ANALYSING DYNAMIC FUNCTION SCHEDULING DECISIONS

Karsten Loer, Michael Hildebrandt and Michael Harrison
Interdisciplinary Research Collaboration in Dependability (DIRC)[1]*,*
Department of Computer Science, University of York, York YO10 5DD, UK

Abstract: Function allocation, as a process used in the construction of dependable complex systems, is a significant aspect of the design and implementation of interactive systems. It involves a documented and rational process for deciding what aspects of the system should be controlled by which human roles in the system and how the system should be automated to support these roles effectively. As computer systems have become more advanced, and the control of systems more complex, the notion of dynamic function allocation becomes increasingly desirable where in certain situations the automation may take over or give back function to the human user. In this paper we explore a further variant of dynamic function allocation that reflects typical work activity where the dynamic scheduling of activities takes place on the time dimension. The paper discusses this approach to dynamic function allocation called dynamic function scheduling and discusses the role that timed model checking may play in helping identify dependable dynamic function scheduling solutions.

Key words: Dynamic function scheduling; timed model checking.

1. INTRODUCTION

Complex work systems typically involve teams of people co-operating and using technology to achieve work goals. These goals are usually achieved under time constraint. In order to achieve them in a timely and reliable manner, the implementation of the *functions* that achieve the goals may vary according to situation. How functions are most reliably implemented in different situations is a vital and somewhat under-represented aspect of building a dependable system. This topic is dealt with

[1] The DIRC project (see http://www.dirc.org.uk) is funded by the UK EPSRC, Grant N13999.

in research into dynamic function allocation – see (Hancock and Scallen, 1998) and (Scerbo, 1996) for an overview. The overall focus of this work is about how automation can be used adaptively, according to the current demands on the system, and the capabilities and workload levels of the agents involved, in order to offer optimal support to the human operator.

The problem of function allocation is to take a set of functions that describe the work that the system is to do, in the *contexts* in which the work is to be carried out, and to decide how these functions should be implemented by the roles that are defined within the system. Methods are required that will enable system engineers both to take task descriptions and consider how the actions within the tasks should be implemented, and to take specific dynamic function allocation designs and analyse their implications. Typically the methods that exist are concerned with static allocations, that is, the decision about how roles are allocated to function occur at design time, see for example (IJHCS, 2000). In practice, it makes sense to consider the appropriateness of different configurations in different situations under different conditions of workload and different requirements of criteria such as situation awareness. Hence an in-car navigation system may have a different level of automation in which certain default inputs are presumed when the car is moving or in gear than when the car is stationary and in neutral.

In addition to sharing and trading functions among humans and automation, it may be possible to change the way functions are allocated *in time* in order to meet the required deadlines. Given that many modern work situations are rapidly evolving or highly scheduled, it is surprising how few human factors studies have attempted to make a conceptual or empirical contribution to understanding the temporal organisation of work – however, see for instance (DeKeyser, 1995), (Svenson and Maule, 1993) or (Hollnagel, 2000) for exceptions. Of particular relevance for designing function scheduling processes is a better understanding of temporal awareness (Grosjean and Terrier, 1999) and of the use of time as information (Michon, 1990), (Block, 1990). The authors are aware of little work that has been published on analytic approaches to function allocation, such as the analysis of a hydraulics system by (Doherty *et al.*, 2001) using the HyTech hybrid checker (Henzinger *et al.*, 1997).

There are a number of properties of temporal decision processes that are important to be understood if dynamic function scheduling is to enhance the dependability of systems. These include: (i) task arrival rates, (ii) predictability of task arrival, (iii) the agents' awareness of task arrivals and event durations (and situation awareness in general), (iv) the agents' control mode (event-driven or anticipative; scrambled, opportunistic, tactical or strategic), (v) the uncertainty about future system states, monitoring and control lags, (vi) the pre-emptability of tasks, (vii) the deadlines of tasks relative to each other, (viii) a task's contribution to the system's objectives (*value*), (ix) the current priorities among system objectives, (x) the available resources and their service rates, (xi) the compatibility of concurrent tasks, (xii) the feasibility of combining, interleaving, postponing or dropping tasks, and (xiii) the discretion for satisficing and trading-off among system objectives.

