TOWARDS A SCHEDULING AND TIMING ANALYSIS SOLUTION FOR INTEGRATED MODULAR AVIONIC SYSTEMS

Alan Grigg & Neil C. Audsley
BAe. Dependable Computing Systems Centre,
Dept. of Computer Science, University of York, UK

Integrated modular avionics (IMA) [1,2] *aims to provide highly flexible, reliable and integrated solutions for future aircraft systems which can readily exploit new advances in processor and networking technologies. In this paper, we discuss the problems involved in the provision of appropriate scheduling and timing analysis techniques for IMA systems and suggest a potential solution.*

1. INTRODUCTION

Achievement of the full potential of IMA presents a number of major challenges to system designers. In this paper we focus upon the problems in the ***temporal domain***, particularly the ability to analyse the temporal properties of an IMA system using ***timing analysis***.

Timing analysis is a collection of mathematically based techniques that enable a system, together with its software, to be analysed off-line ***without the need for system execution***. The result of applying the analysis, is a set of ***guarantees*** regarding the worst-case behaviour of the system – under normal system operation, these worst-case guarantees will be met at run-time[*]. The guarantees include the worst-case execution times of software, worst-case communications latencies and worst-case resource usage, *eg.* memory. These calculated guarantees can be compared with the timing and resource usage requirements of the system to see if they will be met at run-time.

The advantages of applying timing analysis techniques include:

- Cost savings due to *automated* timing analysis – c.f. current *manual* timing analysis.
- Worst-case timing behaviour calculated off-line.
- Worst-case resource usage (*ie.* memory) calculated off-line.
- Sensitivity analysis – "*what-if*" games can be played with different architectures, software allocations etc., *without having to construct the system.*
- High-level timing analysis – prior to any or all code being produced, timing analysis can predict whether timing and resource usage requirements will be met when the system is complete (when given system architecture, software architecture, time and resource budgets of application software).
- Important analytical evidence to support certification.

Timing analysis for simplistic systems is proven, and has been used in certified safety-critical systems. For IMA systems, timing analysis is more problematic, due to characteristics of IMA:

- *Distribution* - Applications with end-to-end timing requirements, *ie.* from sensor, across processing elements via communications media, to actuator.
- *Integration* - High degree of integration, both functional (shared data) and physical (shared processors, communication media, I/O devices);
- *Module interchangeability* - Insensitivity to design-time and in-service modifications, including first-line interchangeability of hardware modules with common specifications.

[*] Guarantees may not be met if (a) assumptions made regarding system behaviour are incorrect; (b) faults occur at run-time– although certain faulty behaviours may be factored into the timing analysis.

In this paper, we discuss the concept of timing analysis for IMA systems. After an introduction to timing analysis, we focus upon the characteristics of IMA systems which provide distinct problems to timing analysis techniques – considering why existing implementation techniques, such as cyclic executives, are not appropriate for IMA. Importantly, we then outline potential solutions for IMA specific problems, culminating in a modular timing analysis approach for IMA, together with an implementation strategy.

2. PRINCIPLES OF TIMING ANALYSIS IN SAFETY-CRITICAL SYSTEMS

Meeting the timing and resource usage requirements imposed upon an avionics computing system involves many aspects of the software lifecycle. Fundamentally, it is the behaviour of the system at run-time which determines whether these requirements will be met. However, practitioners are required to demonstrate to a certification authority that timing and resource usage requirements will be met at run-time *prior* to system operation.

Traditionally, this has been achieved by a combination of:

- Limited off-line analysis of timing properties of code, by *hand*.
- Exhaustive testing of the system, by exposing the system to as many combinations of input as possible, hoping to find any that lead to violation of timing requirements.

The risk with these approaches is that unless testing is complete, there may be (untested) situations that give rise to timing requirements not being met at run-time. This risk becomes greater as systems become more complex, especially when distributed systems are considered.

Alternatively, off-line ***timing analysis*** can determine whether timing requirements will be met at run-time. Rigorous mathematical analysis techniques are employed that require execution of the system software - hence can be performed without expensive system build. Indeed, timing analysis can be applied at many stages of the system life-cycle.

The remainder of this section outlines the basic concepts behind timing analysis, examining potential timing requirements, basic timing analysis principles, accuracy of timing analysis and high-level timing analysis.

