown to members of concurrent design teams. They can be shown to operators and are amenable to experimental analysis. Logic programming environments, such as that supported by PROLOG, provide a convenient bridge between formal specifications and prototype implementations. We have developed the Prelog prototyping system [21, 22, 23]. This integrates a temporal logic interpreter, a structured graphics system and input events. A distributed architecture has been used. A central temporal logic specification runs under the interpreter. This communicates with client processes that provide the graphical environment presented to each user. Communication is via the UNIX socket mechanism [28]. Figure 2 illustrates the architecture of the Prelog prototyping system. The implementation and application of this tool is described in [19, 23]. To illustrate the use of the tool, the following clause can be directly executed by the Prelog system:

$$
\begin{aligned}
&\mathit{select_steam_circuit_high_shut_down} :- \\
&\quad \mathit{multi_user_input_event}(\mathit{user_1}, \mathit{pump_a_icon}, \mathit{on_select}), \\
&\quad \mathit{until}(\mathit{not}(\mathit{value}(\mathit{pump_a_press}, \mathit{min_pump_a_press})), \\
&\quad \mathit{multi_user_input_event}(\mathit{user_2}, \mathit{pump_a_icon}, \mathit{on_select})).
\end{aligned}
$$

There is a close correspondence between this clause and *Select_Steam_Circuit_High_Shut_Down*. In general, however, it is not possible to directly derive executable clauses from Z specifications. The abstract nature of many requirements would prevent partial execution during the early stages of development. A number of techniques have been proposed to ease the refinement towards the stage where prototypes may be developed [29]. This is a research area in its own right and until this work is more fully developed there is a heavy burden upon designers who must derive partial implementations from initial specifications. Having raised this caveat we believe that Prelog's temporal operators as well as its graphical and device abstractions ease the process of validation through prototyping.

8 Conclusions And Further Work

Interface requirements are not, typically, considered within formal specifications. This threatens the safety of many applications. A prime objective in the use of formal methods is to minimise the modifications that are necessary once a specification has been refined towards implementation. Usability considerations are, therefore, relegated to an afterthought in the development process. This paper has argued that interface requirements might be explicitly represented within formal specifications.

A number of limitations restrict the utility of formal notations for interface development. Many specification languages abstract away from the temporal properties that affect usability. A further problem is that designers must represent input from a range of interaction devices within formal specifications. Finally, graphical presentation techniques cannot easily be represented within abstract requirements. Temporal logic, structured graphics and input events have been introduced to address these problems. The logic formalism extends the Z notation to represent changes in the state of a system. It supports a range of temporal operators that can express synchronisation properties between an interface and an underlying application. These operators can be used to express synchronisation properties in multi-user interfaces. Structured graphical representations provide a declarative means of representing complex images. Generic events provide a means of describing operator input without specifying the physical devices that are used to interact with a system. Implementation decisions can be postponed until later in the development cycle. A prototyping environment has been briefly introduced. This provides the means of validating abstract interface requirements.

There remain a number of issues that must be addressed before specification languages, such as Z, can be used to represent and reason about interactive control systems. For example, this paper has not addressed the problems of proof or verification in relation to interactive properties. The formal manipulation of a specification must not hide inappropriate assumptions about operator behaviour. These issues are explored elsewhere [20]. Similarly, we have not addressed the issue of refinement [33]. There remains a divide between the dynamic properties specified using temporal operators and the instructions of an implementation language. It should be noted, however, that the introduction of sequential information using the *History* schema does help to capture the procedural information which is necessary in order to implement an interactive system. Similarly, the level of detail provided by input events and graphical regions has been carefully selected to provide a convenient bridge towards the abstractions supported by prototyping tools and screen management systems [44].

Acknowledgements

Thanks are due to Leonor Barroca, Faculty Of Mathematics, Open University, Andy Dearden and David Duke, Department of Computer Science, University of York. They provided valuable advice about the integration of temporal logic and Z. Thanks are also due to the reviewers; their comments encouraged me to directly address many issues that were avoided in early drafts of the paper.