This paper shall focus on a subset of these issues in the context of a particular system. The aim is to assess the role that timed model checking can play in helping to understand the trade-offs associated with decisions and thereby illustrate how the design of dynamic function allocation in general, and dynamic function scheduling in particular, can be aided by such checking. The paper is concerned with analysis techniques to support further exploration of dynamic function scheduling.

In Section 2 a case study based on a paintshop (Hildebrandt and Harrison, 2003) is introduced that illustrates a simple situation in which decision to delay or interrupt a function can be of value. Although it is relatively uncomplicated, this system raises important issues about the appropriate use of analysis techniques and problems associated with scaling these techniques. In Section 3 the `uppaal` (Larsen *et al.*, 1997) model of the paintshop system is described, and this is used as the basis of the analysis in Section 4. The `uppaal` hybrid model checker is capable of finding traces or counter-examples where constraints are broken. In a work design process, these traces can be used to generate scenarios where the timing constraints are violated. These scenarios form the basis for developing more appropriate scheduling and resource allocation mechanisms. The paper describes the model, the constraints that were used, and discusses the results of checking a variety of safety properties. The paper concludes with a discussion. Conclusions are drawn about how these techniques might be used more systematically, and objectives for future work are discussed.

2. CASE STUDY

The purpose of the example is that the following features of the design may be considered.

1. How resources can be allocated flexibly among multiple functions.
2. How functions can be allocated to agents along the system's time-line.
3. The action sequence of operators and what overall strategies for the implementation of a given function may be available. For instance, decisions may have to be made regarding the postponement, interleaving, synchronisation, speeding up or slowing down of function servicing, or

regarding manual or automatic control. It may be appropriate to attach a notion of "value" to functions to describe the relative importance of a function and to allow the creation of priority structures among concurrent functions. Temporal properties of functions and agents are parameters in the decision process as well as variables that can be manipulated, i.e., temporal decisions are both based on and about time.

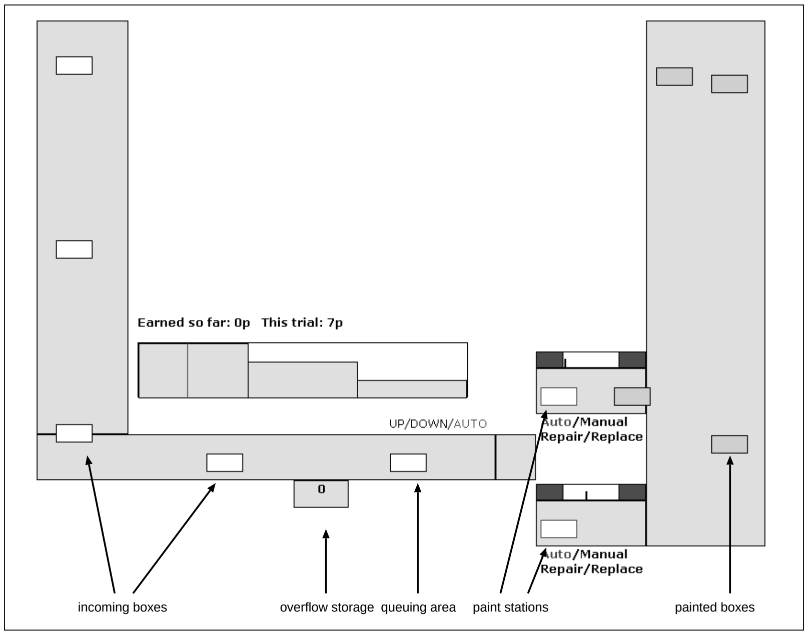

Figure 1. Sketch of the paintshop (Hildebrandt and Harrison, 2003).

