USING TIMED AUTOMATA FOR RESPONSE TIME ANALYSIS OF DISTRIBUTED REAL-TIME SYSTEMS

Steven Bradley * **William Henderson** ** **David Kendall** **

* *Department of Computer Science, Durham University, South Road, Durham, DH1 3LE, UK*
** *Department of Computing and Mathematics, University of Northumbria at Newcastle, Ellison Place, Newcastle upon Tyne, NE1 8ST, UK*

Abstract: Rate Monotonic Analysis (RMA) is a well-established technique for assessing schedulability of periodic and sporadic tasks which share a processor resource using fixed priority scheduling. Adaptations of this technique have been made to perform Response Time Analysis (RTA), accounting for jitter, blocking, distributed systems and end-to end timing constraints. However, the nature of the analysis means that, while good bounds can be given for uni-processor systems with relatively little interdependency, the response times calculated for more complex systems can be very conservative.
An alternative approach to analysing such systems is to build a model which represents the behaviour of the system more dynamically, taking into account the dependency between the tasks. To do this, we introduce a simple language for describing the tasks which comprise a system and the precedence relationships between them. From this a timed hybrid automaton is generated which can be analysed automatically to predict end-to-end response times.
Applying this technique in practice yields promising results, with response times lower than those calculated with RTA. However, there is a trade-off to be made between the complexity of the hybrid automaton analysis (which suffers from the state explosion problem) and the conservatism of the more standard RTA approach.

Keywords: Scheduling algorithms, Timing analysis, Real-time tasks, Real-time operating systems.

1. INTRODUCTION

Since rate-monotonic scheduling (RMS) and rate-monotonic analysis (RMA) were first proposed (Liu and Layland, 1973), work has been carried out to extend the basic model of computation from a set of independent periodic tasks with fixed execution times sharing a single processor, and to extend the analysis from simple schedulability to system-wide end-to-end response times (Audsley *et al.*, 1993). Audsley *et al.* (1995) report on the development of the theory supporting fixed priority pre-emptive scheduling, including extensions to account for interdependence of tasks (through blocking) and the analysis of distributed systems. Analyses of end-to-end response times in distributed systems have been carried out using TDMA (Tindell and Clark, 1994) and more recently with CAN (Henderson, 1998) as the communication mechanism.

Although more recent work has taken account of extra delays incurred through interdependence of tasks, ensuring that the analysis remains conser-

vative, one area which has not received so much attention is to take advantage of the restrictions on possible execution paths brought about by interdependence. This has meant that predicted response times can be overly pessimistic. In this paper we aim to address the problem of pessimism by explicitly modelling precedence (or ordering) relationships between tasks, and performing an exhaustive analysis of all possible execution paths through a system. The analysis is carried out be performing reachability analysis on a timed hybrid automaton (Alur *et al.*, 1995).

The motivation for this work is demonstrated in section 2 through an example which exhibits pessimism under a standard static response-time analysis (RTA) (Audsley *et al.*, 1993). In section 3 we introduce our system model, and describe a simple language (PG) in which such systems can be expressed. We then define a translation from PG into timed hybrid automata in section 4. Sections 5 and 6 explain how response time analysis is carried out, and evaluate the results of the analysis. Finally, our conclusions and relationships with other work are presented in section 7.

2. EXAMPLE OF PESSIMISM

Our simple example consists of a distributed system with two processors connected by a single CAN-style bus, with non-destructive priority-based arbitration. On processor 1 there are two similar tasks, `Sender A` and `Sender B`. These tasks are periodically triggered, and they each read a sensor, the values of which need to be sent to corresponding receiver tasks, `Receiver A` and `Receiver B`, on processor 2. This is achieved by sending messages `Message A` and `Message B` via the bus. We shall assume that each of the tasks and messages has a fixed resource requirement of 200, so we have essentially two clone sub-systems, system A and system B, each of which has a periodically triggered sender task, which triggers a message upon completion, which in turn triggers a receiver task. The tasks and message of system A have higher priority than those of system B.

