Analysing and modelling context in mobile systems to support design

Karsten Loer[1] and Michael D. Harrison[2]

[1]Department of Computer Science, University of York
York, YO10 5DD, UK and
[2]Informatics Research Institute, University of Newcastle upon Tyne,
NE1 7RU, UK
Karsten.Loer@cs.york.ac.uk | Michael.Harrison@ncl.ac.uk

Abstract Mobility of ubiquitous systems offers the possibility of using the current context to infer information that might otherwise require user input. This can either make user interfaces more intuitive or cause subtle and confusing mode changes. We discuss one approach to the analysis of such systems that will allow the designer to predict potential pitfalls before the design is fielded. Whereas the current predominant approach to understanding mobile systems is to build and explore experimental prototypes, our exploration highlights the possibility that early models of an interactive system might be used to predict problems with embedding in context *before* costly mistakes have been made. Model checking is used to perform exhaustive analysis of statechart models of two alternative interfaces to a sewage plant control system.

1 Introduction

Ubiquitous mobile devices provide a number of opportunities for improving the interface between a user and some service or application. However, mobility may also lead to surprising consequences for interaction that were not intended or envisaged by the designer. A particular set of these possible consequences are the *modal* effects introduced by a device's position within its spatial context. These modal effects occur because action makes use of information about the context that may not be transparent to the user.

Context and context awareness in mobile systems can therefore either contribute to the seamlessness of an interaction or make the interface more confusing and opaque to the user. While several systems have been developed to explore these ideas, and there are a number of explorations of the concepts of context, see [Abowd and Mynatt, 2000] and [Dix et al., 2000], there has been relatively little work on the analysis that might be performed to predict some of the consequences of design decisions in an emerging design. The paper explores how model checking might be used to analyse these effects using a case study involving the replacement of a centralised control room interface by a hand-held system. Whereas the control room provides all information about the plant in one format, the hand-held system makes use of positional information, using

mode to offer options to the operator and provides more limited display. Display real-estate constraints imposed by the hand-held device are compensated for by only displaying information that is relevant to the particular context.

Mobile devices have potential to enrich the work and productivity of users. In the context of process control systems, where the consequence of error might be expensive or disastrous, mobile devices can facilitate the movement of an operator through the plant while continuing to enable the operator to observe the plant as a whole, thus potentially freeing people from being confined to the control room. But the consequences of unforeseen interaction problems that might arise from the additional design complexity could be high. The approach is illustrated using a case study involving a sewage plant (Fig. 1). It is assumed that in an existing system the operator monitors and controls tank status and fluid flows centrally. An alternative version is being developed that uses a mobile hand-held device both to gather information about the plant and to control it.

When a problem arises in the plant with the existing system, the controller can telephone a worker who happens to be near to where the problem is in order to investigate it. It is proposed that new hand-held technology will, if properly designed, provide staff in the plant with the means of monitoring and controlling the plant at the same time while being on site. Achieving the same goals in the mobile system is different from the central control system for a variety of reasons including the need to compensate for different characteristics of the mobile device display. Each of these differences provides a possible source of error, which needs exploring in detail, possibly using the services of a human factors expert. One way of doing this is to use models to compare sequences of interactions that occur within the centralised control room to achieve specified goals with an equivalent sequence of interactions to achieve the same set of goals using the mobile device. In both cases the interfaces support alerting mechanisms, the ability to view information about the state of the plant, and the ability to perform actions perhaps in response to a particular alert.

In modelling any interactive system, a distinction is made between a *device* (an appliance or the software of a desk top computer, for example) and the *system* which may involve people, devices and the environment. Both interface variants for the case study also require a distinction between the interactive system that controls the process and the process itself. The industrial process is a fundamental part of context and, for the hand-held interface, the spatial situation in which the device is placed must also be modelled. The two interfaces use context in different ways: the mobile interface infers mode from physical context; the central interface is designed to be decontextualised and separated from the physical details of the plant.

For pragmatic reasons – the notation was used by a sponsor of this work (BAE Systems) – the model of system behaviour is formulated in terms of statecharts [Harel, 1987]. This notation has the advantage that it has been used to explore interfaces, and in particular moding effects, see [Horrocks, 1999] and [Degani, 1996]. Degani's OFAN approach in particular uses additional structure to decompose models of interactive systems into orthogonal sub-states represent-

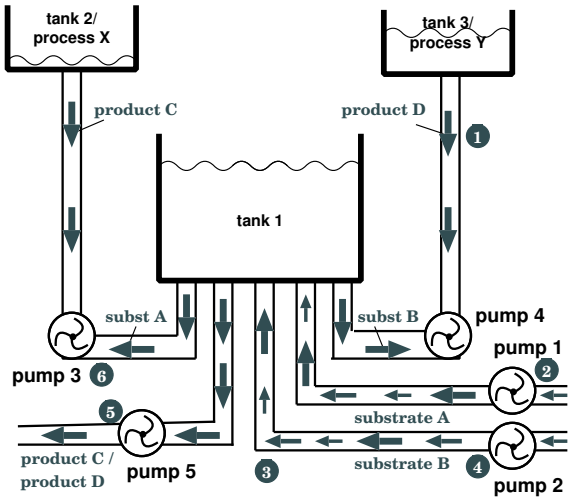


Figure 1. Example: A processing plant.

ing *control elements*, *control mechanism* (the model of the device functionality), *displays* (a description of the output elements), *environment* (a model of relevant environmental properties) and *user task* (the sequence of user actions that are required to accomplish a certain task). The aim of structural decomposition is to ease thinking about the interactive behaviour of the system as well as its refinement.

Section 2 describes a model of plant behaviour and the following section gives descriptions of models of the two different control devices, a central control panel (3.1) and a hand-held control device (3.2). The analysis section (Section 4), while briefly mentioning the need to analyse plant-specific properties, focuses on usability properties and the suitability of the design in supporting the processes in the plant. These analyses are performed on *open* system models, i.e., models that contain variables whose behaviour is controlled by external processes that are not modelled. The behaviour of such variables, which may represent user inputs to control devices, is explored exhaustively in the course of the model-checking analysis. An important issue in discussing the models will be how simplifying assumptions were made to reduce the number of states to make models tractable. These simpler models should not lose those features of the model that enable a predictive analysis of the envisaged systems. Section 4 closes with an analysis of the system adequacy with respect to specific tasks. For this purpose it is demonstrated how specific user tasks and potential deviations can be described as part of the model.

2 The plant model

The plant to be controlled (Fig. 1, numbers in circles denote physical locations of controls in the plant), is based on the processes in the biological stage of a sewage plant. Although the details are not important to the argument of this paper, they are introduced to add realism to the process to be controlled by the interactive system. While the details are fictitious, the underlying moding structure shares features with aspects of a fast jet cockpit system analysed for BAE Systems. The aim is that the reader may draw general conclusions about the analysis process from this example.

