Model-Based Formal Analysis of Temporal Aspects in Human-Computer Interaction

Karsten Loer and Michael Harrison
Department of Computer Science, University of York
York, YO10 5DD, UK
{Karsten.Loer, Michael.Harrison}@cs.york.ac.uk

Abstract

This position paper describes how work representations might be generated using a process based on model checking. Sequences or traces are explored that either satisfy properties or demonstrate the failure of a property. This process may be seen as a contribution to an understanding of constraint based instances of work. Further, time based constraints may be explored using timed automata and model checking systems such as UPPAAL *and* HYTECH.

1 Introduction

The way work is carried out may be critically affected by how the technology (individual elements of which are called devices in this paper) in any system is designed. Hence any task analysis of an existing system or task design for a new system may miss unforeseen consequences that arise because the effects of the device and its interaction with the environment were not envisaged. This issue has been widely discussed. Amongst critiques of normative prescriptive descriptions, cognitive work analysis [Vicente, 1999] argues that work can profitably and appropriately be represented in terms of constraints. Constraints may be generated as a result of characteristics of the environment or because of specific aspects of the device. Both have the effect of narrowing the possible behaviours that the system can exhibit. Hence instead of focussing on specific sequences of actions that might be generated by the task description it is possible to consider classes of behaviour that satisfy the constraints.

The practicality of describing work sufficiently richly through constraints raises a number of concerns. Whilst system or software engineers have representations and models that make it relatively straightforward to deal with "constraint like" abstractions, human factors analysis *typically* requires more concrete instances of activity in context in order to assess the effect that the available resources have on user actions for example. The question that this poses is whether it is possible to provide system models that capture the key constraints and then use these models to produce instances of behaviour that may be of value to users. The paper describes an approach to this that has been of value in exploring aspects of the design of safety critical systems. This approach involves a precise representation of characteristics and the device, its environment and potentially its use. The paper describes how this has been used to model task and location in the context of a ubiquitous system. Once modelled the system can be converted into a finite state machine and used as a basis for checking that the model satisfies specific properties that might relate to users in the context of these constraints.

The paper describes how this approach might be used when analysing timing issues in interactive systems. Hence rather than focus on how timing issues might be added to some kind of task representation the concern is how these constraint style models might be augmented to include time. Also of concern is how sequences might be extracted from checking these models that lead to interesting sequences that satisfy or break timing properties.

2 Modelling contextual aspects of interactive systems

In his OFAN[1] modelling technique Degani uses the statechart notation [Harel, 1987] for the specification of interactive systems [Degani, 1996]. An OFAN model describes a system from the perspective of a single device within the system by means of pre-defined categories:

control elements:	description of the input controls.
control mechanism:	model of the device functionality.

[1] *Ofan*: Hebrew for a set of perpetuating wheels. Degani wants to emphasise that the (super-)states describing the concurrent components of his model interact like a set of cog-wheels.

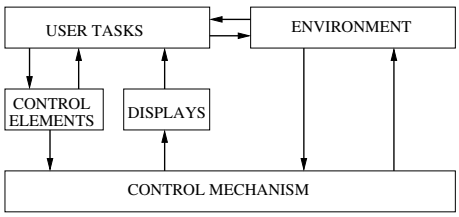


Figure 1. Architecture and information flows of OFAN models.

displays:	description of the output elements (could be multi-modal, for instance audio-visual and haptic).
environment:	model of relevant environmental properties.
user tasks:	sequence of user actions that are required to accomplish a certain task (potentially for multiple users).

These categories are specified as orthogonal sub-states of a statechart. The structure of the models and the information flows between the components is illustrated by Figure 1.

OFAN models capture behavioural properties of the interactive device, its operating context and (optionally) the user. In addition to the standard device-centred issues like reachability, robustness and visibility, the model can also be used to analyse contextual issues of system use.

The control elements and display have much in common with the interactor style models of Duke and Harrison [Duke and Harrison, 1993] and Faconti and Paternò [Faconti and Paternò, 1990]. The innovation here is also to capture the environment and the task.