2.1 Basic Assumptions

To enable off-line timing analysis, certain characteristics of the system must be known:

- *System Architecture* - This describes the physical topology of the system.
- *Software Architecture* - This describes the resource requirements of the software. Included is processor time, communications (by network, shared memory etc.), and other physical resources (*eg.* sensors).
- *Computational Model* - The restrictions placed upon the behavioural characteristics of the application software. Here we consider programming models, such as asynchronous computation, bounded computations (in terms of time and space). Typically, the computational model is defined by the coding standards used for an application, together with the programming language used (*eg.* Ada95[15] or the SPARK Ada95 safety-critical subset[16]).
- *Scheduling Policies* - The policy by which time is allocated to competing application tasks. Where resources are shared, arbitration over resource access must occur via an access protocol or scheduling policy. The access protocol may be embedded in hardware (*eg.* communications controllers) or implemented in software.

For this paper, we assume software is constructed as a number of processes, communicating via physical (communications network) and logical (shared memory) resources. Processes may be either periodic (cyclic) or aperiodic (where a minimum inter-arrival time is known). Processes may be structured into transactions, where one process cannot start until the previous process in the transaction has completed. The hardware architecture assumed is one of multi-processor nodes connected via some network media.

2.2 Timing Requirements

The timing requirements that may be placed upon an application, and can be checked by offline timing analysis, include:

- *Computational* - Each application process may have a deadline which must be met under worst-case conditions.
- *Jitter* - Processes may have requirements on precisely when inputs are read and outputs emitted:
  - *release jitter* - given an event occurrence, release jitter is the difference between the earliest and latest release times of the process. The release time is when the event is recognised by the system and the appropriate process invoked, *ie.* made runnable.
  - *input jitter* - at or after an event occurs, any data that is to be read by the subsequently invoked process has an interval within which it is valid. The input jitter is the difference between the time at which the data becomes valid and the latest time at which it remains valid. Input jitter is sometimes called *input validity*.
  - *output jitter* - when a data value is emitted by a process, output jitter represents the variation in time at which the value is emitted. Thus, output jitter is the difference between the earliest time the value can be emitted and the latest time it can be emitted. The latter is no later than the deadline of the process, by which time it must have completed execution.
- *Shared Resources* - Where resources are shared, arbitration over resource access must occur via an access protocol. The access protocol may be embedded in hardware (*eg.* concurrent asynchronous memory access) or software (*eg.* protocols embedded in the scheduler). However, there are two fundamental timing requirements for shared resources (both of which are usually provided by the access protocol):
  - deadlock cannot occur;
  - the time that a process must wait to access a resource must be bounded.
- *Mode Changes* - The system may execute in several modes, each (potentially) containing different process sets with different timing requirements. There are requirements to ensure that only the correct processes execute in a mode and that the transition between modes occurs without violating timing requirements.
- *Communications* - Where data is passed from one process to a remote process across a communications media, the time taken for the communication must be bounded.
- *End-to-end* - Any timing requirements upon the end-to-end deadline across a transaction of processes (possibly on different processors) must be met.
- *Multiple-criticality interference* - Processes in lower criticality levels must not affect the temporal properties of those in higher criticality levels.

2.3 Basic Timing Analysis

Basic timing analysis can be abstracted into distinct levels:

2.3.1 Level 0 – Hardware

At this level, timing analysis considers the basic hardware operations that need to be quantified, *eg.* the length of instructions on a processor; the physical transmission time across a communication medium. Pertinent behavioural characteristics of hardware are also noted, *eg.* whether instructions on a processor are atomic, processor pipelines/caches.

2.3.2 Level 1 – Basic Blocks

A software component can be broken into basic blocks having a single entry/exit point, with no looping, branching or calls – thus they contain no requirement of resources other than processor. At this level, timing analysis considers the processor time requirements of a basic block, *eg.* the worst-case execution time is calculated by summing the worst-case execution times of component instructions.

2.3.3 Level 2 – Process

The basic blocks identified in Level 1 may be composed into application processes by use of looping, branching and call instructions around the basic blocks. At this level, timing analysis considers the resource requirements of a process. The worst-case execution time of a process is calculated by examining the control-flow paths in the process, noting the worst-case execution times of basic blocks along the path. Also, we note other resources required by a process.

2.3.4 Level 3 – Processor

At this level, timing analysis acknowledges the (possible) presence of multiple processes upon a single processor, which inevitably compete for resources (*eg.* processor time). When multiple processes are concerned, we must take account of the scheduling of processes within the timing analysis. Hence, the behaviour of the scheduling policy is embedded within the analysis. This is the primary reason that a timing analysis is applicable to a specific scheduling policy. Also, that scheduling policy must be used at run-time, otherwise, the timing analysis performed off-line is meaningless.

