Reservation-Based Timing Analysis - A Practical Engineering Approach for Distributed Real-Time Systems

Alan Grigg, Neil Audsley
Real-Time Systems Group, Dept. of Computer Science, University of York, York, U.K.
{alan,neil@cs.york.ac.uk}

Abstract

This paper proposes a timing analysis technique that gives practical support to the industrial systems engineering process for large-scale, distributed real-time systems. The technique, known as reservation-based timing analysis (RBA), gives support throughout the life-cycle of the system, from the early stages of system development through to implementation stages and later in-service upgrades. RBA allows the timing properties of the system to be validated incrementally throughout system development and upgrade. This gives greater control over the costs associated with the development and upgrade of the real-time properties of the system.

1. Introduction and Problem Definition

Many real-time systems applications, such as those in the aerospace domain, perform safety-related functions and are subject to increasingly stringent certification standards like DEF STAN 00-56 [17] and DO-178B [18]. A combination of analytical models and testing techniques is required to support validation of their real-time behaviour. However, given the trend towards increasingly flexible development processes and target system architectures, typified by the civil and military integrated modular avionics (IMA) programmes [1, 2], the provision of appropriate analytical techniques is increasingly difficult.

This paper proposes an analytical technique that has been developed to meet IMA requirements but is generally applicable to large-scale, industrial distributed real-time systems. The work has been funded by BAE SYSTEMS as part of the Dependable Computing Systems Center programme. The key technical challenge is that the timing analysis method for IMA, like other aspects of the engineering process, must deal with two apparently conflicting goals :

- Evolvable system development (and upgrade) - Changes to system requirements, design or implementation details must be accommodated with minimal re-work (and associated costs). Such changes can arise from customer requests, rapidly evolving implementation technologies or, quite simply, technical problems encountered during development. Their impact is currently a major cost factor in the development of avionic systems and is predicted to be much worse in future systems due the trend towards more collaborative, multi-national development teams, in conjunction with…
- A highly integrated system architecture - The IMA architecture is integrated both *functionally* (with increased sharing of functions and data between traditionally independent aircraft sub-systems) and *physically* (with increased sharing of computing resources – processing and communication media). Timing analysis for integrated systems is prone to being *holistic* - it is difficult or impossible to analyse parts of the system in isolation from other parts. The effect of making even small changes to such systems upon their emergent timing behaviour are typically widespread. Hence, there is a need for re-validation of system timing properties and a resulting risk of re-work whenever a change is made.

2. Reservation-Based Timing Analysis

Traditionally, timing analysis has been applied to real-time systems relatively late in their development process, when all of the implementation details of the hardware and software are known. Analysis begins with the determination of *worst-case execution times* for individual software components executed on specific target hardware. Then, at successive stages of system integration, *feasibility analysis* is applied to determine whether or not the overall timing requirements are met for larger-scale applications and, ultimately, the system as a whole. The result of discovering any timing deficiencies relatively late in the development process, however, can be extremely costly, enforcing the need for re-implementation, re-design or even changes to the original timing requirements.

Reservation-based timing analysis (RBA) takes a different approach in order to meet the goals of IMA. RBA encourages the use of timing analysis much earlier in the development process, supporting predictions of end-to-end, system-level timing behaviour on the basis of the latest information available from the development process, allowing for some parts of the system to be

further ahead in their development than others. Figure 1 illustrates the overall RBA process based on the classical V-model.

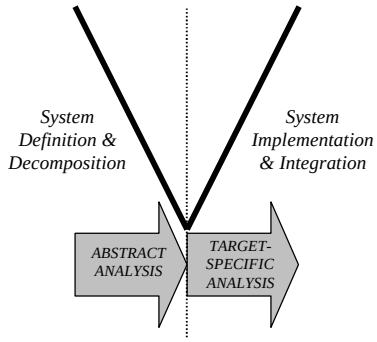

Figure 1. RBA Through the System Life-Cycle

Abstract timing analysis is performed during the decomposition stages of development by adopting :

- an abstract model of the target computing platform;
- the use of timing and resource 'budgets' that can be successively evolved.

Target-specific timing analysis is consequently performed during system implementation and integration with the following constraints :

- the assumptions made in the abstract model of the target computing platform need to be preserved;
- the timing and resource budgets must be validated (shown to be sufficient) for the actual target.