References

[1] J. Borer. *Instrumentation And Control For The Process Industries.* Elsevier Applied Science, London, United Kingdom, 1985.

[2] J. Bowen. Specification of timing constraints. *Z Electronic Forum Newsletter*, 1(6), May 1986.

[3] J. Bowen. X: Why Z? *Computer Graphics Forum*, 11(4):221–234, October 1992.

[4] J. Bowen and V. Stavridou. Safety-critical systems, formal methods and standards. *Software Engineering Journal*, pages 189–209, July 1993.

[5] J. Burgoyne, editor. *The Assessment And Control Of Major Hazards.* Pergamon Press, Oxford, United Kingdom, 1985.

[6] C.T. Burton, S.J. Cook, S. Gikas, J.R. Rowson, and S.T. Sommerville. Specifying the Apple Macintosh Toolbox Event Manager. *Formal Aspects of Computing*, 1:147–171, 1989.

[7] A. Coombes and J.A. McDermid. Specifying temporal requirements for distributed real-time systems in Z. Technical Report YCS 176, Department of Computer Science, University of York, York, United Kingdom, 1992.

[8] N. Delisle and D. Garlan. Formally specifying electronic instruments. In S.J. Greenspan, editor, *The 5th International Workshop On Software Specification And Design*, pages 242–249. IEEE Computer Society Press, Washington, United States of America, 1989.

[9] A. Diller. *Z: An Introduction To Formal Methods.* John Wiley And Sons, London, United Kingdom, 1990.

[10] D. Duke and R. Duke. A history model for classes in Object-Z. Technical Report 120, Key Centre for Software Technology, Dept. of Computer Science, University of Queensland, November 1989.

[11] D. Duke and M.D. Harrison. Abstract models for interaction objects. Technical Report Amodeus Project Document: System Modelling/WP 1, Department of Computer Science, University of York, York, United Kingdom, 1993.

[12] I.A.R. Galer and B.L. Yap. Ergonomics in intensive care: Applying human factors to the design and evaluation of a patient monitoring system. *Ergonomics*, 23(8):763 – 779, 1980.

[13] I. Greif and S. Sarin. Data sharing in group work. *Communications Of The ACM*, 5(2):197–211, 1987.

[14] M.D. Harrison and H.W. Thimbleby, editors. *Formal Methods In Human Computer Interaction.* Cambridge University Press, Cambridge, United Kingdom, 1989.

[15] B. Hills, D. Boettcher, and R. Reed. Sizewell operators face a friendlier control panel. *ATOM*, (428):14–17, May/June 1993.

[16] *ISO DIS8807: LOTOS - A Formal Description Technique*, ISO-TC 97-SC21, 1987.

[17] International Atomic Energy Agency and The Commission of the European Community. *Critical Survey of Research On Human Factors And The Man-Machine Interaction*, IAEA-SM-26B/29, Vienna, Austria, 1984.

[18] J. Jacky. Formal specification for a clinical cyclotron control system. In M. Moriconi, editor, *Proceedings Of The ACM SIGSOFT International Workshop On Formal Methods In Software Development*, pages 45–54. ACM, New York, United States of America, 1990.

[19] C.W. Johnson. Applying temporal logic to support the specification and prototyping of concurrent multi-user interfaces. In D. Diaper and N. Hammond, editors, *People And Computers VI: Usability Now*, pages 145–156. Cambridge University Press, Cambridge, United Kingdom, 1991.

[20] C.W. Johnson. *A Formal Approach To The Integration Of Human Factors And Systems Engineering.* PhD thesis, Department Of Computer Science, University of York, York, United Kingdom, 1992.

[21] C.W. Johnson. A probabilistic logic for the development of safety-critical interactive systems. *International Journal Of Man-Machine Studies*, 39(2):333–351, 1993.

[22] C.W. Johnson and M.D. Harrison. PRELOG - a system for presenting and rendering logic specifications of interactive systems. In C.E. Vandoni and D.A. Duce, editors, *EUROGRAPHICS '90*, pages 469–480. Elsevier Science Publications, North Holland, Netherlands, 1990.