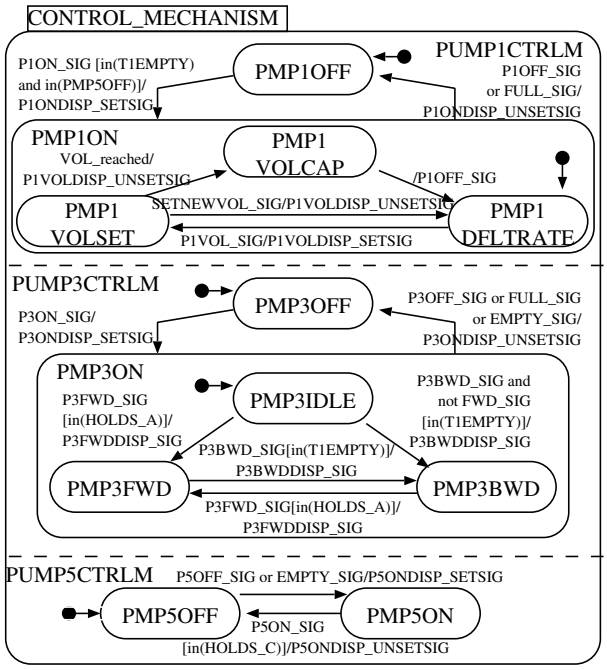
Two different biochemical processes, X and Y, are contained in *tank 2* and *tank 3*. These processes are fed with two substrates, A and B. Process X generates product C when supplied with substrate A. The bacteria in this process are highly temperature and substrate sensitive, so the process must be supplied with substrate A only at a certain range of temperatures. An earlier version of the plant involved a single tank (1) supporting a single process (X). Because of demand for a new substance B that can be produced by the existing infrastructure, a new process Y is introduced that generates product D when supplied with substrate B, but requires a different temperature range than the original process.

Because it is uneconomic to install process Y in a different plant it is proposed (see Fig. 1) that *tank 1* is shared between the processes and that there is a facility to regulate the temperature of the substrates before entering and after leaving processes X and Y. The flows therefore are organised as follows: substrate A can be introduced into the system by pump 1 while substrate B can be introduced by pump 2. Pumps 3 and 4 are bi-directional which means that substances are transported from tank 1 to processes Y and Z in "forward" mode and from processes to tank 1 in "backwards" mode. End products leave the plant using pump 5. In addition pumps 1 and 2 are equipped with an optional `VOLUME` mode that triggers an automatic stop when a selected target volume of the tank is reached.

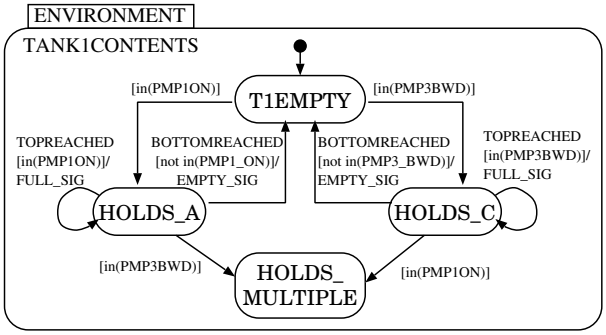
The plant is designed to satisfy a minimal set of safety requirements which any model should also satisfy, namely:

`PSR1:` Substrates A and B must never leave the system unprocessed.
`PSR2:` If its feeding tank is empty a pump must be shut down to prevent damage.
`PSR3:` If the capacity of tank 1 is reached, the pump that currently feeds it must be shut down immediately to prevent an overflow.
`PSR4:` Tank 1 must never hold more than one substance at any time.

The OFAN model that describes the behaviour of the plant is given in Fig. 2 and is designed to take requirements `PSR1` to `PSR4` into account. Part of the process of analysing this system involves checking that these properties hold. The model is *open* in the sense that some events are not controlled inside the environment model. Hence `T1TOPREACHED`, `T1BOTTOMREACHED` and `T1VOL_REACHED` in Fig. 2 can be fired arbitrarily during any analysis. Since pumps 2 and 4 are precisely analogous to the specifications of pumps 1 and 3 they are not described.



a) `CONTROL_MECHANISM` module



b) `ENVIRONMENT` module

Figure 2. Plant model.

Only the *change* of the volume is specified in pumps 1 and 2 because actual *value* of the volume setting is irrelevant here. The OFAN modelling technique requires splitting the specification into control mechanisms and environment. The pump logic is modelled in the `CONTROL_MECHANISM` part of the model (Fig. 2a). The model of tank 1 describes the `ENVIRONMENT` controlled by the process (Fig. 2b) and provides a simple and discrete model of what it means for the tank to hold substances, to be full and to be empty. No constraints are placed on tanks 2 and 3 and therefore they are not modelled explicitly.

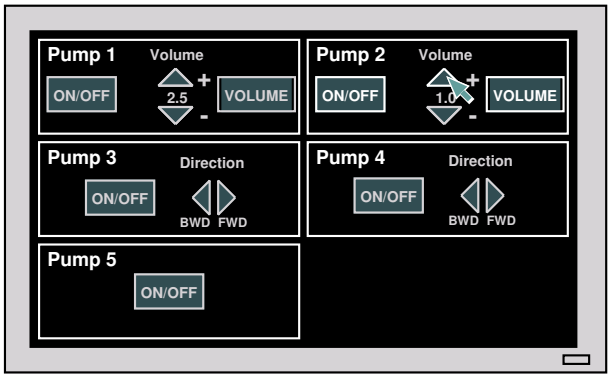


Figure 3. Control Screen layout.

Tank 1 is described in terms of a minimum number of discrete changes. These discrete changes are designed to preserve significant features of its behaviour while facilitating efficient model checking. The tank either holds A, B, C or D exclusively, or it can contain multiple substances. The tank can also be in a state where the top is reached in any of these conditions in which case a full signal is sent. When the bottom is reached the state of the tank becomes empty and an empty signal is sent. Hence the model is simple but captures the extreme conditions of the system.

3 Modelling the user interface

The interactive system that controls the process should reflect the following behaviour in the interactive system: (1) to inform the operator about progress; (2) to allow the operator to intervene appropriately to control the process; (3) to alert the operator to alarming conditions in the plant and (4) to enable recovery from these conditions. A model is required to explore usability issues and design alternatives in the light of achieving particular system goals. The central control mechanism provides all information in one display (Section 3.1), while the personal appliance displays partial information (Section 3.2).

3.1 Representing and modelling the central panel

The control room, with its central panel, aims to provide the plant operator with a comprehensive overview of the status of all devices in the plant. The paper confines itself to modelling aspects that deal with availability and visibility of action. Other aspects of the problem can be dealt with by using complementary models of the interface, for example alarms structure and presentation. How we ensure that these complementary models are consistent is beyond the scope of this paper. The specification describes the behaviour of the displays and the associated buttons for pump 1 (and equivalently pump 2). The effects of actions

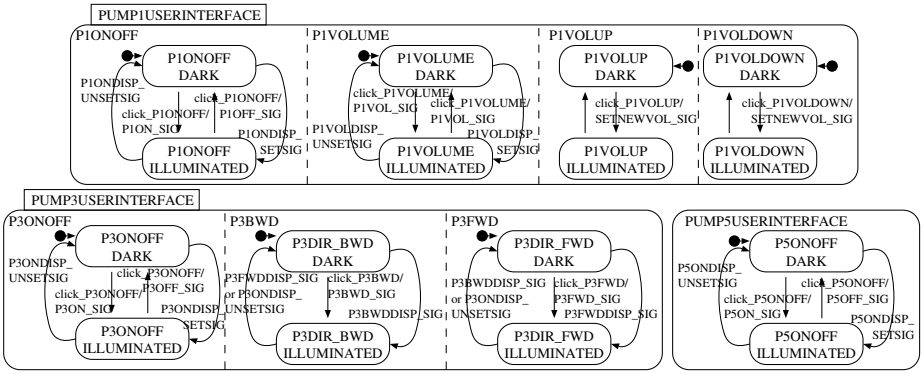


Figure 4. Initial specification of control screen behaviour.

are described in terms of the signals that are used to synchronise with the pump description and the states in which the buttons are illuminated.