At successive stages of system development, timing *guarantees* can be generated in conjunction with a set of timing *obligations* that must be observed during subsequent stages of development. In this way, the timing analysis model can evolve along side the development of the system. A key property of the RBA model is that it allows the timing properties of the system to evolve with minimal re-validation by keeping the impact of localised changes as local as possible and avoiding the need to re-analyse large parts of the system or even the whole system, as is often the case with timing models of distributed systems. In order to achieve this, RBA adopts a partitionable analytical model of timing behaviour based on the approach of *resource reservation*.

The notion of resource reservation is, in itself, nothing new - scheduling solutions and communication protocols have been reported in recent literature on real-time systems. Notable contributions include scheduling solutions for multimedia applications [8, 9, 10], techniques for partitioning of real-time and non-real-time applications [11, 12] on a uni-processor and communication bandwidth reservation schemes [14, 15, 16]. The contribution of RBA is two-fold :

- a consistent, bandwidth-based timing model for prediction of end-to-end delays across diverse resource types in a distributed real-time system - all processing, communication and other resources modeled using the same basic parameters;
- a timing model that can be instigated early in the system development process and updated incrementally as the system development evolves.

For the practical application of RBA to large-scale, distributed real-time system engineering projects, tool support will be required to both manage the details of the evolving timing model and to automatically determine the end-to-end timing behaviour of the system. Such a tool has been developed for research purposes for a MATLAB/Simulink modeling environment and was used to generate the example transactions and results given in the rest of this paper.

3. Representation and Decomposition of System Timing Properties

The end-to-end timing and resource characteristics of the system are captured, decomposed and validated in terms of *transactions* and *activities*. A *transaction* describes the temporal relationship between :

- a set of *input events* from the external environment or some other transaction(s);
- a set of *output events* to the external environment or some other transaction(s).

The body of a transaction is described in terms of :

- *activities*, that capture resource requirements;
- other, *nested* transactions.

An *activity* is any stage in a transaction with static requirements for one or more shared resources. This definition covers all types of resources that typically make up a distributed real-time system, both logical (code, data) and physical (processor, communications).

Each transaction is thereby described by a graph that is acyclic, directed and nested. The nodes of the graph represent activities and the edges capture precedence and nesting relationships.

The transaction model is nested in order to support evolution of application components as the system is developed. In order that the timing model may be executed at any stage of that evolution, the main objects of the computational model - the nested transaction and the activity - must be interchangeable. Hence, any nested transaction can be implemented in terms of activities, but also an activity can be replaced by a nested transaction and then evolved further. This has led to highly scaleable timing model. The parameters via which timing behaviour is represented and observed are the same for a single activity, any group of related activities, a nested transaction and a system-level transaction. These parameters are the input jitter, output jitter and minimum I/O separation, as illustrated in Figure 2. The parameters are respectively denoted by $J^{in}_{i,..,k}$, $J^{out}_{i,..,k}$ and $d_{i,..,k}$ for

any arbitrarily nested transaction or activity, $\lambda_{i,..,k}$ with some system-level transaction λ_i.

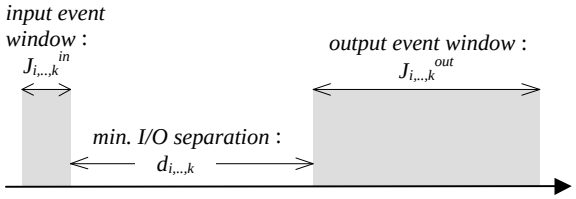

Figure 2. Transaction/Activity Timing Properties

Timing analysis can be performed at any stage of evolution of the transaction graph provided that all nested transactions terminate with at least one activity, *ie.* have defined resource consumption characteristics. All activity resource requirements are denoted by the tuple ($c_{i,..,k}$, $C_{i,..,k}$, $V_{i,..,k}$), where [*c*, *C*] bounds the resource access time and *V* is an assigned *reservation budget* which can take any real value in the interval (0, 1]. *V* governs the rate at which progress is made at run-time for that activity.

The same parameters are used to represent the budgeted timing properties early in the system development through to the actual values at the end of the implementation stage.

4. Calculation of End-to-End Delays

At each stage of evolution of the system, its overall timing behaviour can be assessed by a depth-first traversal of the set of nested transaction graphs, observing the order of precedence at each nested level. As the traversal progresses, local delays and jitter are consolidated to generate cumulative end-to-end values.