In a control situation where the receiver task acts upon the information provided by the sender task to provide feedback into the controlled system, a crucial factor in determining whether the system can be controlled in a stable way is the response time from the reading being taken to the control being applied. Therefore, as well as examining schedulability issues on the processors and bus, it is important to be able to predict the end-to-end response time from the start of the sender task to the end of the receiver task.

The analysis of the end-to-end response time for subsystem A is reasonably straightforward, but subsystem B is more interesting. A standard response-time analysis adds the response times for each section, with each section having contributions from interference by higher priority tasks, blocking by lower priority tasks, and execution time. All sections (`Sender`, `Message` and `Receiver`) of subsystem B are subject to interference of up to 200 time units from the corresponding section of subsystem A, giving a total response time of 600(interference)+600(execution) = 1200. However, taking into account the precedence constraints, we can try to construct the actual worst case for the response time of subsystem B. This occurs when `Sender B` is released and just fails to complete before `Sender A` is released, which then pre-empts `Sender B`. The sequence of events which follows is shown in figure 1. Because of the precedence relationships, it is impossible for competition to occur between subsystems A and B for all three sections if both systems use their maximum resource requirement (200) in all cases. This gives an actual worst-case end-to-end response time of 1000; this kind of pessimism has been noted experimentally (Henderson, 1998).

Such an argument is difficult to construct convincingly by hand, even for a relatively simple example, but the consideration of the many different cases that may occur seems a good candidate for automation. To automate the analysis, however, we first need to declare our assumptions about the systems we analyse, and define the language we are going to use to describe the systems.

3. PG: A SIMPLE LANGUAGE FOR DESCRIBING DISTRIBUTED SYSTEMS

In a simple response time analysis, the precedence relationships between tasks are not made explicit, but the period of a task which is triggered by the completion of some other task is inherited from the triggering task. Our model of computation is very similar to that used in rate-monotonic style analyses, except that we explicitly model these precedence relationships. The most basic components in our model are *tasks*, which can be in one of three states: waiting (to be released), ready (to run, but not running) and running. Each task has a simple life cycle: it is released at some *trigger* event, after which it competes for some *resource* (e.g. access to processor) until completion. The time taken to complete will depend on its *bounded resource requirement* (e.g. processing time) and upon the competition for the resource. Competition is managed using fixed priority preemptive scheduling, although tasks may be declared as *nonpreemptible*, in which case they will retain access to the resource from their first access until their completion. A trigger event can either