The control panel is implemented by a mouse-controlled screen (see Fig. 3). Screen icons act as both displays and controls at the same time. Control Elements and Displays are combined in a single User Interface (UI) module (Fig. 4) by AND-state composition of statecharts. `PUMP1USERINTERFACE` therefore supports four simple on-off state transitions defining the effect of pressing the relevant parts of the display.

3.2 Representing context and the hand-held control device

The hand-held control device (Fig. 5) reflects its position within the spatial organisation of the plant. Hence the Environment model to describe the system involving this device is extended to take location into account. A simple discrete model describes how an operator can move between device positions in the plant (denoted by circled numbers in the plant diagram in Fig. 1). The user's move-

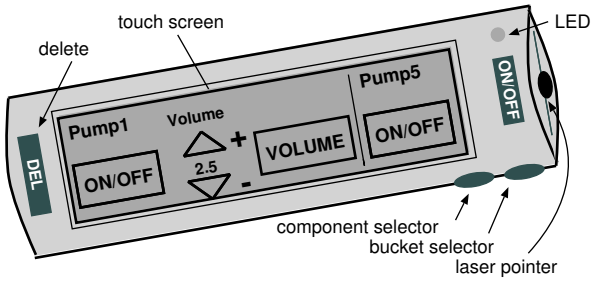


Figure 5. A hand-held control device (modified version of the "Pucketizer" device in [Nilsson et al., 2000]).

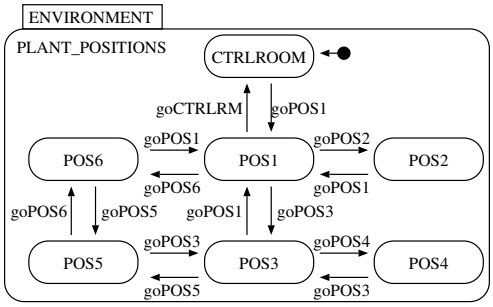


Figure 6. Model of device positions.

ments are modelled as transitions between position states, as shown in Fig. 6.

The hand-held control device "fills" so called virtual buckets on the display (see [Nilsson et al., 2000] for a description of the original device). These buckets can be used to capture information about plant components (i.e. different types of pumps, valves and displays) that are encountered by the operator during their rounds. By pointing the laser pointer at a plant component and pressing the component selector button, the status information for that component and soft controls are transferred into the currently selected bucket. Components can be removed from a bucket by pressing the delete button. With the bucket selector button the user can cycle through buckets. The original Pucketizer can also be used to record and play audio messages that are "attached" to components in the plant in order to remind the operator or inform colleagues of issues related to that device. The intended use of the device has been altered from the published description (monitoring and annotating) to monitoring and manipulation. Appropriate simplifications have been made to the model for the purposes of explanation, for example the audio processing facility is not being considered. Control elements for the manipulation of components in a bucket shall be displayed on a touch screen (as opposed to the plain display of the original Pucketizer), with a display area that is limited to controls for up to two pumps at a time.

The OFAN model of the hand-held device describes both the physical buttons that are accessible continuously and other control elements, like pump control icons, that are available temporarily and depend on the position of the device. When the operator approaches a pump, its controls are automatically displayed on the screen (it does not require the laser pointer). The component may be "transferred" into a bucket for future remote access by using the component selector button. Controls for plant devices in locations other than the current one can be accessed remotely if they have been previously stored in a bucket. When a plant component is available in a bucket and the bucket is selected, the hand-held device can transmit commands to the processing plant, using the pump control icons.

value of B1CONTENT	content of bucket 1	value of B2CONTENT	content of bucket 2	value of B3CONTENT	content of bucket 3
0	–	0	–	0	–
1	P1	1	P3	1	P5
2	P3	2	P5	2	P1
3	P1 & P3	3	P3 & P5	3	P1 & P5

Table 1. Encoding of sample "bucket" configurations.

Fig. 11 (see Appendix) shows an extract of the User Interface and Control Mechanism modules for the hand-held device. Here the user can choose between three buckets and each bucket can store controls for up to two components. In the BUCKETS state the current contents of each bucket x are encoded by variables "BxCONTENT". The encoded configurations in Table 1 cover many situations that are relevant for the further analysis.

An Environment module completes the model which is given by AND composition of the tank content model (Fig. 2b) and the device position model in Fig. 6.

The model described here presumes that the appliance should always know where it is in the network. This is of course a simplification. Alternative models would allow the designer to explore interaction issues when there is a dissonance between the states of the device and the states of the environment. A richer model in which variables are associated with states, and actions may depend on values of the state that have actually been updated, may lead to asking questions of the models as whether "the action has a false belief about the state". These issues are important but are not considered in this paper.

4 Analysis

Model-checking is a technique for analysing whether a system model satisfies a requirement. These requirements may be concerned with a number of issues including safety and usability. The model checker traverses every reachable system state to check the validity of the given property. If the property "holds", a True answer is obtained. Otherwise, the property is False, and the tool attempts to create a sequence of states that lead from an initial state to the violating state. These "traces" are a valuable output because they help understanding *why* a specification is violated. There are many detailed expositions of approaches to model checking, see for example [Clarke et al., 1999], [Huth and Ryan, 2000], [Bérard et al., 2001] or [Holzmann, 2003].

4.1 Properties to be checked

Typical properties that would be checked for the system described in this paper include questions about:

- system sanity: can an end-product be produced at all and does the model reflect the reality of the system to be designed?
- safety and reliability: can an unsafe state be reached and how can the system recover from error?
- usability and efficiency: how do different interfaces influence the productivity of the process?

General usability properties are first considered here including the comparison of alternative designs in terms of the paths discovered by the checker for specific properties. A general criterion of usability is first considered, followed by an analysis of the visibility of aspects of the system before considering a reachability property.

Usability: The central control panel interface of the plant can be operated more efficiently if valid combinations of pumps are operated concurrently. That the hand-held device supports this mode of operation adequately is therefore a matter of importance. On the one hand a safety analysis (for example [Leveson, 1995]) establishes that invalid combinations of pumps are ruled out and, on the other hand, an exploratory analysis based on model checking shows how valid combinations can be reached. By exhaustively exploring whether unsafe pump combinations are possible the model checker produces a trace that shows that pump 3 can be operated in BWD mode while pump 5 is ON (see final steps of sequence 4 in Fig. 7[1]). Such a configuration can be reached by checking the negated property. In the given design, which has a fixed bucket configuration, pump 3 is stored in bucket 2 while pump 5 is stored in bucket 3. A trace is generated in which the desired combination of pumps is achieved by switching between buckets. The model checker helps to demonstrate that either both pumps are stored and selected from buckets, or one pump is stored and the control elements of the other pump override previously displayed information as soon as that pump is approached by the user (sequence 5 in Fig. 7). In both cases only the control elements for a single component are visible at any time. This problem might be resolved by modifying the design so that controls for multiple components can be stored in a bucket – this is likely to require a scrolling mechanism, because not all control elements can be displayed on the limited screen of the hand-held device.

Status and operation visibility: The icons on the hand-held device are the only means available to the user to infer the current system state and the available operations. The visibility of icons therefore is important for the operation of the plant and the usability of the hand-held device. Two concerns for the status

[1] The traces in Fig. 7 are filtered and pretty-printed to illustrate the key information of interest with respect to this paper; the standard outputs of the model checker – ASCII text files or data tables – are not easy to read, see [Loer, 2003, Chapter 6] for a discussion.

and operation visibility analysis are (i) that all available operations are visible, and (ii) that all visible operations are executable.