In OFAN statechart models, temporal progression is measured in terms of model execution steps. These steps are not real-time but they rather describe logical changes based on value changes of discrete variables.

For the analysis of real-time aspects of models, it is necessary to record continuous time. OFAN models are not restricted to statecharts, but could equally well be specified as timed automata (as indicated in Section 4.3).

2.1 Example: The Pucketizer

A fictitious handheld control device (Figure 2) shall serve as a sample application. The device has a substantial resemblance to the "Pucketizer" [Nilsson et al., 2000], a hand-held device that is supposed to provide an operator with monitoring information during his rounds in a

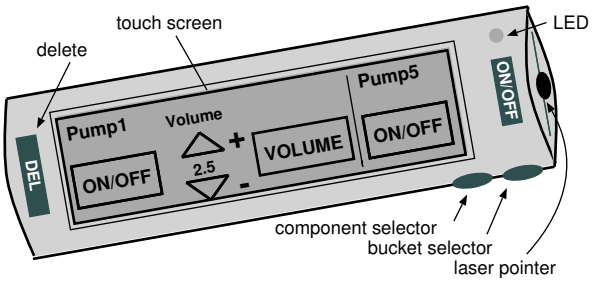


Figure 2. A hand-held control device (modified version of the "Pucketizer" device in [Nilsson et al., 2000]).

water processing plant. The control device implements a "bucket"-metaphor: a number of virtual buckets can be "filled" with status information relating to devices (i.e. different types of pumps, valves and displays) that are passed by a plant operator on his rounds within the plant. By pointing the laser pointer at a device of interest and pressing the component selector button, the status information for that device and a rendering of its control elements are transferred into the currently selected bucket. Devices 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 which can be "attached" to devices in the plant in order to remind the operator or inform colleagues of issues related to that device. For the purpose of this paper the intended use of the device has been altered from monitoring and annotating to monitoring and manipulation. Therefore, the audio processing facility is out of scope, and the display shall be replaced by a touch screen. Control elements for the manipulation of devices in a bucket shall be displayed on this screen, with a display area that is limited to controls for up to two pumps at a time.

Model of the Device and context

The OFAN model of the hand-held device has many features of interest. The device has physical buttons which are accessible continuously. Other control elements, like pump controls, are given by icons that are available temporarily. Their appearance depends on the position of the device and the user actions. When the operator approaches a pump, its controls are automatically displayed on the screen. The operator can then decide to transfer the device into a bucket for future remote access with the component selector button. Controls for devices in other locations can be accessed remotely if they were previously visited and stored in a bucket. Once a component is available in a bucket and that bucket has been selected, the hand-held device can trans-

mit commands to the process plant, using the pump control icons.

Figure 5 (see Appendix) shows an extract of the OFAN model for the hand-held device. The BUCKETS state makes use of variables "COMPHIST*b*" that encode the content history of each bucket *b* as follows:

value of COMPHIST2	content of bucket 2
0	–
1	P1
2	P3
3	P1 & P3

value of COMPHIST3	content of bucket 3
0	–
1	P5

The OFAN model is completed by an ENVIRONMENT module, which shall be given by the device position model in Figure 5.

In the example system the user can choose between three buckets of limited capacities and contents: the first bucket is empty, the second bucket can contain up to two devices (which are pump 1 and pump 3), the third bucket can contain no element or pump 5.

In this representation the environment can be described as a state transition diagram, each state represents a position within the plant. The model describes limited properties of the space, so for example it is specified that it is not possible for the operator to go from POS1 to POS5 without going through other areas. As the operator moves through the space the transitions that are triggered cause transitions in the device model. In this case the user interface component of the OFAN chart consists of concurrent charts that involve alternative states in which different parts of the interface are revealed or hidden. The environment model then uses a standard approach in a hybrid system of reducing the continuous model into a finite set of zones. With this model it is possible to ask questions about the visibility of aspects of the interface in the face of different situations - this is discussed in more detail later in the paper.