Paintshop involves a conveyer belt that transports boxes to two parallel paint stations (Figure 1). Items to be painted enter the system at varying frequencies. A monetary reward is earned depending on the number of boxes painted, the number of boxes spoiled and any repair costs incurred. This system is also designed as a micro-world experiment and actual user strategies have been explored using experiment rather than model checking (Hildebrandt and Harrison, 2003). Boxes arrive at a distribution lift that allocates items to one of the stations. This process can be done automatically whereby the system allocates the box to an empty station. It can also be achieved by the operator overriding the decision of the distribution algorithm by using the 'up' and 'down' buttons forcing the lift in the specified direction. Once the designated production line becomes available, the box is moved onto the paint station and the lift returns to the default position. The paint station can be set to automatic mode (which is the default) or manual mode. In automatic mode, the paint station will automatically specify the number of coats to be painted. The paint cycle for each coat of paint consists of a spraying period and a drying period. With each paint coat, the box whose initial colour is white will become darker. The rate of paint flowing through the nozzles is displayed just above each production line. The flow rate may decrease if nozzles become blocked or increase if the nozzle leaks. The paint process can also be performed manually. To paint an item, the operator has to click on a box and keep the mouse button pressed for a specified period of time. After this period the item will assume the new shade. If the mouse button is released before the minimum paint time the box is not painted and a spoiled box is released. In the model described in the next section, painting takes five time units in the automatic case and two time units in the manual case. When a nozzle ceases to function properly it can be repaired or replaced. Replacing a nozzle incurs no time cost but does incur a certain monetary cost. Repairing the nozzle incurs no monetary cost but causes a delay before the nozzle can be used again. In both the micro-world experiments and the model the cost and time variables were manipulated. Depending on the rate at which boxes arrive at the station, the state of the nozzles and the strategy used to employ the paint stations a certain proportion of the possible boxes will be painted. Boxes can fail to be painted and therefore rejected either because the appropriate procedure has not been carried out inside the paint station or because the queue of boxes waiting to be painted exceeds a certain number.

3. THE MODEL

The `uppaal` tool (Larsen *et al.*, 1997) was chosen to perform the modelling and analysis, as it permits the analysis of networks of linear hybrid automata with clocks whose rates may vary within a certain interval, is readily available and easy to use. The makes it possible to take different temporal reference systems into account, for example, the real-world frequency of items on the belt and the operator's perception of the frequency under varying workload. Automata may communicate either by means of integer variables (which are global) or by using binary communication channels. Communication occurs as a result of two process synchronisations using receiving actions `a?` and sending actions `a!`. Guards are used to describe the circumstances in which communications can take place. Automata may be guarded by conditions involving clocks that can be used to represent delays or time invariants. It is not within the scope of this paper to describe the syntax and semantics of `uppaal` in detail, however the examples given below should be sufficiently clear to give the spirit of the approach. `Uppaal` provides tools for the simulation of systems – the state transition diagrams are animated, and the inter-process communication is displayed as an animated message sequence chart. The tool also supports analysis by state exploration. Thus it is possible to express and check for reachability properties such as:

1. "Is it possible to reach a state where the clock *x* is greater than 20?"
2. "Is it possible to reach a state where all boxes have been painted?"

It is beyond the scope of the paper to describe the details of the verifier – it suffices to describe both the properties that have been checked and those that could be checked.

The model consists of seven concurrent processes. The physical characteristics of the system are modelled as follows:

1. A *dispatcher* automaton dispatches objects to the incoming queue with a frequency that is determined by the workload – frequency is manipulated in the micro-world experiments. In the model that is illustrated in Figure 2a constantly high workload is assumed. This is encoded in terms of frequency, i.e. a new box arrives on the belt every two units (i.e. `workload=2`, values representing a medium and low workload are `3` and `4`, see Section 4.5). In order to reduce the complexity of the analysis, the number of boxes in the model is limited to 10. While it is acknowledged that this is a great simplification in comparison to the real-world continuous flows, this small model is sufficient for the purposes of this paper.

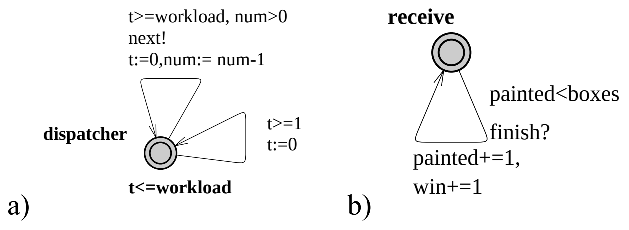

Figure 2. The (a) incoming and (b) receiving conveyor belts. (Key: `t`: clock, `num`: number of boxes yet to be dispatched, `workload`: encoding of workload as dispatch frequency, `painted`: number of finished boxes, `win`: win).