At *any* level of nesting, *ie.* for a single activity or an entire nested transaction, the basic timing parameters $J_{i,..,k}^{in}$, $J_{i,..,k}^{iout}$ and $d_{i,..,k}$ are related through the overall delays experienced at that level of nesting :

$$d_{i,..,k} = r_{i,..,k} - J_{i,..,k}^{in}$$

$$J_{i,..,k}^{out} = J_{i,..,k}^{in} + (R_{i,..,k} - r_{i,..,k})$$

where $r_{i,..,k}$ and $R_{i,..,k}$, are the minimum and maximum delays accrued across that level of nesting. These accrued delays (and consequently the jitter) are calculated by expanding the sub-graph for the nested transaction involved and accounting for localised delays, precedence relationships and nesting relationships.

For scaleability, the analytical model is constructed such that cumulative, end-to-end delays can be calculated *relatively*, from any one node in the transaction graph to any other. Let $d_{i,..,k}^{in}$ denote the cumulative delay that a nested transaction or activity $\lambda_{i,..,k}$ inherits as it arrives for execution. Depending on the position of $\lambda_{i,..,k}$ in the transaction graph, this inheritance may be from its predecessor(s) or a parent (higher level nested transaction), denoted by $\lambda_{i,..,j}$. In turn, let $d_{i,..,k}^{out}$ denote the cumulative delay that $\lambda_{i,..,k}$ exports to its successors upon completion. Figure 3 illustrates the relationships between these parameters.

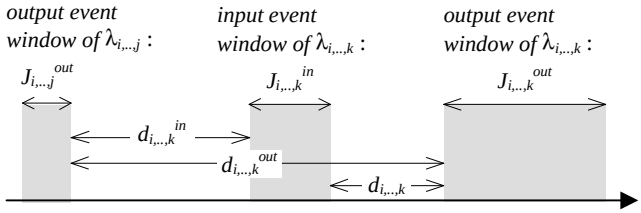

Figure 3. Cumulative Delays and Jitter

Localised delays are those that are attributable to individual activities as they consume shared resources during their execution. Such delays occur at the deepest level of nesting of the transaction graph and can be calculated directly from the specified activity timing characteristics. A simple abstract model of the target hardware implementation and run-time scheduling solution is imposed in order that these delays can be estimated – see Section 5. Localised delays are consolidated into the cumulative delay calculation by increasing the inherited minimum I/O separation by the minimum local delay (the impact of the maximum local delay is taken care of by the output jitter calculation given earlier) :

$$d_{i,..,k}^{out} = d_{i,..,k}^{in} + r_{i,..,k}$$

Precedence and nesting are accounted for in a manner that depends on the form of relationship involved. In the following circumstances, cumulative delays and jitter are directly inherited (unchanged) across the relationship in the direction of control flow :

- *Precedence* - From the predecessor output to successor input(s) across *one-to-one* and *one-to-many* precedence relationships (where completion of a given stage of computation triggers the arrival of one or more subsequent stages);
- *Nesting* - From parent input to child input(s) across *descending* nesting relationships (where a single input event window at the parent level is decomposed into one or more concurrent input events at the child level).

In the following circumstances, however, delay and jitter inheritance is less straight forward :

- *Precedence* - From predecessor output(s) to the successor input across *many-to-one* precedence relationships (where completion of a number of concurrent stages of computation is required to trigger the arrival of a single subsequent stage);
- *Nesting* - From child output(s) to the parent output across *ascending* nesting relationships (where a

number of concurrent output events at the child level are composed into a single output event window at the parent level).

These latter situations can be modeled in the same way, both involving the consolidation of multiple concurrent threads of control, $\{\lambda_{i,..,j} ; j = 1,..,n\}$, into a single thread, $\lambda_{i,..,k}$ whose release is governed by a boolean parameter $Q_{i,..,k}$. The value of $Q_{i,..,k}$ is dependent on the execution status predecessor activities. The only difference between the 'precedence' case and the 'nesting' case being the relative positions of $\lambda_{i,..,j}$ and $\lambda_{i,..,k}$ in the transaction graph - in the first case, they are at the same nested level and in the second case, they are at successive ascending levels. The following expressions give 'safe' bounds independently of the definition of $Q_{i,..,k}$ (whose details can be used to give less pessimistic bounds) :

$$d^{in}_{i,..,k} \geq \min_j(d^{out}_{i,..,j})$$

$$J^{in}_{i,..,k} \leq \max_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k}$$