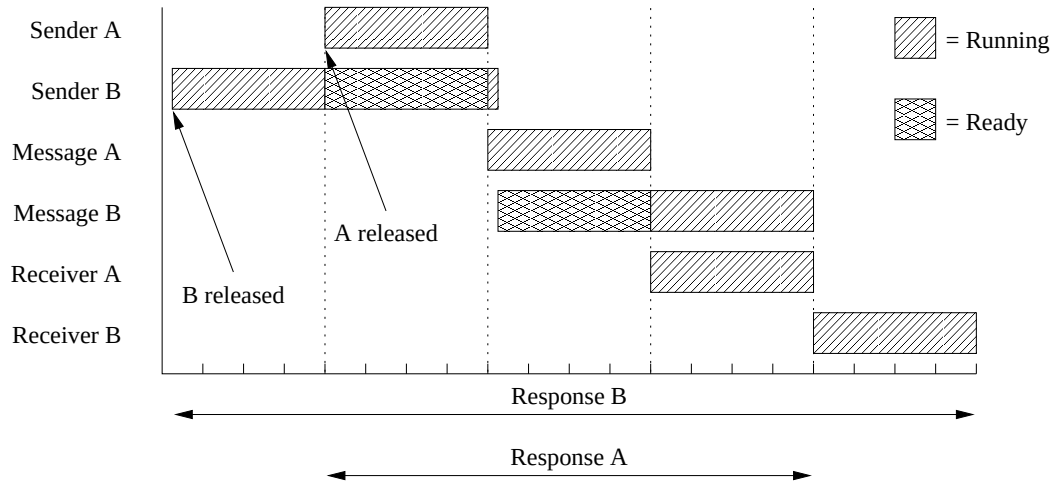

Fig. 1. Example of pessimism

be the elapsing of a fixed period, or the completion of some other task (or message). Communication is assumed to take place via a CAN-style bus with non-destructive priority-based arbitration. In this model, messages have the same properties as nonpreemtible tasks. Critical sections can also be modelled as nonpreemtible tasks. The following assumptions are made:

- each task/message uses exactly one resource
- each task/message has exactly one trigger
- no task/message can be triggered by a task/message that it directly or indirectly triggers (there are no 'loops')
- bounded resource requirements account for any overheads (such as kernel activity, memory management etc)

Important differences between this model and a more standard response time analysis model are that

- Precedence (i.e. triggering or ordering) relationships are made explicit, rather than assuming that all tasks/messages are periodic, with period inherited from their trigger.
- Jitter (i.e. possible delay of release after period has elapsed) is not included, as this is caused by the inheriting of periods from triggers: if an 'upstream' task or message does not complete in constant time, then its completion will not be purely periodic.

In summary, each task or message has a name, an associated resource, a bounded resource requirement, a priority, and a trigger. These are expressed using a very simple syntax: the example of section 2 can be expressed in PG as

```
task senderA on processor1 needs [100,200]
    at priority 0
    triggered by period 1300
task senderB on processor1 needs [100,200]
    at priority 1
    triggered by period 1400
message messageA on can needs [100,200]
    at priority 0 nonpreemptible
    triggered by senderA on processor1
message messageB on can needs [100,200]
    at priority 1 nonpreemptible
    triggered by senderB on processor1
task receiverA on processor2 needs [100,200]
    at priority 0
    triggered by messageA on can
task receiverB on processor2 needs [100,200]
    at priority 1
    triggered by messageB on can
```

This can also be presented graphically, a more complex example is illustrated in figure 3 where the precedence relationships are shown with arrows (periodic triggers are shown as squares labelled with the period). It is this presentation, as an acyclic directed graph where the nodes are tasks or messages, and the edges are precedence relationships, which gives rise to the name PG: precedence graph.

4. MODELLING DISTRIBUTED SYSTEMS WITH HYBRID AUTOMATA

Before explaining how we translate system descriptions in PG into hybrid automata (section 4.2), we first briefly review the definition of hybrid automata (section 4.1). For a more detailed description including a more formal definition see Alur *et al.* (1995).

4.1 *Hybrid Automata*

Timed hybrid automata are an extension of timed automata (Alur and Dill, 1990). The basic notion is that of a finite state machine, extended with real-valued variables. These variables can be used to form enabling conditions for transitions (by comparing with a constant value), and to form

invariant conditions for states. Variables can be reset when transitions take place, but cannot be assigned a value other than 0, and cannot be compared with each other. In a standard timed automaton all variables increase their value in line with the increase in global time, and are known as clocks. In a hybrid automaton, the rate at which a variable changes may vary with the state, but this rate will always be a natural number (an integer $n \geq 0$). For the purposes of this paper, we will only need to consider rates of 0 or 1. A hybrid automaton is defined by

- a set of states S
- a set of transitions between states $T \subset S \times S$
- a set of variables V

A *variable valuation* is a function $v : V \rightarrow Real$ which assigns a real value to each variable.

Associated with each state in S is

- an invariant function which takes a variable valuation and specifies whether it is possible to remain in the state with that valuation
- a rate of change for each variable. We will use the convention that *clocks*, with variable names based on C (e.g. $C1, C_{period}$) always have a rate of 1, so do not need to be given with the state information. *Integrators*, with names based on I will have a rate of 0 or 1, depending on the state.