2. The *paint station* automaton (see Figure 3) – of which there are two instances (`station1` and `station2`) – models automatic and manual operation (top and bottom part of the automaton), fault occurrence and repair and replace costs. The severity of faults increases over time. A nozzle may break as soon as two items are painted but it will break for

sure once four items are painted. Repairs cost 24 time units (see locations repairingA and repairingM). For a particular user, replacing a nozzle costs four tokens (see user automaton discussed below – note that such costs can vary, for instance, depending on a user's skills).

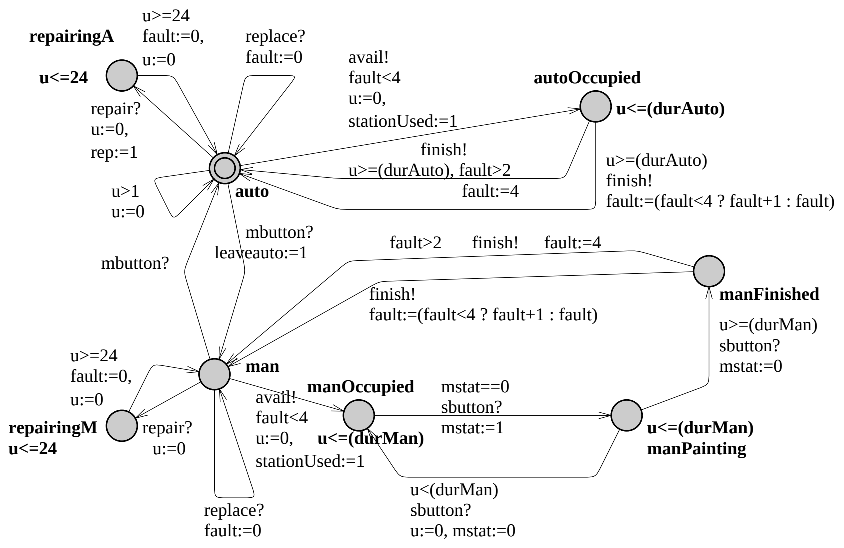

Figure 3. The paint station (Key: u: clock, fault: fault severity, mbutton: toggle manual/automatic painting, sbutton: press/release manual paint button, mstat: global flag denoting that manual painting is in progress, leaveauto: decoration that flags a mode change to manual mode).

3. The *waiter* automaton models the part of the system containing the queue of boxes waiting to be serviced by the paint stations as well as the lift that causes the boxes to be moved to one paint station or the other. It also models a repository for unpainted boxes that have fallen of the queue because the queue is too long, see Figure 4.
4. The final physical element, the *receiver*, models the belt of finished items, see Figure 2b.

3.1 The human interface and scheduling mechanism

Two processes are designed to reflect what the user does. *User* dispatches conditional user inputs and models simple repair/replace decisions: "if the fault (variables p1fault and p2fault) is bigger than 3 and sufficient funds (variable win) are available, replace a nozzle, otherwise perform a repair", see Figure 5a.