When considering the timing analysis of process A in the presence of competing processes, we must include the effect of those processes being scheduled instead of process A. Essentially, process A is prevented from executing by:

- *interference*: normal execution of other processes;
- *blocking*: whilst process A is blocked on access to resource B, blocking includes all execution of other processes required to free resource B.

To enable timing analysis of process A, both the interference and blocking it may suffer must be bounded: we need to calculate the worst-case interference and worst-case blocking. These are a function of the specific scheduling policies used for the processor and other resources.

2.3.5 Level 4 - System

Where multiple processors are introduced, timing analysis becomes more complex - it now has to consider genuine concurrency between processes (on distinct processors). It consists of two main phases:

- allocation of processes and shared logical resources to processors;
- timing analysis of the system, given an allocation.

Timing analysis cannot be carried out until allocation has occurred, since this affects the communication times for interacting processes.

At this level, timing analysis considers the communications between processes on *different* processors. This requires analysis of the communications media involved, *ie.* calculating the worst-case message transmission time across the media, assuming worst-case message load inflicted on the media by the processes on ***all*** processors in the system.

2.4 High-Level Timing Analysis

The previous section suggests that timing analysis is layered, higher levels being dependent upon lower levels – an obvious conclusion being that all software and hardware has to be known before timing analysis can occur. However, high level timing analysis can be performed using *estimated resource budgets* for the resources expected to be required by the software, *ie.* high level timing analysis occurs disregarding levels 0 and 1. This allows *sensitivity analysis* of the system, that is allows playing of "*what-if*" games to estimate the effects of:

- a particular software configuration;
- a particular software configuration change.

When specific resource requirements are known, they are used instead of the estimated budgets.

3. TIMING ISSUES IN IMA SYSTEMS DESIGN AND ANALYSIS

From a timing perspective, IMA systems are characterised by a number of fundamental problems which must be addressed during their design and analysis.

Distributed 'applications' with end-to-end timing requirements :

System level timing requirements must, in general, be mapped onto a diverse set of software activities executing on a set of distributed resources (processing modules, communication media *etc.*). A key consideration during design is the degree of flexibility in the set of *intermediate* timing requirements which result from decomposition of the original end-to-end requirements. This has implications for the choice of scheduling solution and, in turn, limits the extent to which the final system is temporally flexible, *ie.* sensitive to changes in timing requirements, system design or run-time environment.

One measure of flexibility of the system is *changeability*, that is the ability to:

- support limited modifications to any one or a number of identified transactions without affecting the timing guarantees provided for others;
- support addition or removal of complete transactions (within resource limitations).

Functional and physical integration :

In a distributed system with a high degree of integration (logical and physical resource sharing), timing *circularities* will exist unless specific measures have been taken to avoid these. The net result is that scheduling and timing analysis of the system is *holistic*, *ie.* the problem is not partitionable.

Timing circularities arise whenever inherent variability in the timing behaviour of any one process is propagated system-wide through logical and physical inter-dependencies with other processes – either direct or indirect (via shared resources). A simple example of this is depicted in Figure 1, where two transactions of processes traverse the same two physical processors but in opposite directions. Logical dependencies such as precedence constraints between the processes introduce a dependency between the response time of a process and the release time of its successor(s). Physical dependencies arise due to response times of activities which share a physical resource being mutually dependent due to *interference* delays associated with resource contention. The latter effect typically increases in significance with flexibility of the scheduling solution, *eg.* with the use of pre-emption. The avoidance of timing circularities should be a key objective in the development of flexible scheduling solutions for IMA systems.