Similarly, it is possible to calculate the tightest possible bounds that can only be relaxed given $Q_{i,..,k}$:

$$d^{in}_{i,..,k} \leq \max_j(d^{out}_{i,..,j})$$

$$J^{in}_{i,..,k} \geq \min_j(d^{out}_{i,..,j} + J^{out}_{i,..,j}) - d^{in}_{i,..,k}$$

This is the basic analytical framework for end-to-end timing analysis but two further details are required :

- how to calculate localised, activity-level delays in a target-independent way - this is covered in Section 5;
- how to impose constraints on the degree of end-to-end delay variation (jitter) - this is discussed (very briefly, due to space limitations) below, and illustrated by example in Section 7.

The RBA transaction model is basically event-triggered. Whilst this allows a great deal of flexibility at the run-time scheduling stage, the net effect at the application level can be a degree of variability in its end-to-end timing behaviour. This is perfectly acceptable in many applications but, in some, some means of limiting the amount of variation is required. For example, an output to a sensor or a control loop feedback may need to occur in a strictly periodic manner.

Rather than supporting a pure time-triggered transaction model for such applications, to virtually eliminate input jitter at successive stages of the transaction (at the price of reduced scheduling flexibility, even for the majority of other applications in the system that can tolerate jitter), RBA allows input jitter *tolerances* to be prescribed at any stage in a transaction. Hence, the maximum input jitter associated with any nested transaction or activity may be either inherited directly from its predecessors or statically imposed by the application developer.

Maximum values for inherited jitter can be predicted by the RBA model as described above. If these values are beyond acceptable limits, statically imposed values may be specified to over-ride inherited values. These values are then fixed at the appropriate node(s) of the transaction graph. Hence, a purely time-triggered transaction can be produced by specifying the input jitter values to be zero (or negative, to indicate a forced delay offset) throughout the transaction. Similarly, a purely event-triggered transaction can be produced by allowing all input jitter to be inherited. More generally, a hybrid time-and-event-triggered transaction can be defined by specification of appropriate jitter tolerance levels. This allows trade-offs to be made on a per-system basis between its overall timing behaviour and the degree of flexibility available at the integration stage of its development.

5. Calculating Localised Delays based on Scheduling Obligations

Prior to implementation, an abstract model of the target hardware is used to determine the localised delay bounds for each activity at the deepest level of nesting in the transaction graph. The abstract model is based on the notion of *scheduling obligations* :

- Each activity is obliged to make progress at least at the minimum rate specified by its reservation budget;
- The 'granularity' of the underlying resource reservation mechanism is bounded by some value Δ.

The following delay bounds can then be derived independently from the target implementation :

$$r_{i,..,k} = \left\lceil \frac{c_{i,..,k}}{V_{i,..,k}} \right\rceil$$

$$R_{i,..,k} = \left\lceil \frac{C_{i,..,k}}{V_{i,..,k}} \right\rceil + \Delta$$

When the target hardware details are finalised, the abstract timing analysis results can be validated. For each activity, $\lambda_{i,..,k}$, the actual service time (the best and worst-case execution time or equivalent communication resource requirements) must be bounded by the previously budgeted values $[c_{i,..,k}, C_{i,..,k}]$. There is no need to re-execute the end-to-end timing analysis calculation to check that the pre-computed end-to-end delay and jitter bounds will hold. Hence, it is simple to *re*-validate end-to-end timing guarantees following localised changes to the target hardware or software implementation.

The simplest run-time implementations that meet the RBA scheduling obligations are weighted round-robin or weighted fair queuing scheduling/communication schemes [4, 5, 14, 15] - these can provide suitably fine-grained bandwidth allocation. Approaches such as fixed-priority scheduling can also be adapted by the use of bandwidth server techniques [11, 12]. The context for our

research is aircraft IMA systems [1, 2, 3] and Section 7 shows how the civil IMA scheduling model can be configured to support RBA.

6. Examples of Transaction Definition and End-to-End Timing Analysis

Consider the following simple transaction taken from an avionic sub-system case study - a generic *attitude guidance and autopilot* (AGA) missile sub-system. In short, the AGA sub-system reads data from sensors, calculates the attitude of the missile and then, together with information about a potential target, such as an infra-red image, determines an aim point for the missile. Figure 4 illustrates the AGA transaction topology in the form of an acyclic activity graph.