[23] C.W. Johnson and M.D. Harrison. Using temporal logic to support the specification and prototyping of interactive control systems. *International Journal Of Man-Machine Studies*, 36:357–385, 1992.

[24] V. De Keyser. Temporal decision making in complex environments. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 121–128. Clarendon Press, Oxford, United Kingdom, 1990.

[25] M. Kirkpatrick and K. Mallory. Substitution error potential in nuclear power plant control rooms. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 163–167. Human Factors Society, Santa Monica, United States of America, 1981.

[26] W. Kuhmann. Stress inducing properties of system response times. *Ergonomics*, 32(3):271 – 280, 1989.

[27] P. Ladkin. Primitives and units for time specifications. In T. Kehler, S.Rosenschein, R. Filman, and P.F. Patel-Schneider, editors, *Proceedings Of The Fifth Annual Conference On Artificial Intelligence*, pages 354–359. Morgan Kaufman, Los Altos, United States of America, 1986.

[28] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman. *The Design And Implementation Of The 4.3BSD UNIX Operating System*. Addison Wesley, Reading, United States of America, 1990.

[29] F. Lin and F.E. Hunt. LCD-Reification: A formal method for developing PROLOG programs. In S.J. Greenspan, editor, *The 5th International Workshop On Software Specification And Design*, pages 249–256. IEEE Computer Society Press, Washington DC, United States of America, 1989.

[30] B.P. Mahony and I.J. Hayes. A case-study in timed refinement: A mine pump. *IEEE Transactions On Software Engineering*, SE-18(9):817–826, 1992.

[31] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R.S. Boyer and J. Strother Moore, editors, *The Correctness Problem In Computer Science*, pages 215–273. Academic Press, London, United Kingdom, 1981.

[32] Z. Manna and A. Pnueli. Verification of concurrent programs: Temporal proof principles. In D. Kozen, editor, *Logic Of Programs 1981 - Proceedings*, LNCS 131, pages 200–252. Springer-Verlag, Berlin, FDR, 1982.

[33] C. Morgan. *Programming From Specifications*. Prentice Hall, London, United Kingdom, 1990.

[34] National Aeronautic and Space Administration. *Advanced Orbiting Systems - Architectural Specification For The CCSDS Secretariat*, Washington DC, United States of America, 1989.

[35] D.A. Norman. *The Psychology Of Everyday Things*. Basic Books, New York, United States of America, 1988.

[36] D.A. Norman. The 'problem' with automation : Inappropriate feedback and interaction not 'over-automation'. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 137–145. Clarendon Press, Oxford, United Kingdom, 1990.

[37] President's Task Force On Aircraft Crew Compliment, United States' Government. *Report On Aircraft Crew Compliment*, Washington DC, United States of America, 1981.

[38] J. Rasmussen and M. Lind. Coping with complexity. Technical Report Risø-M-2293, The Risø National Laboratory, Roskilde, Denmark, 1981.

[39] N. Siu and C. Acosta. Dynamic event tree analysis - an application to SGTR. In G. Apostolakis, editor, *Probabilistic Safety Assessment And Management*, volume 1, pages 413–418. Elsevier, London, United Kingdom, 1991.

[40] J.M. Spivey. *Understanding Z, A Specification Language And Its Formal Semantics*. Cambridge University Press, Cambridge, United Kingdom, 1988.

[41] M. Stefik, G. Foster, D.G. Bobrow, K. Hahn, S. Lanning, and L. Suchman. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications Of The ACM*, 30(1):32–47, 1987.

[42] B. Sufrin and J. He. Specification, refinement and analysis of interactive processes. In M. D. Harrison and H. W. Thimbleby, editors, *Formal methods in Human Computer Interaction*, pages 153–200. Cambridge University Press, Cambridge, United Kingdom, 1990.

[43] H.W. Thimbleby. Delaying commitment. *IEEE Software*, 5(3):78–86, May 1988.

[44] R. Took. Integrating inheritance and composition in an objective presentation model for multiple media. In F.H. Post and W. Barth, editors, *EUROGRAPHICS '91*, pages 291–303. Elsevier Science Publications, North Holland, Netherlands, 1991.