The OFAN model of the hand-held device defines that a function is *visible* if its control elements are in the visible state. An operation for a pump is *available* if either the user is at the location of that pump, or if the controls for that pump are stored in one of the buckets (see Section 4). The device's position in the plant is recorded by the environment (Fig. 6), and the allocation of components to buckets is recorded by the history variables `B1CONTENT`, `B2CONTENT` and `B3CONTENT`.

With this knowledge, it can be checked that an operation that is available (antecedent state) always becomes visible (consequent state). Take pump 3, for example. The pump is located in position 6 and may be stored in bucket 1, which is indicated by a value of `B1CONTENT` greater than one, according to the encoding in Table 1. Similarly, the encoding that corresponds to the situation where pump 3 controls are stored in bucket 2, is given by the condition "`B2CONTENT` $mod\ \ 2\ =\ 1$" since in that situation either pump 3 alone is in the bucket or pump 3 and pump 5 are in there. The property that requires that if pump 3 is stored in a bucket or is directly accessible then its controls are visible is:

```
AG ( ( (BUCKETS.B1CONTENT>1)
      |(BUCKETS.B2CONTENT mod 2 = 1)
      |(POSITIONS.state=POS6))
   -> AF(PUMP3_CONTROLS.state=P3CTRL_VIS))
```

To check that all visible operations are executable, the same general form can be used repeatedly. This time, the antecedent state is given by the visible state and an input action, whereas the consequent state is instantiated with a desired system status. For example, selecting the ON/OFF icon of pump 3 when it is visible should always switch pump 3 ON:

```
   AG(((PUMP3_CONTROLS.state=P3CTRL_VIS)
        & TAP_P3ONOFF)
      -> AF(PUMP3CTRLM.state=PMP3ON))
```

The property does not hold and the model-checking trace demonstrates that user input on screen is not processed when at the same time a different bucket is being selected. Therefore, the analyst might wish to check that the property holds *before* a different bucket is selected. For the current example the user would choose the situation that the pump 3 controls are visible and a different bucket is selected: "`(PUMP3_CONTROLS.state=P3CTRL_VIS) & BSLCT_SIG`" which would generate the more complex property expression:

```
!E[(!((PUMP3_CONTROLS.state=P3CTRL_VIS) & BSLCT_SIG))
    U((!((PUMP3_CONTROLS.state=P3CTRL_VIS) & BSLCT_SIG))
      &(!(((PUMP3_CONTROLS.state=P3CTRL_VIS) & TAP_P3ONOFF)
         &!A[(!((PUMP3_CONTROLS.state=P3CTRL_VIS)&BSLCT_SIG))
            U((PUMP3CTRLM.state=PMP3ON)
```

```
            &(!((PUMP3_CONTROLS.state=P3CTRL_VIS)
               & BSLCT_SIG)))]))
      & EF((PUMP3_CONTROLS.state=P3CTRL_VIS) & BSLCT_SIG))]
```

sequence 1:	Control room interface, serial use of pumps.
sequence 2:	Hand-held control device, serial use of pumps.
sequence 3:	Control room interface, concurrent use of pumps.
sequence 4:	Hand-held control device, concurrent use of pumps.
sequence 5:	Hand-held control device, concurrent use of pumps, automatic override.
sequence 6:	Hand-held control device, concurrent use of pumps, "forgetful" operator.
sequence 7:	Hand-held control device with interlock, concurrent use of pumps, interlock yields save operation.
sequence 8:	Hand-held control device with interlock, concurrent use of pumps, interlock ignored.

Table 2. Explanation of Sequences in Fig. reffig:bisimPuckx,

This property still does not hold. The resulting trace demonstrates that a screen input is ignored if it is made when a new device is approached by the user. An automatic update of the display is triggered, and different control elements become invisible. The analyst might opt to rule that situation out in one of three ways:

1. First, the conjunct defining the before-predicate of the property above can be extended by the action `exPOS6` (representing the statechart action `ex(POS6)`, which is fired when state `POS6` is left). The property then states that a signal will be sent successfully if the control elements are visible and the ON/OFF icon is selected before the position is left or a different bucket is selected.
2. Alternatively the analyst might decide to formulate an invariant that states that the position of pump 3 must never be left once it was visited. However this is not a realistic possibility.
3. A further possibility is that the analyst might decide that the display override is acceptable as long as (i) the control elements can be retrieved again by appropriate user actions, and as long as (ii) their controls are visible only when operations are executable. In summary, operations are executable after their controls become visible and until they become invisible again.

The resulting property is true. Sequence 6 in Fig. 7 presents a situation where the user keeps returning to position 6 whenever an input to pump 3 is made. This suggests the possibility that the user might forget to store a pump in a bucket before he moves on to a different device.

Consequently, the analyst might decide that it is necessary to change the design of the user interface. For example, before the currently displayed controls are removed, the user could be informed by a pop-up window that they were not stored yet and therefore will be lost, if the user moves on.