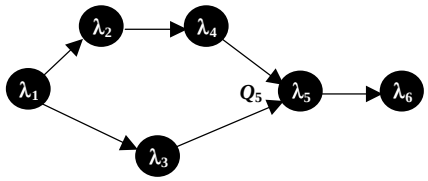

Figure 4. Example Transaction Topology

The release function Q_5 is defined such that λ_3 and λ_4 must both complete execution prior to release of λ_5.

The following table gives the local delay bounds [*r*, *R*] for each activity, calculated from the RBA model from a given set of resource requirements (*c*, *C*, *V*).

i	λ_i	c_i	C_i	V_i	r_i	R_I
1	awaitTrig	0	1	0.25	0	4
2	readSensor	1	2	0.2	5	10
3	readTarget	2	3	0.2	10	15
4	calcAtt	4	7	0.3	13	24
5	calcAim	3	8	0.4	7	20
6	write	2	3	0.25	8	12

For a given λ_1 input jitter value of, say, 4, the end-to-end delays are predicted to be as follows :

i	λ_i	J_i^{in}	d_i^{out}	J_i^{out}
1	awaitTrig	4	-4	8
2	readSensor	8	1	13
3	readTarget	8	6	13
4	calcAtt	13	14	24
5	calcAim	24	21	37
6	write	37	29	41

Notice from the table that λ_1 has a negative I/O separation value - this is perfectly valid, corresponding to the fact that its earliest possible completion time is earlier than its latest possible arrival time.

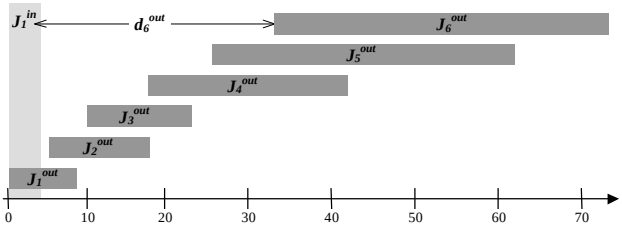

Figure 5. Example End-to-End Analysis Results

Figure 5 illustrates the analysis results graphically. The overall end-to-end behaviour of the transaction is captured by the tuple (J_1^{in}, d_6^{out}, J_6^{out}). The results also show how delays and jitter accumulate at intermediate stages in the transaction. This information is useful when the predicted end-to-end delay or jitter is unacceptable and intermediate parameters need to be modified to give acceptable timing behaviour. An example of this is given in Section 7 in the context of reducing end-to-end jitter.

7. A Target-Specific Implementation of RBA for Civil IMA Systems

In this section, we address the transition from the abstract RBA model to a target-specific implementation model. We consider the case of civil IMA systems, where the run-time CPU scheduling model is prescribed by the APEX standard [3]. APEX defines the following two-tier scheduling policy :

- *Partition* scheduling - CPU time on each processor is allocated cyclically to create multiple temporal partitions;
- *Process* scheduling - A partition can contain multiple concurrent processes which are scheduled according to a fixed-priority regime within the time allocated to that partition.

For the duration of its execution, a partition is granted exclusive access to all resources shared with other partitions. Communication between partitions is handled via OS API calls, giving a degree of independence from the underlying network topology and from the allocation of partitions to processors. Within a partition, additional 'logical' communication resources may be shared between processes.

The RBA model can be mapped most readily to the APEX model by assuming temporal equivalence between an RBA activity and an APEX partition. It then remains to derive the partition level attributes of the APEX run-time schedule from the RBA timing model at system integration time. There are many potential mappings from the RBA domain into the APEX domain as will now be illustrated by example.

Consider the AGA sub-system example used above. Figure 6 depicts a resource allocation for parts of the

example transaction, reflecting that other parts might be allocated to other resources in a distributed IMA system. Assume that R1 represents a CPU resource that is scheduled according to the APEX scheduling model and that R2 and R3 are synchronous, logical communication resources (typically mapped to shared memory) that provide a means for activities λ_2 and λ_3, respectively, to read their required sensor data. R2 and R3 are termed 'synchronous' since they rely on R1 (the CPU) to also be available to enable their use.

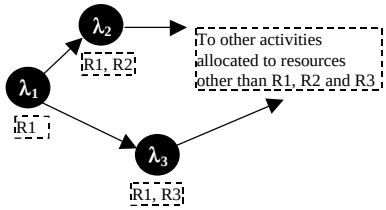

Figure 6. Example Resource Allocation

Three alternative APEX-compliant schedules for processing resource R1 are described below. A number of trade-offs involved in the production of the final scheduling solution are highlighted.