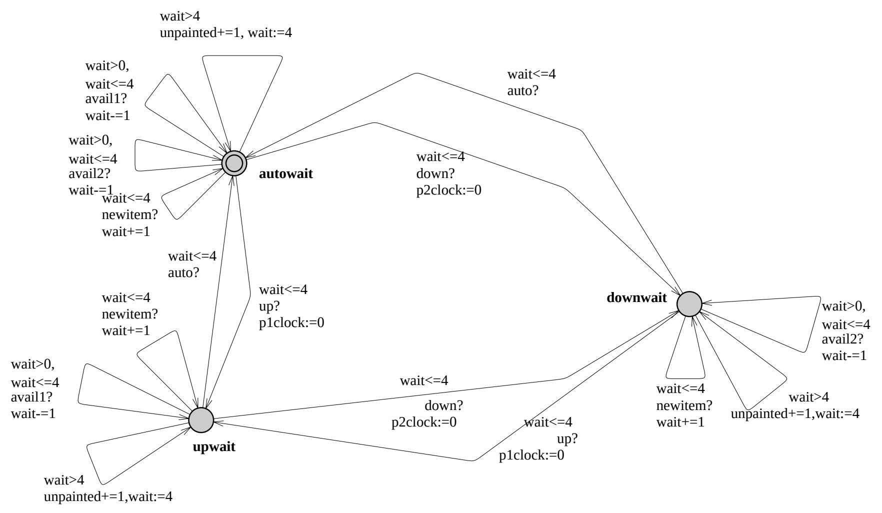

Figure 4. Boxes waiting to be channelled to the appropriate station (Key: `wait`: dispatched items waiting in queuing area, `unpainted`: overflow queue ot items failing to reach paint station, `p1clock,p2clock`: local clocks of paint stations).

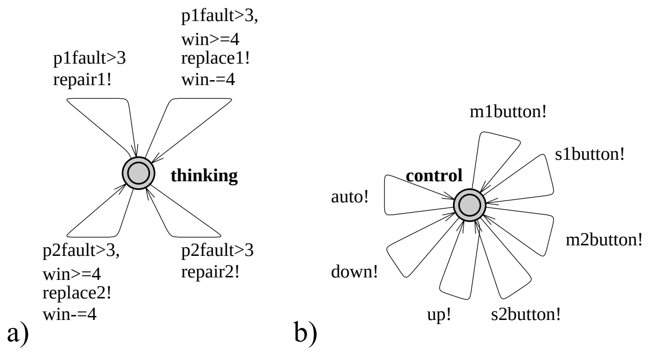

Figure 5. Simple models of (a) a user who implements a simple strategy and (b) a random user (Key: `win`: current earnings; `p1fault,p2fault`: fault severity of stations 1 and 2; `repair1,repair2,replace1,replace2`: repair/replace decision; `auto`: toggle automatic station selection; `up/down`: select paint station manually; `m1button, m2button`: toggle manual/automatic painting; `s1button, s2button`: press/release manual paint button).

The *randomizer* (Figure 5b) provides an alternative process to the user which dispatches unconditional user inputs that are consumed by other processes (“monkey at the keyboard” style) but only generated when no internal synchronisations can be performed.

4. THE ANALYSIS

Analysis was performed on the system in a number of steps. Starting with some sanity checks to gain confidence that the model performs as intended, properties are then formulated in order to investigate possible scheduling decisions.

4.1 Sanity Checks

At this level properties are intended to assess whether the model provides the base functionality of the system effectively. Properties in this category include deadlock freedom and the reachability of system states that represent crucial system features, such as (i) different lengths of the drop-out queue, (ii) switching between automatic and manual paint mode, (iii) switching between paint stations and (iv) the concurrent operation of both paint stations.

4.2 Reachability of system goals

Once the results of the analysis in Section 4.1 give confidence that the model behaves as intended, the next stage is to assess whether system goals can be reached. For instance:

P1: *Can all n items be painted?*
The property (“`E<> painted==`*n*”) is true for $0 \leq n \leq 10$.

When the negated property (here, the *never-claim* “`A[] painted!=`*n*” – “*n* items can never be painted”) is checked, the model checker produces a trace that can be simulated. Stepping through that trace, the analyst is guided through a scenario where both manual and automatic mode of painting are applied. The simulation and the sequence chart provided by `uppaal` can point to simple flaws or instances of unexpected behaviour of the model. In order to obtain a broader understanding of the reasons behind flaws, additional traces of similar instances are required. However, the tool only produces a single trace for each property. Additional traces, focussing on different aspects that may be considered contributing factors to a discovered problem, require a refinement of the property. For instance:

P2: *Can all n items be painted, using only a single paint station?*
For the analysis of this property the verifier shall explore only paths that involve a single instance of the paint station process. This is achieved by temporarily decorating the paint station by a write-once flag

`stationUsed` (see Figure 3) that cannot be reset and that would be set to `1` if the second paint station was used. Property P1 then needs to be extended by a condition "`stationUsed==0`".

4.3 Finding out minimal durations under different conditions

Having considered properties associated with the verification of the model and with the reachability and mechanisms for achieving specific goals, the next stage is to consider temporal issues associated with the paint shop model.