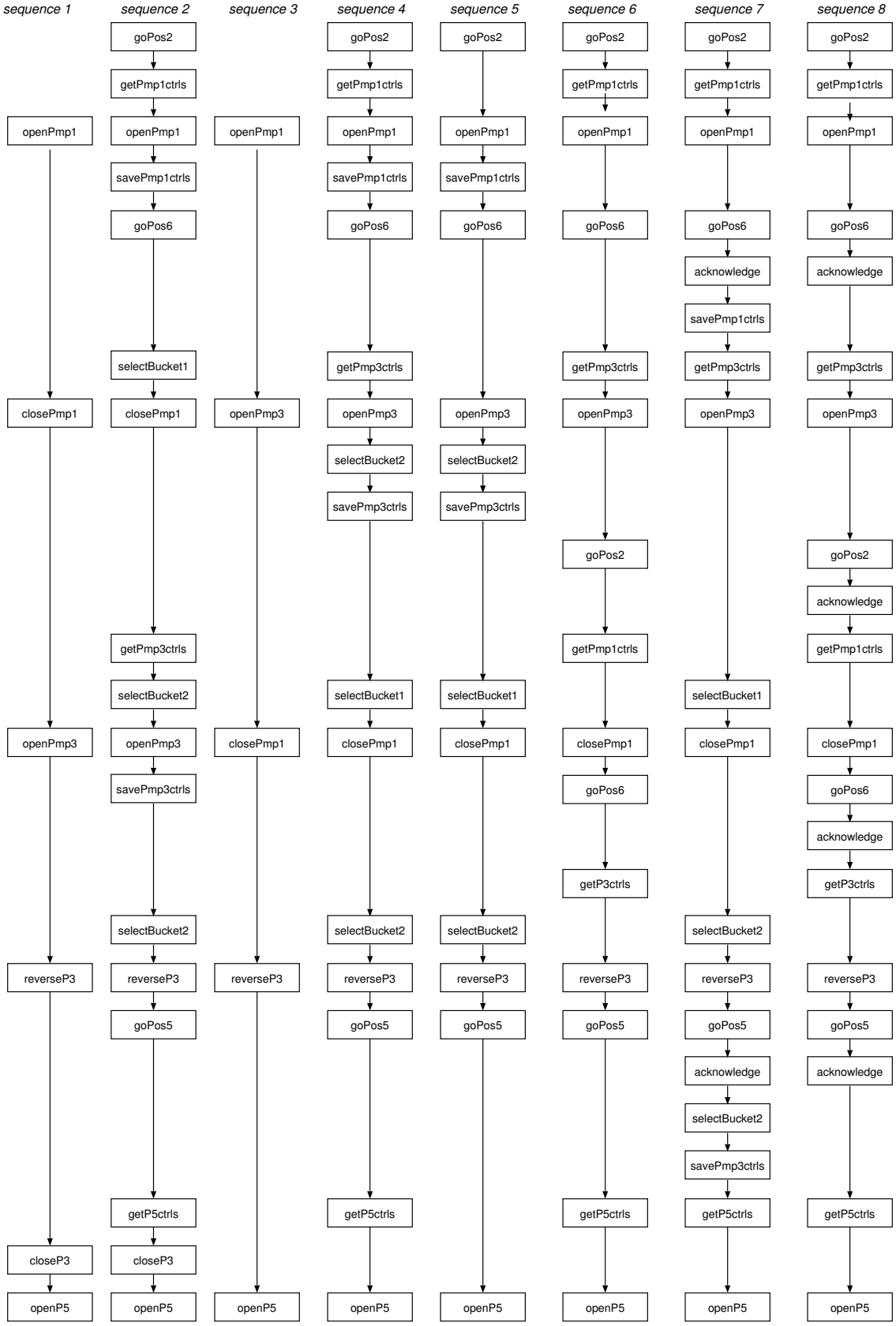


Figure 7. Comparison of behaviours for a goal: "*Produce substance C*". — See Table 2 for explanation.

Reachability: A reachability property may be formulated for a user level "goal" of the system. The goal chosen here for illustration is "*Produce substance C*" which is a primary purpose of the system. If a property does not hold then the checker finds one counter-example. Alternatively, the negated property may be used to find a trace that satisfies the property. Usually the model checker only produces a single trace giving no guarantee that it is an interesting one from the point of view of understanding design implications. Additional traces can be created by adding assumptions about the behaviour. This approach contrasts with the approach given in the next section where the model checker is used to explore a particular way in which the goal can be achieved (the task) because the idea is to use as few behavioural assumptions as possible. We are interested in any behaviours.

The sequences in Fig. 7 are visualisations of the traces obtained by checking for different models if and how the plant can deliver substance C to the outside world. This is specified as property:

```
SAN1:
    F (PUMP5CTRLM.state=PMP5ON)
      & (TANK1.state = HOLDS_C)
```

In this case the negated property "`not SAN1`" is used because instances that satisfy the property are required. The two models involving the different interfaces are checked with the same property. The first sequence in Fig. 7 satisfies the control room interface. The second sequence was generated by checking the property against the hand-held device model. While the first two traces assume a serial use of pumps, the third and fourth sequences show the same task for a concurrent use of pumps. Comparison of these sequences yields information about the additional steps that have to be performed to achieve the same goal and, as a result of this comparison, a further assumption is introduced that an operator might forget certain steps (see Section 4.2) is introduced. Checking this property leads to the sixth sequence and as a result a decision to modify the design of the hand-held device so that an interlock mechanism is introduced to reduce the likelihood that human error might arise. This change to the design (discussed in 4.3) leads to the seventh and eighth sequences.

The central control panel characterises the key actions to achieving the goal since the additional actions introduced by the hand held device are concerned exclusively with the limitations that the new platform introduces, dealing with physical location, uploading and storing controls of the visited devices as appropriate. The analysis highlights these additional steps to allow the analyst to subject the sequence to human factors analysis and to judge if such additional steps are likely to be problematic. The reasons why a given sequence of actions might be problematic may not be evident from the trace but it provides an important representation that allows a human factors or domain analyst to consider these issues. For example, action `goPOS6` may involve a lengthy walk through the plant, while action `savePmp4ctrls` may be performed instantaneously and the performance of action `getPmp3ctrls` might depend on additional contextual factors like the network quality. The current approach leaves the judgement

of the severity of such differences to the designer, the human factors expert or the domain expert. It makes it possible for these experts to draw important considerations to the designer's attention.

The nature of the assumptions that may be added to properties for checking is next considered in more detail.

4.2 Restricting search by adding assumptions

A number of assumptions can further focus the exploration of the model. These may be properties that are analysed alone or may focus further analysis to a subset of paths that satisfy the assumptions. The assumptions may concern:

- robustness of the system to the achievement of goals in the face of failure. An assumption might require that goals can be achieved even when defined failure states eventually occur in every execution path.
- that paths will include given patterns of response by users in given situations.

System assumptions can be specified in a number of ways with varying degrees of advisability depending on the model checker, for example we used SMV [Cimatti et al., 2002] and [Cadence Berkeley Laboratories, 2000]: (i) by adding *state invariants*, (ii) by extending the property specification by *temporal assertions*, and (iii) by binding the model execution to the behaviour of *observer automata*. The first two options are described here while observer automata are used in Section 4.5 [Bérard et al., 2001]).

State invariants specify that certain combinations of actions cannot occur at any time or must occur all of the time. For example, two user inputs may be required not to occur at the same time. An invariant can be used to specify that only a single user input can occur at any step. So if $\mathtt{E}_U$ is the set of all user-initiated events in the model and $i, j \in \{1, 2, \ldots, |\mathtt{E}_U|\}$ then

$$\texttt{INVAR}\ \bigwedge_{e_i \in \mathtt{E}_U} (e_i\ \wedge\ \neg \bigvee_{e_j \in \mathtt{E}_U, j \neq i} e_j)$$

Temporal logic assertions are illustrated in the case study by the assumption that the operator consistently forgets to store control elements on the handheld device (see sequence 6 of Fig. 7). This is achieved by specifying an assertion "`alwaysForget`" as follows:

```
assert alwaysForget:
 G !(savePmp1ctrls| [...] |savePmp5ctrls);
```

and then checking the original property `SAN1` under the assumption that this assertion holds[2]:

[2] It can be argued that such an assumption is too restrictive. For a list of patterns that make it possible to specify more sophisticated sequential properties from a system-centred and usability point of view, see [Dwyer et al., 1999] and [Loer, 2003, Chapter 5]. Observer automata for specifying more elaborate user behaviour is demonstrated using the example of this hand-held device in [Loer, 2003, Chapter 7].

```
assume alwaysForget;
using  alwaysForget prove SAN1;
```

4.3 A suggested design alteration

Having explored the forgetfulness issue a design solution is required. It is proposed that an interlock is introduced that warns the user and asks for acknowledgement that the currently displayed control elements are about to disappear. The warning is issued whenever a device position is left and the device's control elements are neither on screen nor stored in a bucket. The extended model in Fig. 11 contains additional paths from PxCTRL_INVIS to PxCTRL_VIS states.

For each PUMPx_CONTROLS sub-state of chart TEMPORARY_ELEMENTS add an intermediate state PxVISWARNING and two transitions t_i from state PxCTRL_VIS to state PxVISWARNING, and t_j from state PxVISWARNING to state PxCTRL_INVIS, where $label(t_j) =$"user_ack". For pump 1 $label(t_i)$ is given by:

```
ex(POS2)[(in(BUCKET1)
          and (B1CONTENT==0 or B1CONTENT==2))
         or in(BUCKET2)
         or (in(BUCKET3) and (B3CONTENT<2))]
```

This design does not prevent the user from acknowledging and then doing nothing about the problem. Checking the same properties, including the assumptions about the forgetful user, produces Sequences 7 and 8 in Fig. 7.

4.4 User tasks as context

The OFAN models have been used without a USER TASKS component so that the model checker can explore all possible interactions between the user and the system. However, in the course of the analysis the analyst might wish to further constrain the exploration to typical uses of a system as defined by task descriptions. Tasks either capture the designer's anticipation of system use, may be the operating procedures that are imposed by the operating company or regulatory authorities, or may arise as a result of careful study by task analysts of the work that a user must carry out.