7.1 APEX Schedule 1

Figure 7 depicts possibly the simplest solution to meet the stated timing requirements. The APEX cycle time is set to 4 (RBA) time units – note that a finer grained clock may be used in practice to implement the schedule. Each activity λ_1, λ_2 and λ_3 is then assigned one time slot in each cycle to either meet exactly or exceed its minimum bandwidth requirement.

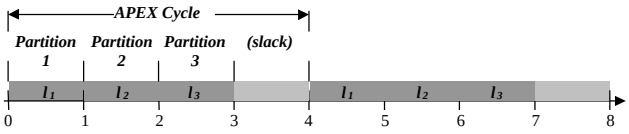

Figure 7. APEX Schedule 1

There are several means of assessing this APEX schedule, including the following :

- The effectiveness of the final resource allocation, measured in terms of the amount of reserved bandwidth compared to its actual usage under worst-case assumptions about activity execution;
- The extent to which additional run-time controls must be imposed to preserve the results of the RBA predictive model (typically the enforcement of minimum end-to-end delays and thereby the maximum jitter accrued).

The issue of resource allocation effectiveness is one that concerns all 'bandwidth reservation' scheduling solutions. Consider APEX schedule 1 :

- λ_1 is allocated exactly the minimum bandwidth required to meet its worst-case requirements, *ie.* 25% of the bandwidth;
- λ_2 is allocated more than the minimum bandwidth required to meet its worst-case requirements, *ie.* 25% of the bandwidth rather than 20%;
- λ_3 is allocated more than the minimum bandwidth required to meet its worst-case requirements, *ie.* 25% of the bandwidth rather than 20%.

The reason that two activities have been over-allocated resource bandwidth is that a compromise has been made in favour of defining a simple APEX schedule. This has resulted in an additional 10% of the total resource bandwidth being reserved over and above that which is actually required in the worst-case. Remember, however, that the benefits of the RBA solution have been retained in terms of the ability to incrementally modify and analyse the distributed IMA system; which may be sufficient to justify the final APEX schedule. Alternatively, a different APEX schedule could be defined, such as either of those described below, that comply with the RBA model and give better resource utilisation, at the expense of a slightly more complex APEX schedule definition.

The second issue identified above - the extent to which additional run-time controls must be imposed to preserve the results of the RBA predictive model - is now considered for this first APEX scheduling solution. The need for such controls is typically to enforce minimum end-to-end delay requirements and thereby limit the maximum jitter accrued. In the example, the bandwidth allocated to activities λ_2 and λ_3 is greater than that required to meet worst-case requirements and so their actual best-case delays will be shorter than those predicted by the RBA model. This is shown in the table below, alongside the predicted RBA values denoted by V_i^* and r_i^* :

i	λ_i	c_i	V_i	V_i^*	r_i	r_i^*
1	awaitTrig	0	0.25	0.25	0	0
2	readSensor	1	0.25	0.2	4	5
3	readTarget	2	0.25	0.2	8	10

The following table shows the resulting impact on end-to-end delays, alongside the previously predicted RBA values denoted by d_i^{out*} and J_i^{out*} :

i	λ_i	J_i^{in}	d_i^{out}	d_i^{out*}	J_i^{out}	J_i^{out*}
1	awaitTrig	4	-4	-4	8	8
2	readSensor	8	0	1	14	13
3	readTarget	8	4	6	15	13

To counteract this effect without resorting to an alternative APEX schedule definition, additional controls can be imposed as follows :

- The earliest completion time of the affected activities (λ_2 and λ_3) could be artificially delayed;
- The earliest release times (and hence the input jitter) of all direct successor activities (in this case, λ_4 and λ_5) could be controlled as described below.

By either of these means, the predicted input jitter values can be preserved for the direct successors to λ_2 and λ_3 and, hence, for all other 'downstream' activities. The second of the two approaches is chosen now for illustration purposes. The next table gives the inherited input jitter values for the direct successors to λ_2 and λ_3 and the values that should be used to over-ride these in order to comply with the predictive RBA model. The over-ride values are simply those that were previously predicted by the RBA model (denoted by J_i^{in*}).

i	λ_i	J_i^{in}	J_i^{in*}	d_i^{out}	J_i^{out}
4	calcAtt	14	13	14	24
5	calcAim	24	24	21	37
6	write	37	-	29	41

Notice that, due to the particular topology and timing characteristics of the transaction, the input jitter control for activity λ_5 is actually redundant, being equal to the naturally inherited value. Hence, from the point of release of λ_4, there is no observable impact on the timing behaviour of the transaction.