Associated with each transition in T is

- an enabling function which takes a variable valuation and specifies whether the transition is allowed with that valuation
- a reset function which specifies which variables are to be reset to 0 when the transition is taken.

4.2 *From PG to hybrid Automata*

In order to translate from a distributed system described using PG to a hybrid automaton, we first of all need to construct the set of states. Each task or message can be in one of three conditions: waiting, ready, or running. (In the following, read 'task or message' for 'task'). The whole state space consists of all possible combinations of each of these conditions for each of the tasks, plus state(s) corresponding to failure. Transitions between states correspond to the triggering or completion of tasks. If a task is released in a state where it is not already runnable, then a transition is made to the state where that task is runnable. The condition of each of the other tasks is not changed, unless the task that has been released has a higher priority than a preemptible running task which shares the resource. In this case, the preemptible task has its condition changed from running to ready. If a task is released in a state where it is already runnable, then this indicates a problem with schedulability, so a transition is made to a fail state. Invariants are added to states to ensure that some transition is made (either trigerring or failure) once the period has elapsed.

A clock variable is required for each periodically triggered task. This clock will be reset at the start of the period, and will be used to evaluate the enabling of transitions corresponding to the release of the task. An integrator variable will also be required for each task, to keep track of how much resource time has been used by that task. When a task is released, the integrator for that task is reset. In a state where a task is running, the corresponding integrator will have rate 1; otherwise the integrator will be 0. Transitions corresponding to task completions are enabled by the integrator for that task being greater than the minimum resource requirement. Invariants are added to ensure that completion transitions are taken at or before the maximum resource requirement has been used. The destination state of a completion transition takes into account the change of condition of the completing task to waiting, and also the release of any tasks triggered by the completion. As before, if a triggered task is already in a runnable condition then the destination state will be a failure state. Figure 2 shows the timed automaton corresponding to the PG definition given as

```
task A on processor needs [100,150]
  at priority 0 triggered by period 250
task B on processor needs [50,75]
  at priority 1 triggered by period 300
```

This example does not demonstrate all features of building the graph, as there is is only one resource, there are no non-preemptible tasks, and there are no completion triggers (as opposed to periodic triggers), as any example which covers any or all of these yields too large an automaton to present easily. See section 6 for a discussion of the size of the resulting automaton.

5. RESPONSE TIME ANALYSIS

Having built an automaton which models the behaviour of the system we wish to analyse, we now discuss how to perform the analysis upon the model. If we are only interested in schedulability then we need to check whether there is any allowable sequence of events which leads to a failure state. However, we are interested in more than schedulability: as we stated earlier, end-to-end response times are of interest, particularly for distributed systems. We extend our language slightly to define any end-to-end properties we are interested in, such as

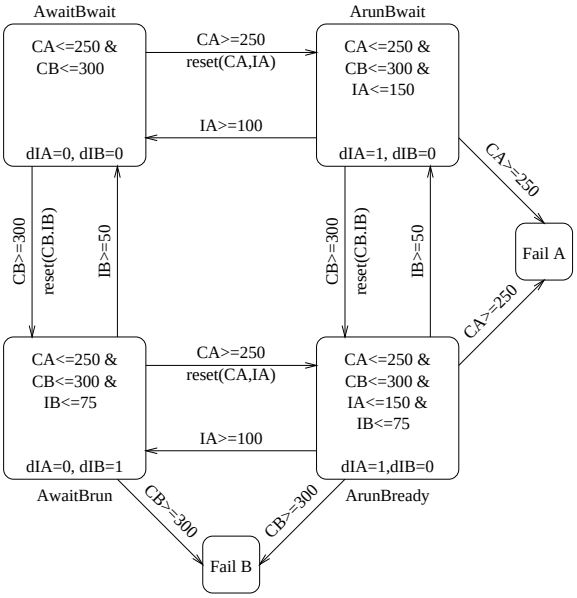

Fig. 2. Automaton for two periodic tasks

```
property loop_A
 from start senderA on processor1
 to end receiverA on processor1
```