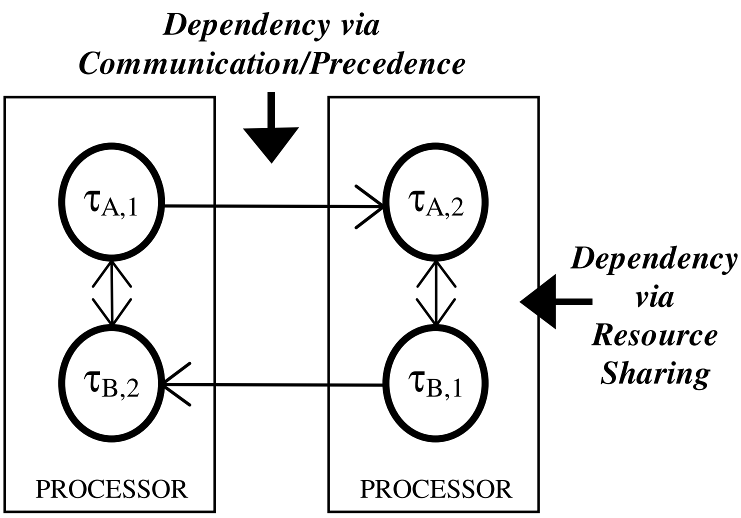

Figure 1 - Holistic Analysis Due to Timing Circularities

A further consideration attributable to the integrated nature of an IMA system is how to provide co-ordination between the multiple scheduling mechanisms in the system, each of which operates over a specific resource or set of resources with its own notion of time. Ideally, these mechanisms should be co-ordinated in such a way as to achieve effective, parallel use of resources whilst supporting the provision of end-to-end timing guarantees. An underlying issue here is the degree to which the clocks associated with different scheduling mechanisms require synchronisation, if at all, *ie.* whether relative clock drift must be bounded and, if so, the precise extent to which it can be tolerated. Consistency of any global state information must also be addressed since delays inherent in communications between processing nodes can lead to uncertainty in any single node's perception about the state of the system as a whole at any given time.

Insensitivity to design-time and in-service modifications :

There are a number of issues relating to the provision of flexible system designs/implementations which can readily support modification - such as the use of VBI technologies for code portability[3]. In this paper, we are particularly concerned with scheduling and timing analysis.

The need to support first-line interchangeability of hardware modules precludes the use of fully deterministic scheduling solutions in IMA systems. Such solutions are, in any case, inappropriate for a number of other reasons (see section 4.1). Instead, we need to think in terms of *system-level predictability* backed up by appropriate analytical guarantees - we need to be able to confidently forecast the behaviour of the system with respect to its operating environment. Below system level, we must be able to accept a degree of uncertainty about internal behaviour, since we are unable to predict the precise individual and combined behaviours of hardware and software components within the system - this is sometimes termed *bounded non-determinism*. In order to achieve such predictability, we require:

- a set of assumptions about the worst-case behaviour of the operating environment;
- a set of static system level timing requirements described in terms of *hard* end-to-end deadlines;
- the use of flexible scheduling mechanisms to provide bounded non-determinism;
- the use of analytical techniques to predict worst-case system level behaviour.

There are some serious implications for the manner in which certification is achieved for IMA systems. In particular, we must adopt a more abstract view of the notion of timing guarantees without the need to demonstrate complete determinism. Without such a shift in emphasis, the life-cycle cost benefits potentially available from IMA may not be achievable.

4. IMPLEMENTATION APPROACHES FOR AVIONIC SYSTEMS - PAST, PRESENT AND FUTURE?

In the context of this paper, we consider an implementation approach to contain a computational model for application software, scheduling approaches for processor time and other resources, together with timing analysis.

4.1 Traditional Approaches are Inadequate for IMA

Static, *time-driven* scheduling solutions have been applied in the majority of existing industrial solutions for real-time avionics and safety critical systems. Whilst such solutions can provide fully deterministic behaviour, their applicability must be questioned as system complexity increases through increased levels of integration and there is a need for increased system flexibility. Unfavourable characteristics of such solutions are significant and include the following :

- *Inflexibility* - The final scheduling solution is not amenable to incremental change; any changes in application requirements/behaviour or resource availability imply a new static schedule.
- *Compromised application timing requirements* - True timing requirements of processes (or transactions) are compromised as all processes must fit within a common, cyclic schedule.
- *Inefficiency* – Quantity of unused resource may be significant as when a process completes early, spare time is not reassigned to other processes. Also, when timing requirements are compromised (see above), processes are forced to run more frequently than they actually need.

- *Pessimistic worst-case estimates* - Deviation in actual run-time behaviour beyond predicted worst-case behaviour may result in information loss or cycle over-run. This encourages pessimism in design-time predictions of worst-case behaviour (computation times and rates).

- *Lack of responsiveness* - End-to-end response times may be large (compared to the total computation time involved) due to the release of successive activities based on worst-case behaviour of all predecessors. Further, any pessimism in the estimation of worst-case execution times may be directly manifest at run-time as an actual delay in end-to-end response times.