Note that the impact of reserving more than the required minimum bandwidth on the worst-case delay prediction of the RBA model is not to invalidate it, *ie.* it is still 'safe', but only to increase its pessimism. Hence, the imposition of input jitter controls to counter the effects on best-case delays is sufficient to preserve the integrity of the RBA model in these situations.

7.2 APEX Schedule 2

Figure 8 depicts an alternative APEX schedule, where the cycle time is set to 20 (RBA) time units and with each activity λ_1, λ_2 and λ_3 assigned multiple time slots in each cycle to either meet exactly or exceed its minimum bandwidth requirement.

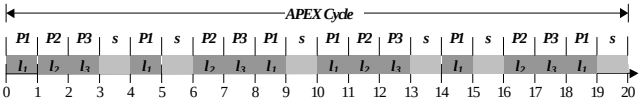

Figure 8. APEX Schedule 2

Considering this schedule from the perspective of resource utilisation effectiveness :

- λ_1 is allocated more than its minimum bandwidth requirement, *ie.* 30% rather than 25%;
- λ_2 is allocated exactly its minimum bandwidth requirement, *ie.* 20%;
- λ_3 is allocated exactly its minimum bandwidth requirement, *ie.* 20%.

Hence, 5% of the bandwidth of resource R1 has been allocated over and above that which is actually necessary to meet the stated worst-case resource requirements of the activities. This over-allocation is still 5% less than for APEX schedule 1, as described previously, but then APEX schedule 2 is less simple.

The impact of reserving more than the minimum bandwidth required for activity λ_1 is to reduce its best-case response time below that predicted by the RBA model. Once again, this is easily compensated for by the imposition of jitter controls on the direct successors to the activity, *ie.* λ_2 and λ_3.

7.3 APEX Schedule 3

The granularity of the clock that provides a time reference for a given system resource need not be the same as that used in the abstract RBA model, so long as the actual clock is finer-grained than the RBA clock. The schedule shown in Figure 9 exploits this fact to give a perfect implementation of the RBA model for resource R1 at the expense of imposing an additional requirement on the APEX clock to be (at least) ten times finer-grained than the RBA clock.

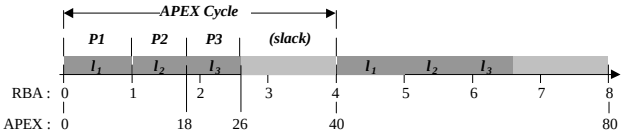

Figure 9. APEX Schedule 3

The enforcement of a minimum clock granularity is not unreasonable. The net effect is that the RBA model can be implemented exactly without any additional controls, *eg.* to enforce minimum delays. This is beneficial in terms of supporting future changes to the system. In practice, it is likely that the schedule can be defined without the need for this clock granularity requirement since the transaction timing requirements will typically be much greater than the clock cycle time.

8. Related Work

It was stated earlier that there are many and varied forms of valid implementation schemes for RBA, as described in the literature on the class of reservation-based scheduling solutions [4, 5, 11, 12, 14, 15]. In this section, the focus is on other studies that have specifically addressed the issue of timing analysis at multiple stages in the life-cycle of a distributed real-time system.

Sha and Sathaye [13] have used the notion of *scheduling abstractions* as a mechanism for providing a degree of target-independence into the timing analysis

model of a distributed system. However, their approach is restricted to time-triggered transactions and static, priority-based scheduling.

Stoyenko *et al* [6] have developed a language, CaRT-Spec for describing the timing and resource characteristics of complex, distributed real-time systems. Applications are described in terms of concurrent processes, each of which is described by a single entry, single exit acyclic graph of *actions*. Actions capture the resource consumption characteristics of the applications and can be structured hierarchically to reflect ongoing refinement during the development of the system. There are clear similarities between CaRT-Spec and RBA but some notable differences are apparent. Firstly, CaRT-Spec allows the application developer to model resource usage explicitly, providing a powerful description language to do so. RBA uses a consistent bandwidth-based model of resources that supports independent timing analysis of the RBA activities assigned to each resource - this encourages the application design to be kept free from implementation details and integration decisions and also ensures that a holistic timing analysis model is avoided. Secondly, CaRT-Spec provides a general assessment of feasibility based on symbolic execution. RBA provides an analytical evaluation of localised and end-to-end delays and jitter.