for the example shown in figure 3. Each property adds one further state and its associated transitions to the automaton. The state is to be used during reachability analysis, and is entered only when the property is violated. A new clock variable is added, and this clock is reset at the start of the first named task. All states which are on the trajectory of this end-to-end response have a transition to the fail state added (the assumption that each task has only one trigger is needed to unambiguously identify whether a state is on a given trajectory). Specific claims about end-to-end response times (e.g. that the response will occur within 1000 units) can be checked, by adding a `within` clause to the end of the property. In this case, the enabling condition of the fail transitions will be that the property clock has not exceeded the deadline set.

A more sophisticated analysis can be carried out by using the parameterisation mechanism within HyTech (Henzinger *et al.*, 1995). Instead of giving a literal constant, a declared parameter can be used instead. The constraint-solving engine which underlies the model-checker will then calculate the conditions on the parameter(s) under which reachability can occur. In our case, the set whose reachability we are interested in is the failure set, so by fixing the enabling condition on failure transitions to be bounded by a parameter, the model-checker will calculate the least value of the parameter for which failure will occur; in other words, the end-to-end response time. Note that because the property clock is reset at the release of the first named task, the response time recorded will be at most the minimum inter-release time (the period) for this task.

Size	RTA		PG		T_{calc}
	A	B	A	B	
5	1600		1000		1.3
3 + 3	800	1200	800	1000	24
4 + 4	1400	2400	1200	1400	108
5 + 3	1800	1600	1200	1000	87
4 + 5	1400	3000	1200	1600	271
5 + 4	1800	2600	1400	1400	280
5 + 5	1800	3400	1400	1600	1083

Table 1. Results for reachability analysis

6. EVALUATION OF RESULTS

There are two main attributes of our technique which we wish to evaluate:

(1) the values for response times given
(2) the time taken to calculate the response times

We wish to compare the response times with those achieved experimentally (we always want to be conservative), and with those achieved using static analyses (we want to be less pessimistic). The calculation times stand by themselves, as comparison with static analyses will almost always be unfavourable: standard RTA is computationally cheap. The question to be asked here is whether the reachability problem is tractable, and whether any gains made in predicting better response times are worth the computational effort. To make the comparison, a range of systems were chosen, with varying sizes (number of tasks and messages) and varying amounts of inter-dependence (length and number of chains of precedence). Initially, the assumption that all messages and tasks had the same bounds on resource requirement was kept. The example of section 2 can be seen as two cloned precedence chains, each of length three. This example was extended to chains of length four and five, by adding first another message, and then a further task. The largest example considered had two chains of length five; the precedence graph for this example is shown in figure 3. Periods were chosen to have a GCD of 100, to ensure that the results were not sensitive to slight changes in period. Table 1 shows the results obtained using on a 233MHz Pentium II under Linux with 64MB RAM and 256M swap space. Calculation times shown are reported user time (in seconds) to perform the reachability analysis. Time taken to build the automaton is not always small, but is insignificant compared to the reachability analysis.

The examples may appear contrived, as conflicts are forced by having two essentially identical systems operating. However, this example is one of