- *Lack of visibility of true precedence relationships* - In the final scheduling solution, there is no identification of true precedence relationships between processes executing on the same processor (or across multiple processors). Unless these relationships are otherwise documented at design-time, this can result in problematic modification or upgrade of the system.

The merits and drawbacks of static, time-driven scheduling have been documented by Audsley[4] and Locke[5], in comparison to static priority-based scheduling, and Stankovic[6,7], in comparison to dynamic on-line scheduling solutions.

4.2 So What Are The Alternatives?

Many of the scheduling methods developed to date, and their associated analysis, are applicable only to a particular subset of the overall problem of scheduling in real-time distributed systems like IMA. Solutions have been typically targeted at communication networks or individual processing nodes in isolation. These solutions do not scale well to distributed systems with a high degree of integration.

4.2.1 Fixed Priority Implementations

Fixed priority implementations range from pre-emptive to non-pre-emptive, including hybrid systems. There is extensive timing analysis available for uni-processor systems[8], covering a range of application requirements including periodic and sporadic processes, hard and soft deadlines, inter-dependent processes with precedence constraints and shared resources. Results documented for distributed systems include the treatment of end-to-end transactions through the application of *release offsets*[9] and through *release jitter analysis*[10]. However, both of these solutions are holistic, *ie.* not partitionable, due to the potential for timing circularities, the net effect of which is to restrict the flexibility of the final solution.

4.2.2 Reservation Implementation Strategy

Reservation-based scheduling supports a more abstract interface between application processes and their required resources - processes specify their time and other resource requirements in terms of a simplistic utilisation-based specification. This gives the scheduler for each resource a high degree of freedom in terms of how processes are subsequently executed such that their timing specifications are met. Off-line timing guarantees require reservation of capacity or bandwidth for each activity assigned to each resource in the system such that all worst-case resource requirements and end-to-end timing requirements are met.

Conventionally, resource reservation has been used for scheduling communications and multimedia applications[11,12,13]. These approaches have not been aimed at safety-critical applications, with little concept of off-line timing guarantees. Neither do they provide an end-to-end scheduling solution over a common set of processing and communication resources. However, resource reservation, when combined with the approaches such as fixed priority scheduling for individual processing resources, can be extended into the safety-critical domain.

4.2.3 Comparison

Upon comparison between cyclic, fixed priority and reservation based approaches, we note:

- *Cyclic* – allows quantifiable off-line computation specification and timing guarantees; inflexible scheduling (not easily changed), hence not easily scalable.

- *Fixed Priority* - allows quantifiable off-line computation specification and timing guarantees; flexible scheduling; non-partitionable timing analysis, hence not easily scaled.

- *Reservation Based* - allows quantifiable off-line computation specification and timing guarantees; flexible scheduling and scalability via partitionable timing analysis.

In the next section, we develop a reservation based approach for IMA systems.

5. TOWARDS A RESERVATION-BASED IMPLEMENTATION APPROACH FOR IMA

This section describes the development of a reservation-based implementation approach for IMA systems. This approach enables partitionable system design and is underpinned by partitionable timing analysis. We consider the issues of computational model, specification of timing requirements, scheduling approach and timing analysis.

5.1 Computational Model

An application consists of a number of transactions. In its simplest form, a transaction consists of a linear sequence of processes with one-to-one precedence constraints. We further decompose each process into a precedence constrained graph of activities, the boundaries between activities in a transaction occur at points where the resource requirements of that process change. An activity (or process) does not start until its predecessor activity (or process) has completed execution (hence sent some form of "event", potentially data), *and* a minimum start time (specific to that activity or process) has elapsed.

5.2 Timing Requirements

The timing requirements of a transaction, E, with component processes $\{\tau_{E,i}$ where $i = 1,..,n\}$, can be expressed as (T_E, D_E, J_E), where T_E specifies the minimum inter-arrival time, or period, of the transaction; D_E specifies the end-to-end deadline of the transaction; J_E defines the tolerable output jitter.

Each process $\tau_{E,i}$ consists of a number of constituent activities $\{\tau_{E,i,j}$ where $j = 1,..,m\}$. Each activity has timing requirements $D_{E,i,j}$, the deadline of the activity, $T_{E,i,j}$, the period of the activity, $O_{E,i,j}$, the minimum start time of the activity (relative to the start time of transaction E), and $C_{E,i,j}$, the worst-case computation time of the activity. We note that under most circumstances, the period of each activity is equal to the period of the transaction.