P3: *Can all n items be painted in m time units, using only a single paint station?*
"`E<>(painted==`*n* `and stationUsed==0 and gtime==`*m*`)`"
This property was checked for different values *m* of a global clock gtime, in order to establish the minimal duration[2] (in this case 22 units for ten items, but the nozzle needs to be replaced at least twice, so the win is only two units – see first row of Table 1). Similarly, one can ask:

P4: *Can all items be painted in m time units, using both paint stations?*
Again, a minimal duration of 22 time units was found. However, while the execution time remains the same this time, only one of the nozzles needs to be replaced, so the monetary win is six units.

All traces above confirm that the fastest way to perform the work is to opt to paint it manually (compare top and bottom of Table 1). The effect the automatic strategy had on the duration was then analysed.

P5: *What is the minimal time required to paint all items automatically?*
Here, user intervention is recorded by decorating the paint-station automaton with a temporary global write-once flag `leaveauto` (following the procedure described for property P2 above). The minimal time required to paint all items without manual intervention and by using both stations is 29 units.

[2] From version 3.4 of `uppaal` it is possible to access execution duration for the trace that is generated. This is achieved using the "fastest" option within the "diagnostic trace" menu. This feature of the tool consumes a lot of resources and it turned out to be easier to use the cruder approach of iterating over *m*.

Table 1. Minimal paint durations for manual and automatic mode allowing replacement costs.

Minimum required time for `manual` painting alone; allow replace											
duration (win)		number of items									
		1	2	3	4	5	6	7	8	9	10
1 station	duration	4	6	8	10	12	14	16	18	20	22
	win	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)
2 stations	duration	4	6	8	10	12	14	16	18	20	22
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(5)	(6)

Minimum required time for `automatic` painting alone; allow replace											
duration (win)		number of items									
		1	2	3	4	5	6	7	8	9	10
1 station	duration	7	12	17	22	27	32	37	42	47	52
	win	(1)	(2)	(3)	(4)	(1)	(2)	(3)	(4)	(1)	(2)
2 stations	duration	8	10	12	14	17	19	22	24	27	29
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(5)	(6)

Table 2. Minimal paint durations for manual and automatic mode for maximising earnings.

Minimum required time for `manual` painting alone; maximise `win`											
duration (win)		number of items									
		1	2	3	4	5	6	7	8	9	10
1 station	duration	4	6	8	10	34	36	38	40	64	66
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
2 stations	duration	4	6	8	10	12	14	16	18	42	44
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)

Minimum required time for `automatic` painting alone; max. `win`											
duration (win)		number of items									
		1	2	3	4	5	6	7	8	9	10
1 station	duration	7	12	17	22	51	56	61	66	95	100
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
2 stations	duration	7	9	12	14	17	19	22	24	48	50
	win	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)

The remaining row in Table 1 was obtained by analysing property P3 extended by condition "`leaveauto==0`". The analysis so far yields the following findings that might be used to devise operation strategies:

1. Painting items manually is faster than automatic painting.
2. Using both stations does not necessarily gain a time advantage over using a single station only.
3. However, using both stations can save repair costs if the operator is prepared to take the risk and leave one station broken.

It should be noted that the temporal properties of this stage could have been calculated in an alternative way by using a simple numeric model of the processes. However, the additional effort of creating the `uppaal` model pays off when multi-valued decisions are considered, as the following section demonstrates.

4.4 Focussing on monetary costs

So far the analysis has only been concerned with temporal costs and effects. The following properties have been used to check temporal *and* monetary costs associated with replacing faulty nozzles.

P6: *Can all boxes be painted without losing money?*
This property forces a search strategy where nozzle replacements are avoided. The resulting trace demonstrates that the task can be completed in 50 time units. The simulation demonstrates that both stations are used to paint in automatic mode until they break; then one station is repaired.

P7: *What is the shortest time for painting everything without losing money?*
The analysis yields that best performance (finishing the task in 44 time units) can be achieved, and the new trace suggests that this performance can only be achieved if manual control is selected. Again, both stations break, but the trace indicates that only one station needs to be repaired.