Analysis can be performed automatically in a style akin to the state exploration approach of [Fields, 2001] where assumptions about user and environment behaviour are specified as invariants or temporal logic models. The addition of user task models focuses the analysis to two perspectives:

1. System behaviour can be investigated assuming a given user behaviour. Hence it is possible to see if the system responds adequately to a given user input and how different versions of the system specification – or different system designs – respond to the same user input.
2. System reactions to particular *deviations* of user behaviour can be examined. For example, it might be appropriate to explore whether the system response to particular user deviations remains within tolerable bounds. A similar technique that makes use of rule-based system descriptions is described in [Fields, 2001, Chapter 4].

The analysis of the system response to a specific user behaviour requires the specification of that behaviour in terms of states and events in a parallel statechart module. This module is designed to be concurrent and to interact with the remaining modules of the specification. The resulting state machines can be viewed as observer automata [Bérard et al., 2001].

With a user task description in place the aim of the analysis that is carried out here is to check (i) *if* the state that marks task completion can be reached establishing the general possibility of completing the task with the system, and (ii) *how* that state can be reached (via exploratory analysis) thereby investigating the conditions under which the task can be completed. A focus of interest in the second step might be, for example, to consider unwanted side effects during the task execution, like additional and potentially hazardous user activities that are not task related.

In a related approach by [Marrenbach and Leuker, 1998] normative user task models are generated from GOMS models. These models are then used as drivers for a simulation. The evaluation is then performed manually on the basis of automatically generated simulation protocols [Melchior, 1987].

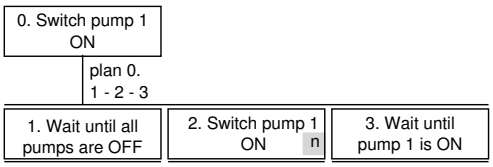
4.5 Task models for checking system adequacy

In this section, a simple task is considered before considering a particular task that achieves the same goal as was discussed in the last section.

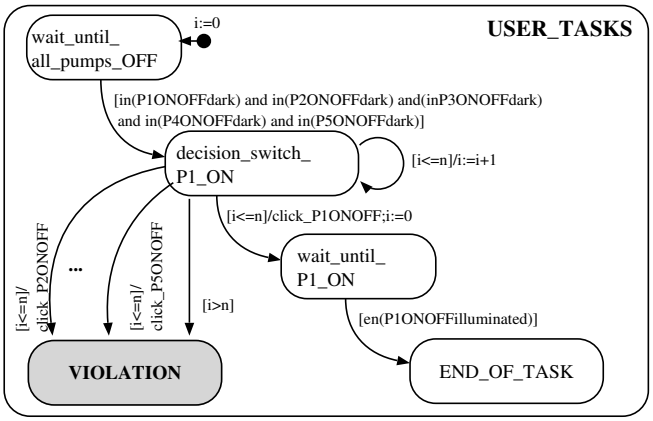
A simple task Fig. 8a shows the goal hierarchy for task:
"*Once all pumps are off, switch pump 1 ON (after at most n steps delay).*"
The corresponding statechart specification in Fig. 8b defines this task as a sequence of interleaved "wait" and "decision" states. In this task model the initial state represents a user waiting for all pumps to be OFF. The waiting period ends and the task state machine proceeds to the next state when the triggering condition of the exiting transition becomes true. This condition specifies that the user senses from the states of the DISPLAYS module that all pumps are off. The second state represents the user's decision to click the ON/OFF icon of pump 1 at some point in the future. If the user decides to click the icon, the machine proceeds to the next step and eventually arrives in the final state. If the user decides to wait, a self-loop is taken and the state machine remains in the current state.

The self-loops can be taken n times before the task must progress. Failure to progress within the time limits (that is, a *delay error* in [Hollnagel, 1991]) or the generation of additional actions (that is, an *intrusion error*) leads to a transition into the `VIOLATION` state. The analysis of the system can then be performed under the assumption that the `VIOLATION` state is never reached.

A less trivial task Fig. 9a shows the specification of the task "*Produce substance C*". This task can be decomposed into a sequence of sub-tasks and actions that can be mapped directly onto events in the OFAN model. The translation of



a) HTA description



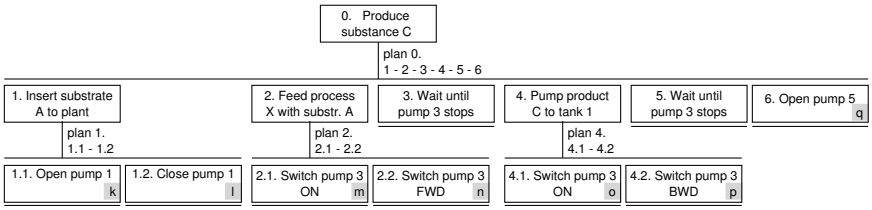
b) Statechart specification

Figure 8. Specification for User Task: "*Once all pumps are off, switch pump 1 ON (after at most n steps)*".

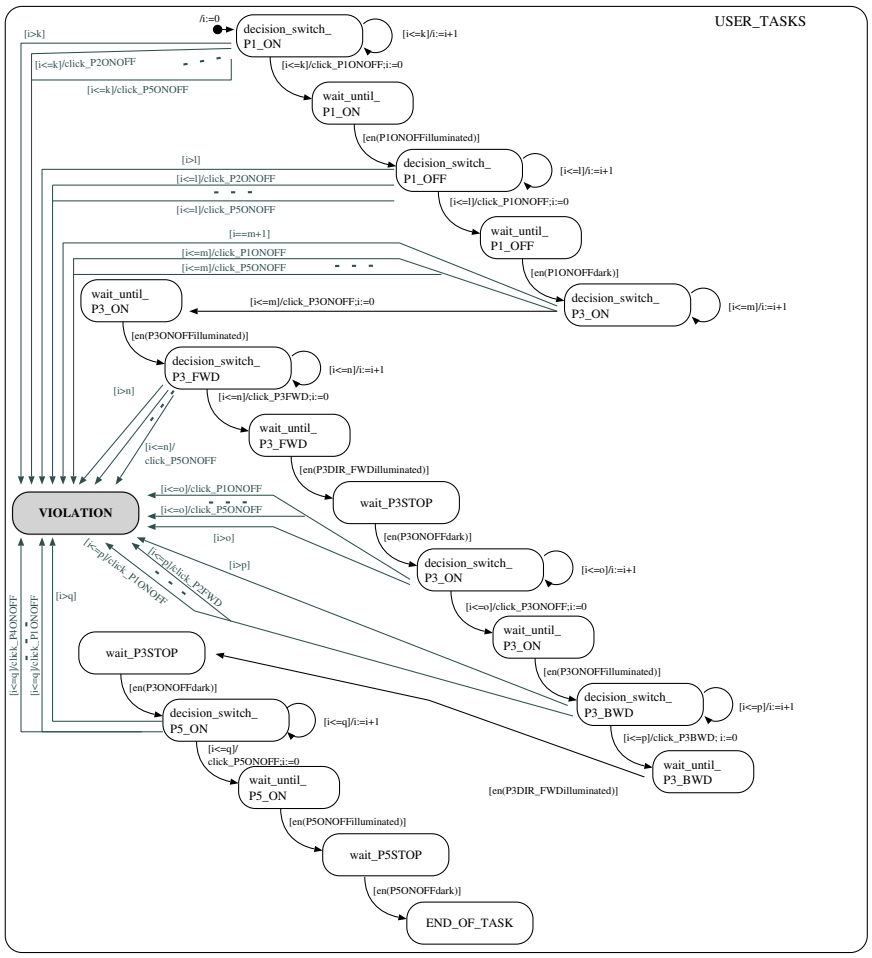
this linear task yields a sequence of interleaving wait and decision states again (Fig. 9b).