Note that two separate run-time behaviours are evident. Firstly, if the minimum start times ($O_{E,i,j}$) for *all* activities in *all* transactions are set to 0, an *event-driven* system is obtained, *ie.* an activity can proceed as soon as its predecessor(s) are complete. Secondly, if minimum start times are set, a more static effect is seen, where the intervals in which all computations occur are known off-line – noting that exactly *when* the computation occurs within that interval is completely dependent upon the scheduling strategy employed.

5.2.1 Decomposition of Timing Requirements

The decomposition of processes into activities implies that a decomposition of timing requirements is also required. One approach is to treat each transaction separately, merely splitting the process into a control-flow graph of where arcs between nodes in the graph represent a change in resource requirements. This can be achieved using the techniques outlined for basic block analysis in section 2.3.2 – this also enables the value of $C_{E,i,j}$ to be assigned. The assignment of minimum start times (if required), and deadlines, can be according to any chosen heuristic. Examples include equal or proportional distribution of the *slack* time in a process, *ie.* the difference between its deadline and worst-case computation time. We note that optimal solutions require a completely holistic approach[9,10], where offsets and activity deadlines are assigned with full knowledge of *all* transactions in the system.

5.3 Scheduling

As indicated in section 4.2.2, the reservation-based implementation approach does not prescribe a specific scheduling approach, only that it is able to be applied to a specific resource without reference to the scheduling of other resources – this is fundamental to achieving partitionable timing analysis (see section 5.4). For the purposes of this paper, we assume that scheduling of the processor time resource is according to fixed priority.

The scheduling of other resources in the system is more difficult to prescribe – however, they must be complementary to the scheduling of processor time. Essentially, we must consider the possible forms of co-ordination between resource scheduling mechanisms in the system; these are depicted in Figure 2 (where R1 and R2 are inter-dependent resources). Resources which can sustain their own operation are termed *active* and those which cannot are termed *passive*. Passive resources are entirely dependant upon active ones for their operation.