[45] I.A. Watson. Review of human factors in reliability and risk assessment. In J. Burgoyne, editor, *The Assessment And Control Of Major Hazards*, pages 323–337. Pergamon Press, Oxford, United Kingdom, 1985.

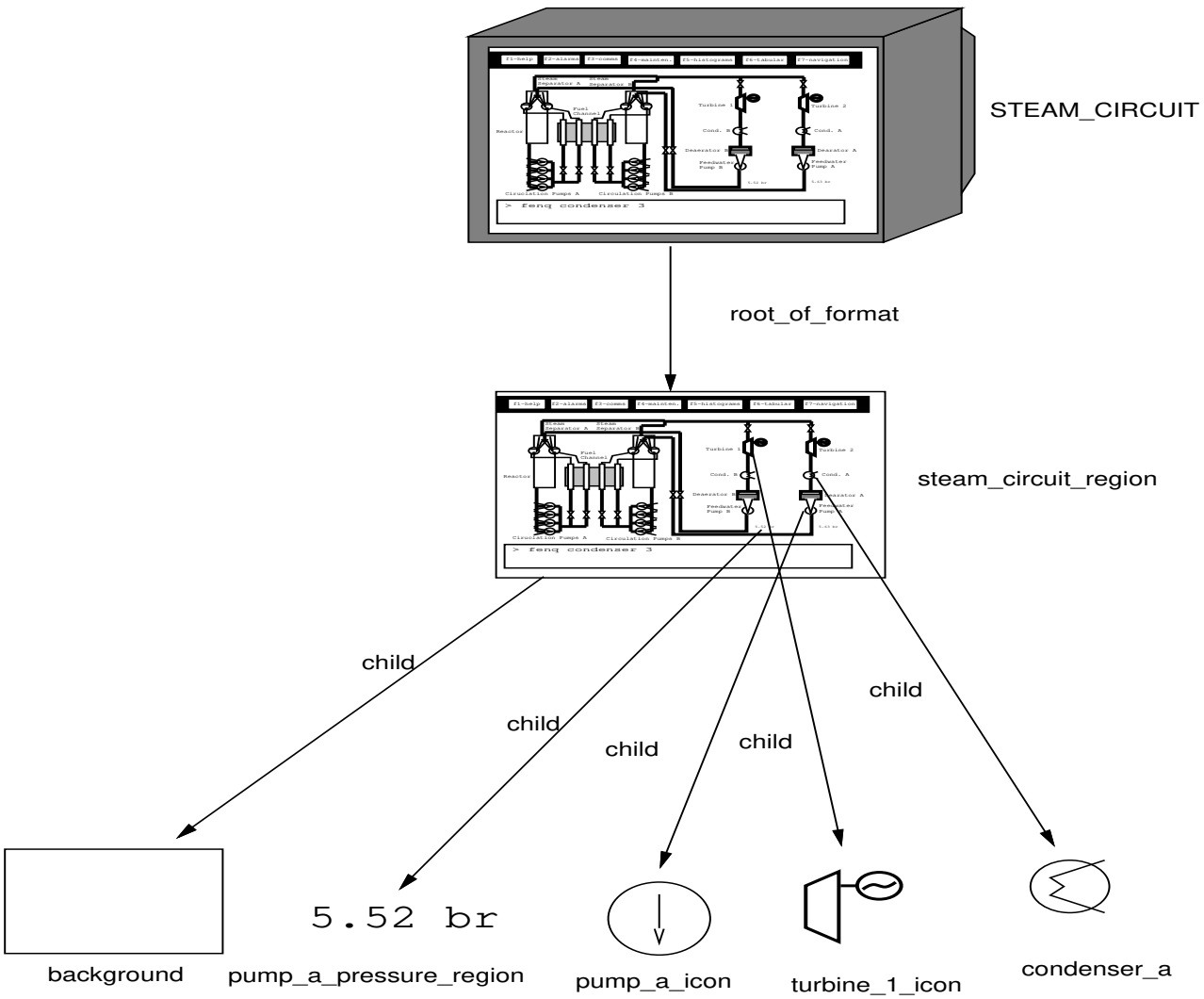

Figure 1: Decomposition of the feedwater display.