The only flexibility allowed by these specifications are the non-deterministic choices in the "decision" states. The task model in Fig. 10 illustrates how a more complex user behaviour can be modelled, including plans containing loops and an explicit erroneous user action. In state `choose_option`, the user can opt to wait before he proceeds with the task. Additionally, the user might decide to repeat the process of adding substrate A to the plant (decision loop in plan 1 of Fig. 10a).

Finally, there is a possibility that the user chooses to perform a non-accepted operation (erroneous input – see task 2a in plan 0 of the task hierarchy). In this case substance A would be released into the environment using pump 5. According to requirement `PSR1` this is supposed to be ruled out by the system logic, so that state "`OOPS`" should not be reachable. The model checker can be used to analyse if this requirement holds. Modelling erroneous user behaviour can be useful, if during the model-checking analysis erroneous behaviours are to be ruled out, except for explicitly tolerated errors.

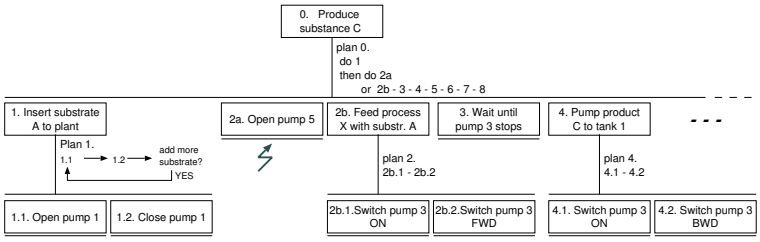


a) Decomposition of task: "*Produce substance C*".

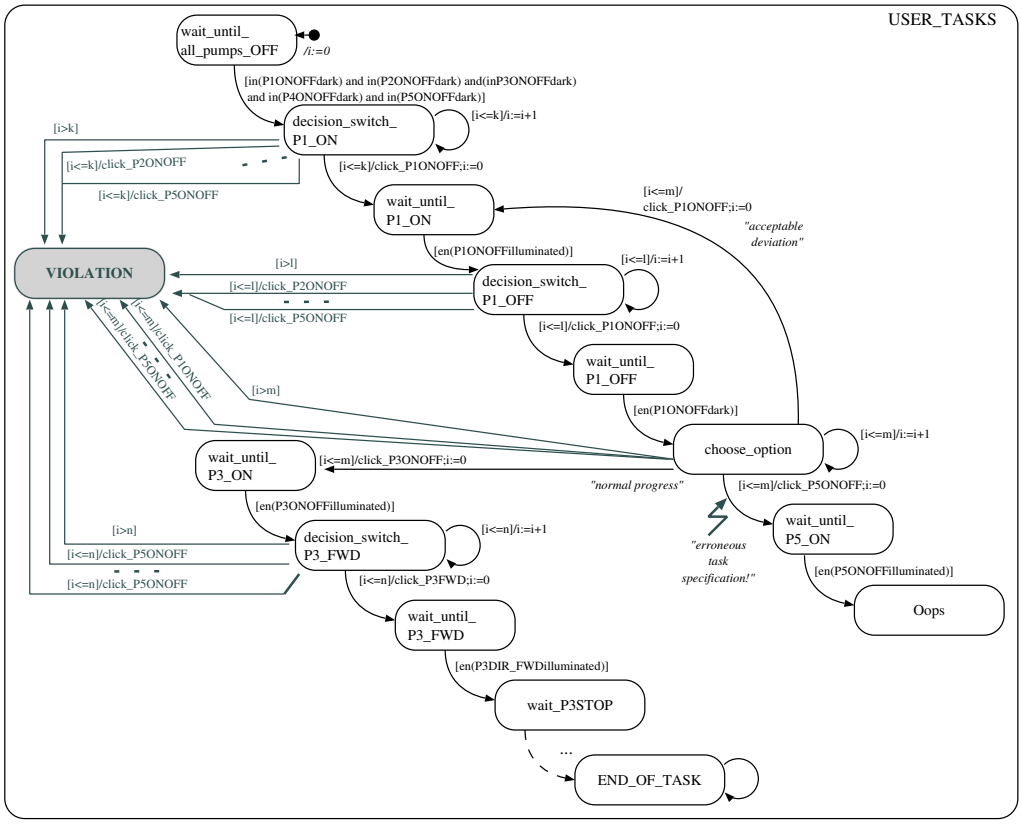


b) User Task specification of the task hierarchy.

Figure 9. User Task specification for task: "*Produce substance C*".



a) Task hierarchy with decision steps.



b) Corresponding statechart specification

Figure 10. Specification of a User Task with decision alternatives.

5 Conclusions

In many stages of the analysis it became clear that the suggested hand-held device can be an aid to the operator in the plant. However, the moded interface with its limited display capabilities requires a number of additional activities in order to access functions. This increases the potential for omission errors that

might arise, particularly as a result of a loss of contextual awareness. Initial analysis revealed the need to modify the design by adding an interlock, and the analysis was repeated for the altered design. During the analysis of the modified design, all property specifications that were created for the original design were re-used.

In conclusion it can be argued that none of these devices on its own provides optimal support for the work activities in such a processing plant. As argued in [Nilsson et al., 2000] computerised "centralised control room" activities and manual "in the plant" activities are inseparable. In a follow-up analysis it would remain to be shown if the combination of control interfaces provides suitable support. In particular, one would need to show that multiple control devices do not interfere or create situational awareness problems, especially if more than one plant operator is involved. However, the analysis of such issues requires the development of more comprehensive OFAN models, which also increases the resource requirements for model checking.
Many of the issues that arose from the analysis could have been found manually. The techniques presented here however offers a thorough exploration of the state space which will be valuable in revealing unforeseen consequences of the design. The paper demonstrates how a number of modes of analysis can be used to explore contextual issues in the design of a handheld control device used to replace the displays and control of a central control room. Simple properties associated with the underlying system can be investigated as a sanity check that the system model has appropriate semantics.

The paper demonstrates a method of capturing key characteristics of the context model using a statechart description where discrete states represent spatial regions. Explorations were achieved by comparing and considering sets of sequences of actions that reach a specified goal state. No assumptions are made about user behaviour initially. The paper then demonstrates how user constraints might be used in order to explore subsets of the traces that can achieve the goals. The intention is that a human factors expert or a domain expert may be provided with sufficiently rich information that it is possible to explore narratives surrounding the traces generated. These experts may also assist the process of adding the appropriate constraints to the properties to be checked.

Traces can form the basis for scenarios that aid exploration of potential problems in the design of mobile devices, e.g. the additional work that would be involved for the system operator if subtasks are inadvertently omitted in achieving the goal. The tool can also be used to find recovery strategies if an operator forgets to store control elements.

While this paper outlined the technical solutions for restricting the analysis, it will be necessary to devise strategies for appropriate guidance with respect to (i) finding an efficient sequence of analysis steps and (ii) devising a strategy for the introduction of appropriate assumptions on "component" behaviour. In this context proof planning strategies will be required that avoid unnecessary future analysis steps based on current analysis results. Finally, the size of the models that can be analysed is limited. Suitable techniques and heuristics for semantic abstraction of system models need to be devised to avoid the state explosion

problem. However, the size of models that can be dealt with is encouraging and this situation can be improved through appropriate abstraction and consistency checking.