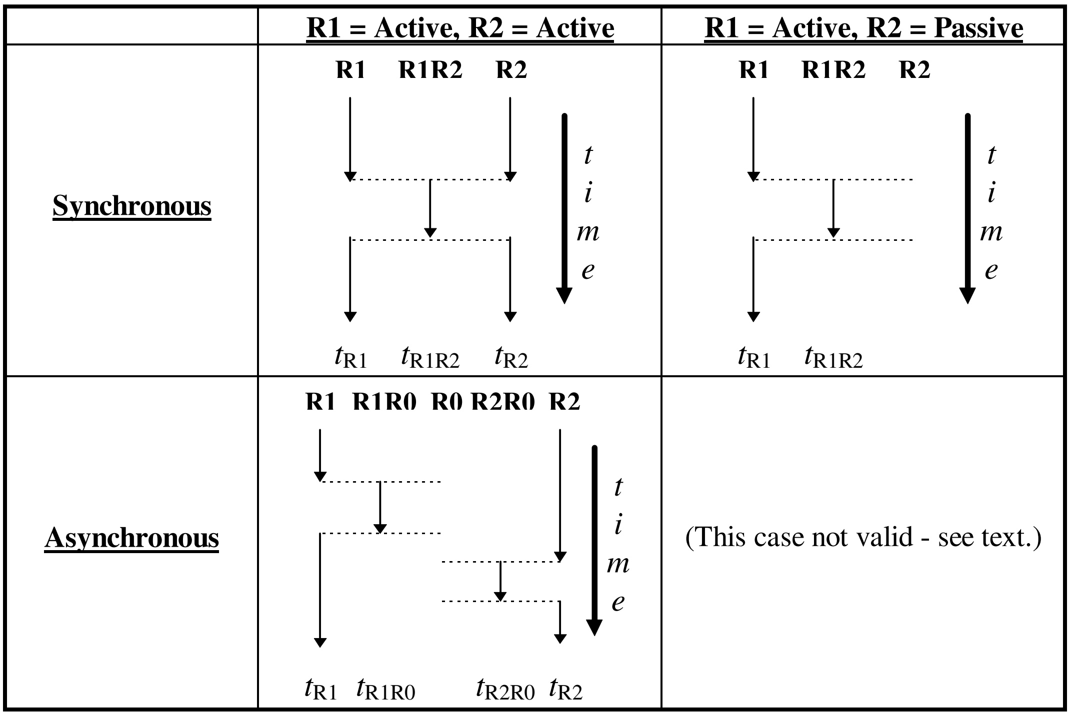

Figure 2 - Co-ordination between Scheduling Mechanisms

In the two synchronous cases depicted in the diagram, three time domains must be considered; t_{R1}, t_{R2} and t_{R1R2}. In domains t_{R1} and t_{R2}, R1 and R2 may execute activities with resource requirements {R1} or {R2} independently of each other. In t_{R1R2}, both resources R1 and R2 must simultaneously execute activities with resource requirements {R1, R2}. For asynchronous co-operation, some intermediate passive resource (R0, say) must be introduced to provide temporal decoupling of R1 and R2. Hence, the case of R1-active and R2-passive is not valid, since R2R0 cannot exist (two passive resources cannot co-operate directly since there is no mechanism to provide control). This representation is scaleable in the sense that R1 and/or R2 can represent a single resource or a collection of resources, given that any set of resources which contains at least one active resource is itself active.

In general, the transition between non-co-operative and co-operative behaviour may be time-driven or event-driven. Time-driven co-operation is underpinned by a common view of time across resources or, more precisely, bounded clock drift. Event-driven co-operation may be achieved without a common view of time but leads to increased risk of transient overload. The choice of time-driven or event-driven transition between non-co-operative and co-operative behaviour is significant in terms of partitionability of the temporal design and analysis of the system. In particular, we wish to support event-driven activity arrivals within transactions as far as possible in order to achieve improved average response times. Under such conditions, the use of synchronous co-operation would lead to holistic analysis, since the calculations of interference effects on each resource are mutually dependent. Asynchronous co-operation, however, does not suffer from this problem and gives us a basis for providing partitionable design and analysis for event-driven activities with co-operation across resources.

For more flexible scheduling solutions to distributed systems, co-ordination must be achieved on a more 'as required' basis using some combination of time-driven and event-driven mechanisms for run-time transfer of control. Hence, in our solution, we adopt asynchronous event-driven transfer of control between schedulers. Whatever the choice of scheduling solution for individual processors or communication media, the combined behaviour of these must be considered.

5.4 Timing Analysis

The timing analysis for the approach prescribed in the previous sections needs to be partitionable. This is achieved by noting that the partitioning of resource requirements during timing requirement decomposition (of processes to activities) should enable us to perform the timing analysis of each resource *independently* from each other resource. We consider the levels of timing analysis outlined in section 2 (assuming levels 0 and 1 have been completed, and the timing decomposition has been performed for level 2) – we also assume that an allocation of activities and resources to processors has occurred.

- *Level 3 – Processor*

  Here we must calculate whether the resource requirements of activities assigned to a processor will be met. Thus, we perform *separate* analysis for each resource. The analysis for processor time is similar to those proposed in the literature for uni-processor fixed priority scheduling[8]. If a resource is *logical*, *ie.* the processing for critical sections is performed on the processor, it is analysed in exactly the same manner as the time resource. For other resources, a suitable timing analysis is required reflecting the behaviour of that resource, *eg.* intelligent sensor/actuator.

- *Level 4 – System*

  Here we must calculate whether the system timing requirements will be met. We note that for a partitionable analysis, system analysis cannot re-consider analyses already performed above. Essentially, at this stage, the only resources that have not been analysed will be those involved in communication *between* processors. Thus, we have to perform timing analysis upon the process components (activities) that access inter-processor communications media.

  For reservation-based scheduling of communication resources, many bandwidth reservation solutions have been developed, the applicability of which depend on traffic type, medium type and network topology. For multi-access communication media, some form of *demand-adaptive* protocol[14] would seem to be required in order to provide real-time behaviour with the appropriate degree of flexibility - communication delays are boundable and access control incorporates a degree of event-driven behaviour which can improve average responsiveness. For multi-hop networks, the bandwidth reservation solution will require the use of connection-oriented channels and must invariably incorporate some form of message release control as a means of limiting congestion.

  Also at this stage, we must examine the end-to-end deadline requirements. If minimum start times are used for activities, this stage merely involves checking that each activity of each process in a transaction meets its deadline – this has already been calculated by level 3 and level 4 analysis. If no minimum start times are used, *ie.* the system is purely event-driven, the end-to-end problem is more complex and beyond the scope of this paper.