There may be a number of reasons why an appliance does not reflect the state of the system appropriately. The network may have failed or because of the characteristics of the topology of the environment it may be the case that the appliance is in some kind of communication shadow. There are two reasons why topological issues may be problematic. The first is that environment transitions take place but the device model is not updated equivalently. There is a dissonance between the states of the device and the states of the environment. The second possibility is that, given a richer type of state where variables are associated with states, it may be the case that actions depend on values of the state that have actually been updated. This can be expressed as "the action has a false belief about the state".

3 Using model-checkers for exploration

Model checking [Clarke et al., 1999] is a technique for the exhaustive exploration of system models in order to establish if a specified requirement holds. In its application for safety-critical interactive systems the process is:

1. to model the device and its environment,
2. assert properties that capture features of concern related to the constraints, and then
3. check the system in relation to these constraints.

The interesting features of the system are captured amongst the situations where the OFAN model fails to satisfy the specified constraints. When appropriately configured and the property fails counterexamples are generated by the model checker showing a sequence of states that captures the circumstances where the model fails. Hence, when appropriately carried out, uses of the system that have interesting properties can be expressed.

For OFAN-like system models, model checkers can be used to perform three kinds of analysis, focussing on:

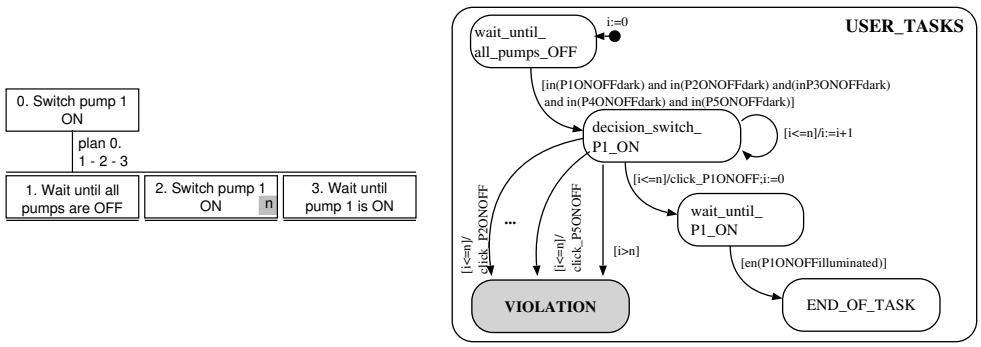
1. The interaction of a user with a device for a given task.
2. The response of the system to possible user inputs.
3. The influence of specific task variations on the overall system state.

Constraints are innate in the OFAN specification but can also be expressed as properties of the model. For example, a property might express that a mode change is always associated with a display action that notifies of the change. It may be appropriate to analyse sequences that involves paths where this fails to be true, or it may be where specific types of sequence fail to be completed within a certain time limit. In the following sections these kinds of analysis are demonstrated.

4 Model exploration with focus on task models

This kind of process can also be carried out with a relatively strong notion of task. A task description may be embedded within the OFAN model.

From the task hierarchy in Figure 3a (annotation *n* denotes time limit [number of execution steps]), a statechart model can be produced. The model in Figure 3b is very restrictive. It represents an automaton that accepts exactly the task sequence presented in the HTA. Any additional or delayed action leads to a task violation – "*catch-all*" error state VIOLATION. Reachability of target state is checked



a) HTA description b) Statechart specification

Figure 3. Specification for USER TASK**: "***Once all pumps are off, switch pump 1 ON (after at most n steps)***".**

by a model-checker; now under the assumption that only acceptable user inputs are made.

Events/conditions/actions in the statechart can represent atomic user inputs (as in this example), they can also represent higher-level tasks, depending on the level of abstraction of the model.

4.1 Analysis of task sequencing

By adding normative models of tasks and plans to the OFAN model, it is possible to investigate the system behaviour in response to a particular task.

In [Loer, 2003, chapter3] it is shown how a number of commonly used plans (called "fixed sequence", "optional tasks", "waiting for events", "cycles", "time sharing", and "discretionary" in [Dix et al., 1998, p.267]) can be translated into state machines.