References

[Abowd and Mynatt, 2000] Abowd, G. and Mynatt, E. (2000). Charting past, present and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58.

[Bérard et al., 2001] Bérard, M., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., and Schnoebelen, P. (2001). *Systems and Software Verification. Model-Checking Techniques and Tools.* Springer.

[Cadence Berkeley Laboratories, 2000] Cadence Berkeley Laboratories (2000). Cadence SMV Homepage. `http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/`.

[Cimatti et al., 2002] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., and Tacchella, A. (2002). NuSMV 2: An Open Source Tool for Symbolic Model Checking. In Larsen, K. G. and Brinksma, E., editors, *Computer-Aided Verification (CAV '02)*, volume 2404 of *Lecture Notes in Computer Science*. Springer-Verlag.

[Clarke et al., 1999] Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT Press.

[Degani, 1996] Degani, A. (1996). *Modeling Human-Machine Systems: On Modes, Error, and Patterns of Interaction*. PhD thesis, Georgia Institute of Technology.

[Dix et al., 2000] Dix, A., Rodden, T., Davies, N., Trevor, J., Friday, A., and Palfreyman, K. (2000). Exploiting space and location as a design framework for interactive mobile systems. *ACM Transactions on Computer-Human Interaction*, 7(3):285–321.

[Dwyer et al., 1999] Dwyer, M., Avrunin, G., and Corbett, J. (1999). Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering, Los Angeles, California*.

[Fields, 2001] Fields, R. (2001). *Analysis of erroneous actions in the design of critical systems*. PhD thesis, Department of Computer Science, University of York, Heslington, York, YO10 5DD.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274.

[Hollnagel, 1991] Hollnagel, E. (1991). The Phenotype of Erroneous Actions: Implications for HCI Design. In Weir, G. and Alty, J., editors, *Human-Computer Interaction in Complex Systems*, pages 1–32. Academic Press.

[Holzmann, 2003] Holzmann, G. (2003). *SPIN Model Checker, The: Primer and Reference Manual*. Addison Wesley.

[Horrocks, 1999] Horrocks, I. (1999). *Constructing the User Interfaces with StateCharts*. Addison Wesley.

[Huth and Ryan, 2000] Huth, M. R. A. and Ryan, M. D. (2000). *Modelling and reasoning about systems*. Cambridge University Press.

[Leveson, 1995] Leveson, N. G. (1995). *Safeware: System Safety and Computers*. Addison-Wesley Publishing Company Inc.

[Loer, 2003] Loer, K. (2003). *Model-based Automated Analysis for Dependable Interactive Systems*. PhD thesis, Department of Computer Science, University of York, UK. pending.

[Marrenbach and Leuker, 1998] Marrenbach, J. and Leuker, S. (1998). Konzept zur evaluierung technischer systeme in der entwicklungsphase. *ITG Fachbericht: Technik für den Menschen*, 154:103–113.

[Melchior, 1987] Melchior, E.-M. (1987). Protokollanalyse als methode zur erfassung des wissensstandes während des lernprozesses beim erlernen der bedienung komplexer geräte. In *Trainingsverfahren und Lernverhalten*, pages 72–81. Deutsche Gesellschaft für Luft- und Raumfahrt e.V.

[Nilsson et al., 2000] Nilsson, J., Sokoler, T., Binder, T., and Wetcke, N. (2000). Beyond the control room: mobile devices for spatially distributed interaction on industrial process plants. In Thomas, P. and Gellersen, H.-W., editors, *Handheld and Ubiquitous Computing, HUC'2000*, number 1927 in Lecture Notes in Computer Science, pages 30–45. Springer.

[OFAN model for the hand-held device]

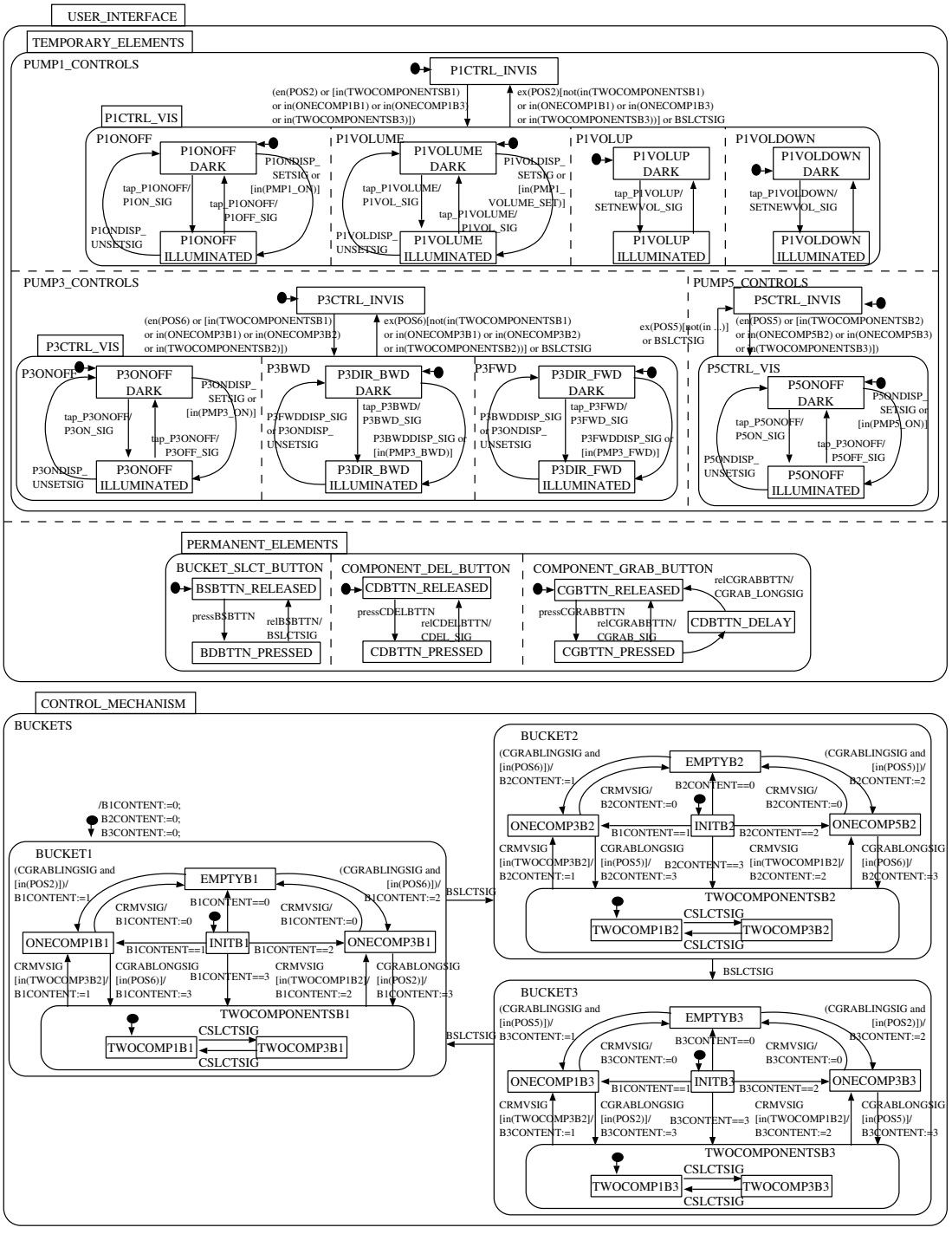


Figure 11. OFAN model for the hand-held device: The USER INTERFACE and CONTROL MECHANISM modules.