P8: *Can all items be painted without losing money, using only one paint station?*
This analysis is dual to P6, but focussing on a single paint station only (using the boolean flag procedure described in P3). This property is concerned with the robustness of the system and the additional temporal costs. The strategy exhibited by the model-checking trace could be used by an operator who does not have time pressure and therefore aims at maximising the win.

Analysing the durations under the assumption that temporal costs are secondary to monetary costs (see summary in Table 2) reveals again that the best possible performance can be achieved by using both stations in manual mode, but the required duration increases to 44 units.

The results produced so far give some indication of what a good operation strategy might be under temporal and monetary cost extremes. However, it remains the task of the system designer to resolve if any of these strategies are suitable, and if they should be implemented as part of the system or as part of the operator training. For an informed decision it also remains necessary to draw on human-factors experience. A crucial additional factor that will influence this decision is the operator workload.

4.5 Variable workload

The analysis so far was performed assuming a constantly high workload, given by the *dispatcher* model in Figure 2a. The analysis can be repeated – using increasing, decreasing or alternating workloads – in order to collect insights about further strategies. Possible modifications of the dispatcher automaton are shown in Figure 6. However, for the purpose of this paper this analysis is omitted here.

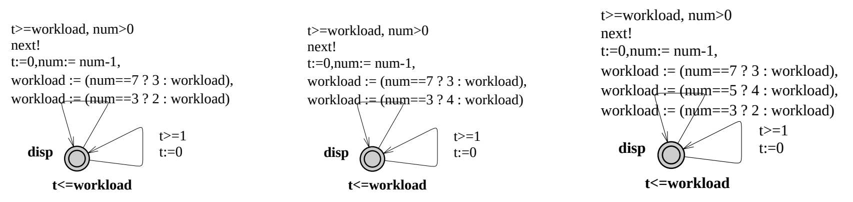
t>=workload, num>0
next!
t:=0,num:= num-1,
workload := (num==7 ? 3 : workload),
workload := (num==3 ? 2 : workload)
disp
t>=1
t:=0
t<=workload
t>=workload, num>0
next!
t:=0,num:= num-1,
workload := (num==7 ? 3 : workload),
workload := (num==3 ? 4 : workload)
disp
t>=1
t:=0
t<=workload
t>=workload, num>0
next!
t:=0,num:= num-1,
workload := (num==7 ? 3 : workload),
workload := (num==5 ? 4 : workload),
workload := (num==3 ? 2 : workload)
disp
t>=1
t:=0
t<=workload

Figure 6. Modelling (a) increasing, (b) decreasing and (c) alternating workloads.

5. CONCLUSIONS

This paper discussed the feasibility of using model checking techniques to explore scheduling constraints in dynamic production systems under worst-case fault conditions. How the process might help in articulating the problems that must be resolved by human factors experts has also been briefly considered.

A number of problems emerged during the modelling and the analysis which could limit the utility of a model checking approach. First, model checking is not yet a light-weight method. Generating a state model is an effortful and time-consuming exercise, unless a model of the physical characteristics of the system has been produced in earlier stages of the design process. This is a problem which occurs equally with most other formal modelling approaches, such as for instance micro-world simulation and is reduced as the modeller's skill increases.

Simplification of some physical characteristics (see Section 3) makes the model less representative of the physical system, and failures related to the interactions of these non-linear processes might be missed.

Another problem is introduced when the human operator is to be modelled. It is important to make the right assumptions about the operator's control mode (scrambled, opportunistic, tactical or strategic), the accuracy of the operator's temporal awareness (knowledge of available and required time, dynamics of change, probability of events, and so forth) and the operator's general situation awareness when modelling temporal reasoning performance. Formal modelling may improve the design process as it makes explicit the designer's assumptions about agents' capabilities, performance and availability, about the value and priority structure of functions in the system, and about the costs and benefits of a particular control strategy. Although the richness of naturalistic planning and control processes, and the complexity of scheduling decisions, may not be captured by these models, they help to assess how robustly a set of prototypical control strategies perform across a range of operational circumstances. This preliminary investigation explored only very simple strategies, and only analysed the effects on safety properties. These strategies tend to be focussed on extreme situations, such as gaining maximal earnings in a minimal duration. Consequently, the stated goal of assessing under what circumstances certain action can and should be delayed is limited to extreme behaviour. This is useful, since it is often extreme situations where failure has particularly dangerous effects. Although solutions to resolve extreme situations are relevant, it is essential to also consider the "normal" operating conditions. It is argued that these techniques are also useful in posing the problems clearly that must be solved by human factors experts for the particular system.