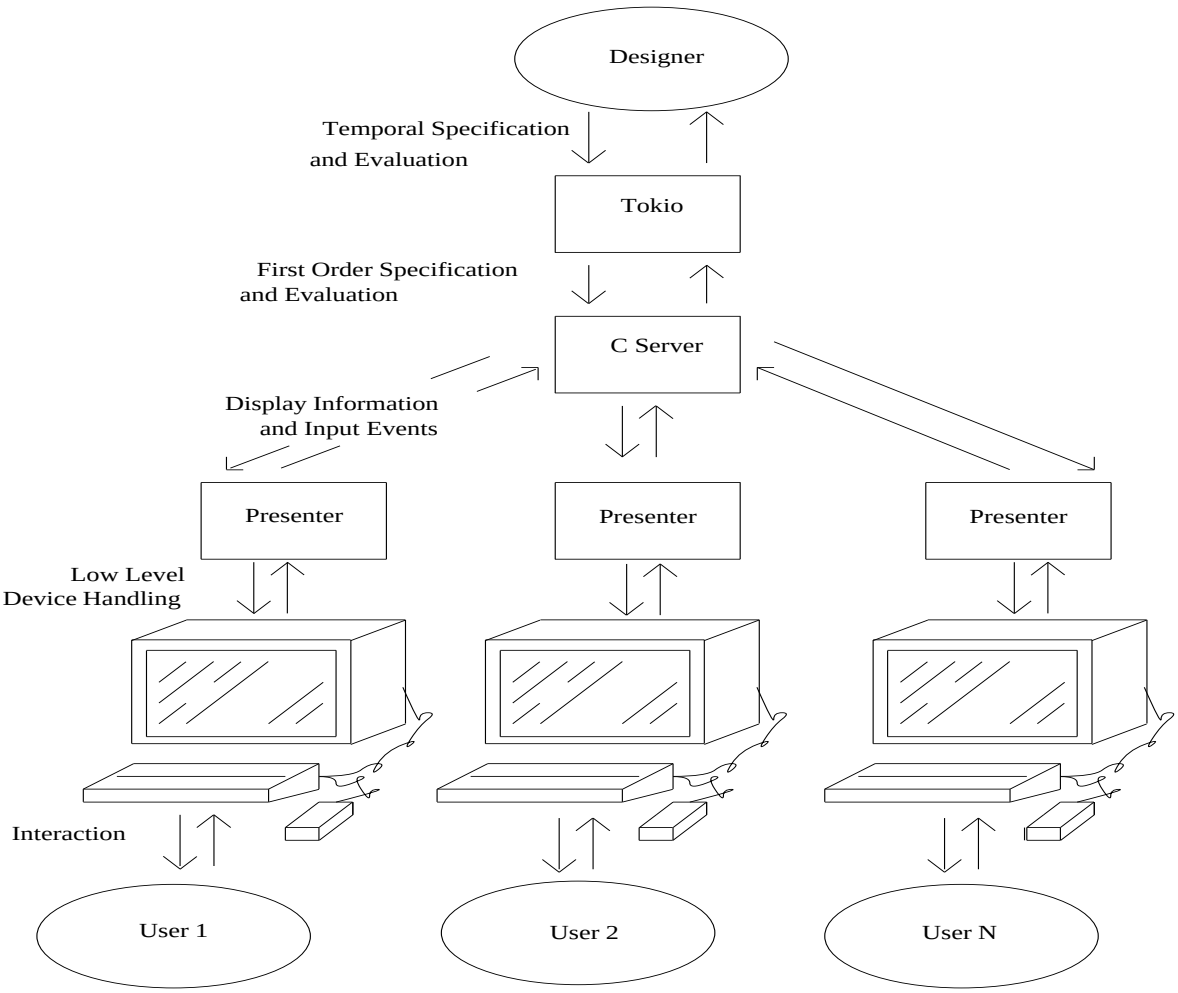

Figure 2: The revised Prelog architecture