These translations were used to produce the normative task model in Figure 3b. For demonstration purpose, only two basic violations are considered in this task model: (i) performing a wrong action and (ii) performing an action late. These simple phenotypes are called "intrusion" and "delay" in the taxonomy of [Hollnagel, 1991]. Intrusion errors occur if a wrong action is performed in some sub-task. The action part of a statechart transition label cannot contain disjunctions. Therefore, each erroneous action needs to be modelled separately by a transition from the state that represents the current sub-task to the violation state, see Figure 3b (for space reasons only two intrusion errors per subtask are shown). A delay error occurs if a prescribed action is not performed within a given time limit[2]. In the task hierarchy for this example an explicit time limit `n`, denoted by a shaded box in Figure 3a, is placed on sub-task 2 ("Switch pump1 ON", action "`click_P1ONOFF`"). Such a time limit is modelled by a self-loop counter `i` that records how long a user resided in a sub-task, see second state in Figure 3b. If the model remains in a state until the counter exceeds the time limit, a transition to the `VIOLATION` state is taken.

The addition of normative user task models focuses the analysis of OFAN models to two perspectives:

1. The system behaviour can be investigated under a given user behaviour. This analysis makes it possible to see if the system responds adequately to a given user input. In a further step it can also be investigated how different versions of the system specification, or different system designs, respond to the same user input.

2. The system reactions to particular *deviations* of user behaviour can be examined. For example, it can be explored if 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].

4.2 Analysis of constrained "task space"

In the model in Figure 3 the progression of task is described normatively. Task violations are irrecoverable, because the `VIOLATION` state has no exiting transitions. More permissive formulations are conceivable, for example, by introducing different kinds of violating states with exiting transitions that permit error recovery.

The specification of task models can be omitted altogether, so the model checker will explore all possible user inputs. Insights on task characteristics can then be inferred from model-checking traces. Within this space of possible behaviours are a number of input behaviours, one would consider erratic, breaking work constraints.

A "constrained user behaviour" can be achieved by introducing "whenever..then"-rules that describe limited user responses to certain conditions. Constraining rules can be implemented by cyclic state machines. For example, the

[2] Note, that in the models in this section the notion of time is qualitative in terms of execution steps of the model as opposed to real-time.

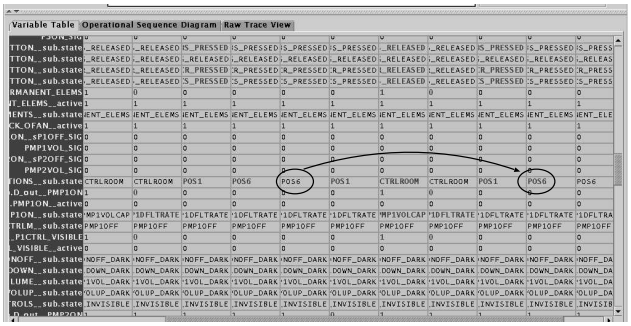
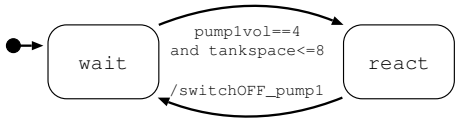


Figure 4. Demo trace: If device is forgotten to be stored in bucket user needs to return to device position.

constraint "*whenever the user realises that pump 1 is operating full volume while its target tank it close to full, the user will switch off the pump*" is decribed by the following state machine:



These rules have to be justifiable/justified, of course.

The analysis of model-checking traces can point the analyst to potential problems.

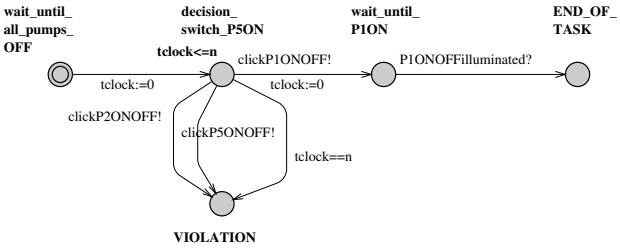
For example, the demo trace in Figure 4 was generated during the analysis of the property: "*whenever the controls for pump 3 are visible on the pucketizer, operating these controls will result in a control signal being sent to the pump*". The trace presents a situation where the user keeps returning 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.