For the purpose of informing design decisions the value provided by traces that are obtained from the model checker is limited. The traces that are obtained represent single instances of behaviour that may indicate problems in the design. The `uppaal` tool supports the understanding of the component behaviour in a trace by providing animations of the automata. Additionally, the message sequence chart visualisation provides insights about the inter-process communication in the trace. However, single instances of behaviour rarely provide sufficient insights to discover problem tendencies. For a broader understanding of a problem, a *set* of traces that describe the same problem would be required. To our knowledge, no tool currently provides such information. The analysis of scheduling trade-offs will most likely require a combination of several different approaches. These will include queuing models, production scheduling models, simulation approaches, work and task analysis techniques, and experimentation.

Future work will concentrate on assessing the contributions that each of these approaches can make towards improving our understanding of temporal planning and control, and their limitations in representing temporal properties. The appropriate method or methods for analysing flexible scheduling might be domain specific, as work domains themselves differ dramatically in their temporal properties (e.g. slow versus fast, synchronised versus independent, continuous versus discrete, periodic versus aperiodic, concurrent versus sequential, event-driven versus self-paced).

Work on elaborating the `uppaal` model of the paintshop continues. Parallel to this activity, a javascript micro-world simulation of the system has been developed in order to perform experimental studies (Hildebrandt and Harrison, 2003). In these studies, a human operator had the task of controlling paintshop. The study is currently being evaluated, and the results will be used to refine the `uppaal` model.

REFERENCES

Block, R., editor, 1990, *Cognitive Models of Psychological Time*. Lawrence Erlbaum Associates.

De Keyser, V., 1995, Time in ergonomics research. Ergonomics, **38**:1639–1660.

Doherty, G., Massink, M., and Faconti, G., 2001, Using hybrid automata to support human factors analysis in a critical system. *Journal of Formal Methods in System Design*, **19**(2):143–164.

Grosjean, V. and Terrier, P., 1999, Temporal awareness: Pivotal in performance? *Ergonomics*, **42**:1443–1456.

Hancock, P. and Scallen, S., 1998, Allocating functions in humanmachine systems. In R. Hoffman, M. S. and Warm, J., editors, Viewing psychology as a whole: the integrative science of William M. Dember, pp. 509–537.

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H., 1997, HyTech: A Model Checker for Hybrid Systems. Software Tools for Technology Transfer, pp. 110–122.

Hildebrandt, M. and Harrison, M., 2002, Time-related trade-offs in dynamic function scheduling. In Johnson, C., editor, 21st European Annual Conference on Human Decision Making and Control, pages 89–95. GIST Technical Report G2002-1, Department of Computing Science, University of Glasgow, Scotland.

Hildebrandt, M. and Harrison, M. D., 2003, PaintShop – A microworld experiment investigating temporal control behaviour. Technical report, DIRC.

Hollnagel, E., 2000, Modeling the orderliness of human action. In Sarter, N. and Amalberti, R., editors, Cognitive engineering in the aviation domain. Lawrence Erlbaum Associates.

IJHCS, 2000, *International Journal of Human-Computer Studies*. **52**(2), Special Issue - Dialogues on Function Allocation.

Larsen, K. G., Pettersson, P., and Yi, W., 1997, Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, **1**(1–2):134–152.

Michon, J., 1990, Implicit and explicit representations of time. In Block, R., editor, *Cognitive Models of Psychological Time*, pp. 37–58. Lawrence Erlbaum Associates.

Scerbo, M., 1996, Theoretical perspectives on adaptive automation. In Parasuraman, R. and Mouloua, M., editors, Automation and Human Performance: Theory and Applications, pp. 38–63. Lawrence Erlbaum Associates.

Svenson, O. and Maule, A., editors, 1993, Time Pressure and Stress in Human Judgement and Decision Making. Plenum Press.