Burns and Lister [7] have described the TARDIS framework for top-down development of timely and reliable distributed systems. The main similarity between TARDIS and RBA is in the clear distinction between the logical and physical levels of the architecture with implementation-independent timing obligations being imposed from the former to the latter, although the TARDIS logical architecture is described using an object-oriented approach. Unlike RBA, TARDIS assumes a global time-base is available to coordinate scheduling policies across the distributed platform - the timing analysis is application-dependent but can be holistic.

9. Summary

A practical problem of how to apply timing analysis in a cost-effective way in the development of large-scale industrial real-time systems has been identified. The analysis should be applicable throughout the life-cycle of the system, starting from its early stages of development. The analysis should be insensitive to localised changes in system requirements, design or implementation details.

RBA has been proposed as a potential solution to this problem. RBA provides a consistent model for capturing the end-to-end timing and resource requirements of a distributed system and performing timing analysis based on this information. Timing analysis can not only be applied to individual end-to-end transactions in the system but also to specific stages within a transaction to calculate relative delays and jitter. Localised changes to the system can thus be re-validated incrementally, *ie*. with minimal re-analysis of other parts of the system. This supports the evolvable nature of large-scale, distributed real-time system development.

10. References

[1] "ARINC 651 : Design Guidance for Integrated Modular Avionics" *Airlines Electronic Engineering Committee*, 1991.

[2] Edwards, RA., "ASAAC Phase 1 Harmonised Concept Summary" *Proc. of ERA Avionics Conf. and Exhibition*, 1994.

[3] "ARINC 653 : Avionics Application Software Standard Interface" *Airlines Electronic Engineering Committee*, 1997.

[4] Parekh, AK., Gallager, RG., "A Generalised Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case" *Proc. of IEEE INFOCOM*, 1992.

[5] Kuo, TW., Yang, WR., Lin, KJ., "EGPS: A Class of Real-time Scheduling Algorithms Based on Processor Sharing" *Proc. of Euromicro Workshop on Real-time Systems*, 1998.

[6] Stoyenko, AD., Marlowe, TJ., Laplante, PA., "A Description Language for Engineering Complex Real-time Systems" *Real-time Systems Jnl.* 11(3), 1996.

[7] Burns, A., Lister, AM., "A Framework for Building Dependable Systems" *Computer Jnl.* 34(pp.173-181), 1991.

[8] Stoica, I., *et al*, "A Proportional Share Resource Allocation Algorithm for Real-time Time-shared Systems" *Proc. of Real-time Systems Symp.*, 1996.

[9] Jeffay, K., Bennett, D., "A Rate-based Execution Abstraction for Multimedia Computing" *Proc. of Int. Workshop on Network and OS Support for Digital Audio and Video*, 1995.

[10] Yau, DKY., Lam, SS., "Adaptive Rate Controlled Scheduling for Multimedia Applications" *Proc. of ACM Multimedia Conf.*, 1996.

[11] Deng, Z, Liu, JW-S., "Scheduling Real-time Applications in an Open Environment" *Proc. of Real-time Systems Symp.*, 1997.

[12] Spuri, M., Buttazzo, G., "Scheduling Aperiodic Tasks in Dynamic Priority Systems" *Jnl. of Real-time Systems* 10(pp179-210), 1996.

[13] Sha, L., Sathaye, SS., "A Systematic Approach to Designing Distributed Real-time Systems" *IEEE Computer Jnl.* 26(9), 1993.

[14] Zhang, L., Deering, S., Estrin, D., Shenker, S., Zappala, D., "RSVP: A New Resource Reservation Protocol" *IEEE Network* 7(5), 1993.

[15] Raha, A., Kamat, S., Zhao, W., "Guaranteeing End-to-End Deadlines in ATM Networks" *Proc. of IEEE Distributed Computing Systems Conf.*, 1995.

[16] Ermedahl, A., Hansson, H., Sjodin, M., "Response Time Guarantees in ATM Networks" *Proc. of Real-time Systems Symp.*, 1997.

[17] "Defence Standard 00-56: Safety Management Requirements for Defence Systems" *UK Ministry of Defence*, 1996.

[18] "DO-178B: Software Considerations in Airborne Systems and Equipment Certification" *RTCA/EUROCAE*, 1992.