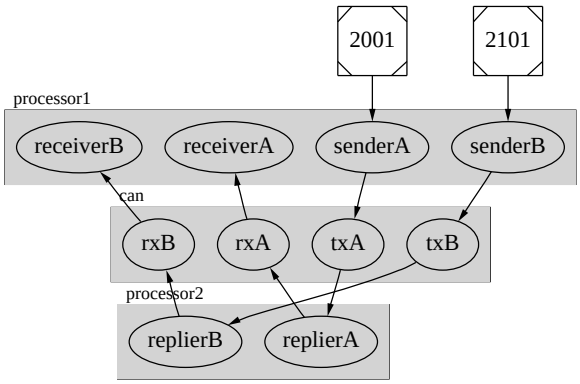

Fig. 3. Two precedence chains of length five

the simplest that demonstrates the kind of dependency that we can usefully identify. Typically, and unsurprisingly, the more dependency that can be identified in a system, the more can be gained from a detailed analysis of those dependencies. Most benefit is gained when chains of dependency are comprised of equal-length segments; in other words, the advantages of considering global properties are maximised when the response time is not dominated by local properties. Also, the results show that most benefit is gained on lower-priority chains of precedence.

7. CONCLUSIONS

We have argued that existing analysis techniques for response time can yield overly pessimistic results for distributed real-time systems with precedence constraints, and have presented a language (PG) for describing such systems. The translation of system definitions written in this language into hybrid automaton models has been discussed, along with the use of parameterised reachability analysis to derive end-to-end response times from the models. This technique has been applied to a range of examples with some positive results.

Related work has been carried out in two main areas:

- The adaptation of standard Response-Time Analysis (RTA) techniques to account for *offsets* which can be used to make the analysis less pessimistic (Tindell, 1994). However, Audsley and Burns (1998) have shown that when task periods are co-prime, this technique fails.
- Corbett (1996) has used timed automaton modelling of Ada tasking programs to carry out scheduling analysis. His work is related to ours, but is based on an explicit model of the scheduler, yielding less tractable (larger) models, and is only applied to uni-processor systems.

Further work is needed in the area of comparing the results we have obtained with those achieved in practice, and also with those obtained by standard and adapted RTA. There is also the possibility of adapting the approach to include more complex scheduling behaviour, such as that given by the priority ceiling protocol or with the use of dynamically assigned priorities (e.g. earliest-deadline first).

8. REFERENCES

Alur, R and D Dill (1990). Automata for modeling real-time systems. In: *17th International Colloquium on Automata, Languages and Programming (ICALP 90)*. number 443 In: *Lecture Notes in Computer Science*. Springer.

Alur, R, C Courcebetis, N Halbwachs, T A Henzinger, P-H Ho, X Nicollin, A Olivero and J Sifakis nad S Yovine (1995). The algorithmic andalysis of hybrid systems. *Theoretical Computer Science* **138**, 3–34.

Audsley, N, A Burns, M Richardson, K Tindell and A J Wellings (1993). Applying new scheduling theory to static priority pre-emptive scheduling. *Software Engineering Journal* **8**(5), 284–292.

Audsley, N C, A Burns, R I Davis, K W Tindell and A J Wellings (1995). Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems* **8**(2/3), 173–198.

Audsley, N C and A Burns (1998). On fixed priority scheduling, offsets and co-prime task periods. *Information Processing Letters* **67**(2), 65–70.

Corbett, J C (1996). Timing analysis of Ada tasking programs. *IEEE Transactions on Software Engineering* **22**(7), 461–483.

Henderson, W D (1998). An holistic approach to performance prediction of distributed real-time can systems. In: *5th International CAN Conference, San Jose*. To Appear.

Henzinger, T A, P-H Ho and H Wong-Toi (1995). A user guide to HYTECH. In: *Tool and Algorithms for the Construction and Analysis of Systems: (TACAS 95)* (E Brinksma, W R Cleaveland, K G Larsen, T Margaria and B Steffen, Eds.). Vol. 1019 of *Lecture Notes in Computer Science*. Springer. pp. 41–71.

Liu, C L and J W Layland (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *JACM* **20**(1), 40–61.

Tindell, K (1994). Adding time-offsets to schedulability analysis. Technical Report YCS 221. Department of Computer Science, University of York.

Tindell, K and J Clark (1994). Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming* **40**(2-3), 117–134.