The models introduced so far allow investigation of task logic and task sequence, but there is no notion of real-time.

4.3 Analysis of real-time aspects of tasks

There are two types of model-checkers that take real-time into account. They use timed automata or hybrid models

Tools like UPPAAL [Amnell et al., 2001] make it possible to add continuous time to models. For example, the timed automaton below enhances the task model in Figure 3b by temporal invariants and clock conditions:



The UPPAAL model-checker can be used to analyse whether a task can be completed in n time units. Furthermore, by iteration over n the minimum/maximum duration can be determined that is required to perform a task.

The analysis of real-time models is complex and existing model-checking tools prohibit use of large models (state explosion) and therefore there are practical limits to the use of these tools.

A third class of model-checkers combine discrete and continuous variables within the same model. Discrete behaviour is defined in terms of transitions between locations (similar to Figure 3b). Within locations the behaviour of continuous variables (for example, temperature or volume) over time is defined in terms of differential equations.

An advantage of model checkers for hybrid systems, for example HYTECH [Henzinger et al., 1997] over model checkers for timed automata is that they can be applied to perform parametric search. For instance to analyse questions of the style: "What is the latest deadline by which an action needs to be taken in order to be able to complete a task in time?"

The main problem with timed model checkers is that because many more states are generated to deal with continuous time, much smaller models are possible than, for instance, with the SMV model checker [McMillan, 1993] that was used for the analysis in Section 4.1.

5 Conclusions

Model-checking and the formal modelling of devices in context can play a role in understanding the effects of tasks and the unforeseen consequences of various types of task violation. The paper has briefly outlined the role of OFAN models in the context of a handheld device used to control a sewage plant. The paper describes how these descriptions can be translated into models to form the basis for a model checking process in which property failures may be used to identify interesting sequences. These sequences can be particularly interesting from a timing point of view when the model translation is to timed automata.

6 Acknowledgements

This work was funded by the EPSRC *DIRC* project.

References

[Amnell et al., 2001] Amnell, T., Behrmann, G., Bengtsson, J., D'Argenio, P., David, A., Fehnker, A., Hune, T., Jeannet, B., Larsen, K., Möller, M., Pettersson, P., Weise, C., and Yi, W. (2001). UPPAAL - Now, Next, and Future. In Cassez, F., Jard, C., Rozoy, B., and Ryan, M., editors, *Modelling and Verification of Parallel Processes*, number 2067 in Lecture Notes in Computer Science Tutorial, pages 100–125. Springer–Verlag.

[Clarke et al., 1999] Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. The 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., 1998] Dix, A., Finlay, J., Abowd, G., and Beale, R. (1998). *Human Computer Interaction*. Prentice Hall Europe, 2nd edition.

[Duke and Harrison, 1993] Duke, D. J. and Harrison, M. D. (1993). Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36.

[Faconti and Paternò, 1990] Faconti, G. and Paternò, F. D. (1990). An approach to the formal presentation of the components of an interaction. In Vandoni, C. and Duce, D., editors, *Eurographics '90*, pages 481–494, Montreaux. North-Holland.

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

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

[Henzinger et al., 1997] Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997). HyTech: A Model Checker for Hybrid Systems. *Software Tools for Technology Transfer*, 1:110–122.

[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.

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

[McMillan, 1993] McMillan, K. L. (1993). *Symbolic model checking*. Kluwer.

[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, *Proceedings of the Second International Symposium on Handheld and Ubiquitous Computing,HUC2000*, number 1927 in Lecture Notes in Computer Science, pages 30–45. Springer Verlag.

[Vicente, 1999] Vicente, K. (1999). *Cognitive Work Analysis: Toward Safe, Productive, and Healthy Computer-Based Work*. Lawrence Erlbaum Associates.