6. CONCLUSIONS

Timing analysis enables systems to be analysed *off-line* to see if time and resource requirements will be met at run-time. It also allows designers to perform analysis at a high-level, prior to code even being written – “what-if” games can also be played with different system architectures, software designs and resource provision to see if timing requirements will be met.

IMA systems provide unique challenges for timing analysis, attributable to highly integrated resource usage. In classic timing analysis, this leads to holistic analysis where all components need to be considered together – analysis is not partitionable to reflect the modularity of the system.

The need for IMA solutions to support first-line interchangeability of hardware modules precludes the use of fully deterministic scheduling solutions. Instead, the solution must incorporate bounded non-determinism and provide system-level predictability. We propose the *reservation implementation strategy* that enables *partitionable* system design and timing analysis. The solution consists of:

- An approach to decomposition whereby system transactions with end-to-end timing constraints are decomposed into processes and activities with indivisible resource requirements. Explicit timing requirements are assigned to activities only where required for subsequent timing analysis.
- Co-ordination between scheduling mechanisms responsible for individual system resources shall be consistent and amenable to timing analysis.
- Partitionability of the system design and timing analysis can be achieved through the use of reservation-based scheduling. Both static, priority-driven scheduling and dynamic scheduling mechanisms offer a potential basis of such a solution.

It is proposed that such a solution will lead to a flexible, end-to-end scheduling solution with partitionable timing analysis for functionally and physically integrated real-time distributed systems. Such requirements are typical of IMA systems proposed for future aerospace applications.

7. REFERENCES

1. Edwards, RA., "ASAAC Phase 1 Harmonised Concept Summary," *Proc. of ERA Avionics Conference and Exhibition*, 1994.
2. Wake, AS., Miller, PR., Moxon, P., Fletcher, MA., "Modular Avionics Operating System – Software Concept," *Proc. of ERA Avionics Conference and Exhibition*, 1996.
3. Field, D., Grigg, A., "The Impact of Interchangeability Requirements on Operating Systems for Modular Avionics," *Proc. of ERA Avionics Conference and Exhibition*, 1994.
4. Audsley, NC., Burns, A., Tindell, KW., "The End of the Line for Static, Cyclic Scheduling?," *Proc. of 5th Euromicro Workshop on Real-time Systems*, 1993.
5. Locke, CD., "Software Architecture for Hard Real-time Applications: Cyclic Executives vs. Fixed Priority Executives," *Real-time Systems Jnl.* 4(1), 1992.
6. Stankovic, JA., "What is Predictability for Real-time Systems," *Real-time Systems Jnl.* 2(4), 1990.
7. Stankovic, JA., "Distributed Real-time Computing: The Next Generation," *Jnl. of the Society of Instrument and Control Engineers of Japan*, 1992.
8. Burns, A., "Pre-emptive Priority-based Scheduling: An Appropriate Engineering Approach," in Son, SH., *Advances in Real-time Systems*, ISBN 0-13-083348-7, 1995.
9. Bate, I., Burns, A., "Timing Analysis of Fixed Priority Real-time Systems with Offsets," *Euromicro Workshop on Real-time Systems*, 1997.
10. Tindell, KW., "Fixed Priority Scheduling of Hard Real-time Systems," *Dept. of Computer Science, Uni. of York, UK*, report YCST-94/03 (DPhil thesis), 1994.
11. Jeffay, K., Bennett, D., "A Rate-based Execution Abstraction for Multimedia Computing," *Proc. of Int. Workshop on Network and Operating System Support for Digital Audio and Video*, in Little, TDC., Gusella, R., *Lecture Notes in Computer Science* 1018, 1995.
12. Mercer, CW., Savage, S., Tokuda, H., "Processor Capacity Reserves: OS Support for Multimedia Applications," *Proc. of IEEE Int. Conf. on Multimedia Computing Systems*, 1994.
13. Shin, K., Chang, YC., "A Reservation-based Algorithm for Scheduling both Periodic and Aperiodic Real-time Tasks," *IEEE Trans. on Computers* 44(12).
14. Kurose, JF., Schwartz, M., Yemini, Y., "Multi-access Protocols and Time-constrained Communication," *ACM Computing Surveys* 16(1), 1984.
15. "Ada95 Reference Manual," *ANSI/ISO/IEC* Std. 8652:1995, 1995.
16. Barnes, J., "High Integrity Ada : The SPARK Solution," *Addison-Wesley* 1997.