A OFAN model of Pucketizer

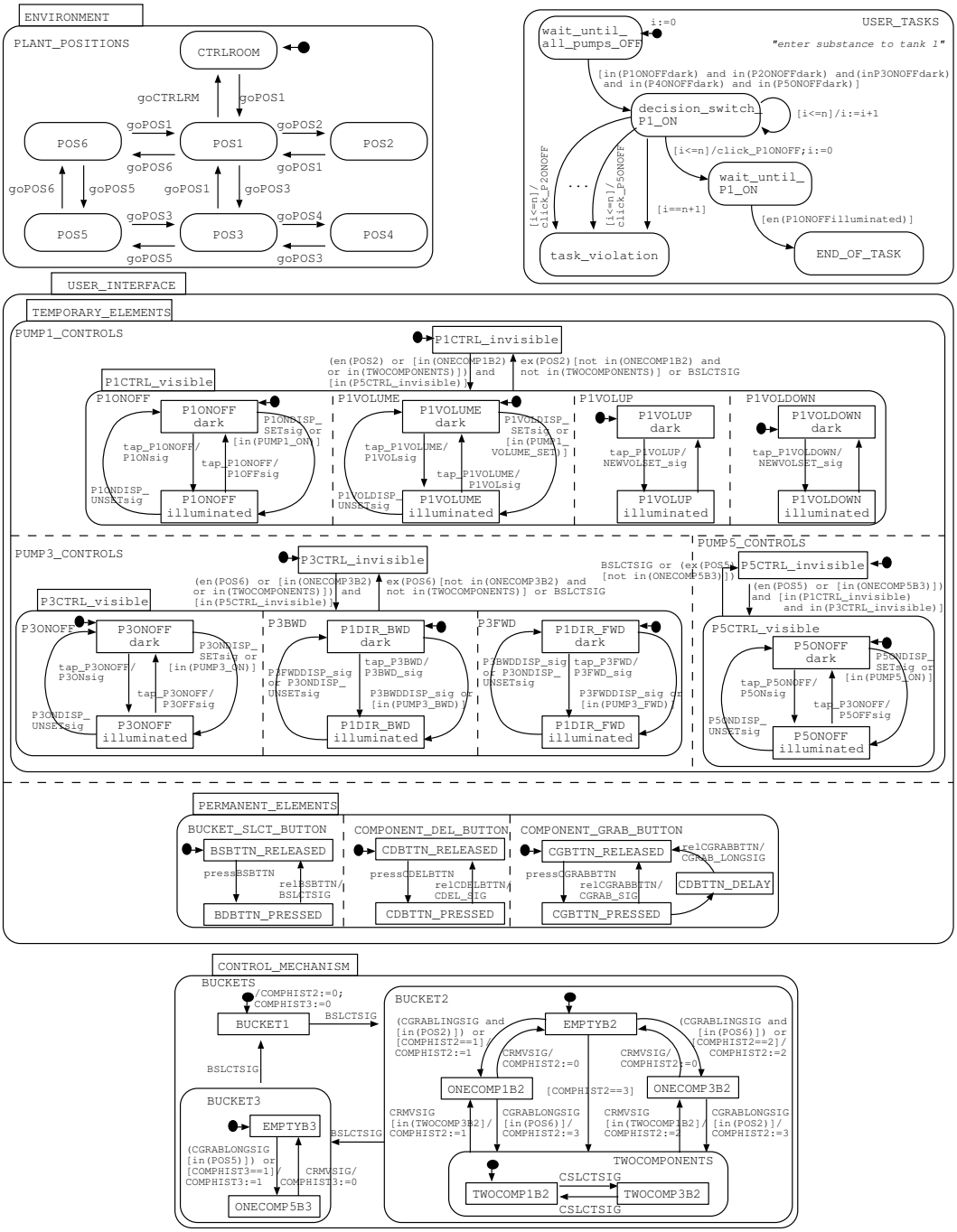


Figure 5. OFAN **model for the hand-held device: The** USER INTERFACE **and** CONTROL MECHANISM **modules, as well as (part of the)** ENVIRONMENT **module, and a** USER TASK **module.**