A Method for Design and Analysis of Next Generation Aircraft Computer Systems

Alan Grigg, Dr. Neil Audsley
British Aerospace Dependable Computing Systems Centre,
Department of Computer Science,
University of York,
York, UK

Martyn Fletcher, Allan Wake
Advanced Systems Group,
British Aerospace Military Aircraft & Aerostructures,
Brough,
East Yorkshire, UK

Abstract. Next generation aircraft computer systems require a major advance in system design practice and associated methods of analysis compared to even the most modern aircraft systems now in service. As well as the ongoing need to demonstrate predictable, real-time behaviour for a wide range of safety critical and mission critical applications, a significant improvement in flexibility is required in order to support cost-effective design-time and in-service modifications to the system. The system design and analysis problem is further complicated by a trend towards increasing functional and physical integration. In this paper, we address the temporal aspects of the problem and propose a design/analysis approach based on *resource reservation*. This supports top-down development by incorporating only appropriate timing information at each stage of development and defering the use of implementation details until absolutely necessary. At each stage of refinement, the system model supports localised changes with minimal re-analysis, thus minimising the impact (and associated cost) of such changes.

INTRODUCTION

This paper describes the results to date of a British Aerospace funded research project in the Dependable Computing Systems Centre (DCSC) at the University of York. The project is addressing the real-time aspects of the design and analysis problem in next generation aircraft computer systems. The work is particularly motivated by the ongoing military Integrated Modular Avionics (IMA) programme but the results have wider applicability to real-time, distributed systems in general.

The military IMA programme (Edwards 1994, Edwards 1997) is a medium-term, multi-national aerospace programme aimed at reducing costs associated with aircraft computer systems over their complete life-cycle. This programme has proposed particularly demanding requirements for future military aircraft systems. In particular these systems will be highly flexible to cope with evolving system requirements and implementation technologies.

A central theme in the IMA programme is the specification of a standard computer system architecture model which will support cost-effective development and in-service maintenance. The architecture is characterised as follows :

Hardware architecture - This must be constructed from standard modules using prescribed inter-connection principles. A number of standard hardware module specifications are described in the documentation (Edwards 1994) - these include the general data processing module and the signal processing module, each of which can be multi-processor. The architecture supports heterogeneous hardware components and interchangeability between modules from different suppliers (Grigg 1994). The communication network is aimed at delivering high inter-connectivity between hardware modules and providing independence from specific network technologies (such as Asynchronous Transfer Mode).

Software architecture - A standard software model is applied across the whole system. This model is depicted in Figure 1 and described below; see (Wake 1996) and (Field 1997) for more details.

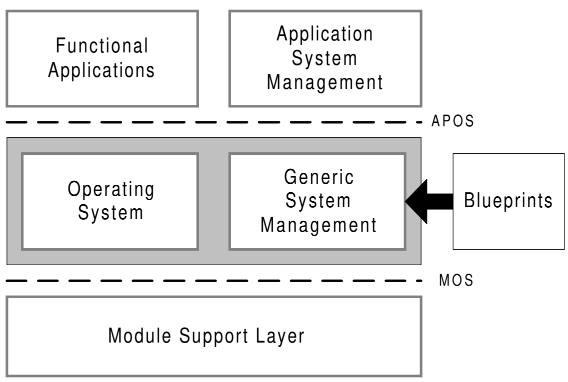

Figure 1. IMA Software Model

Functional applications provide aircraft-specific functionality and are structured according to a standard computational model that supports multi-processing and network-independent communication. Application system management software provides application-specific control over the operation of the system, including mode selection to instigate new mission phases or degraded operating modes under fault conditions. Generic system management software provides the services which implement the configuration/re-configuration mechanisms required for system initialisation, moding and fault recovery.

The operation of the system management software is tailored by *blueprints*. These are data structures which describe the operational requirements for particular aircraft types and mission scenarios and the physical constraints imposed by the hardware architecture of the system.

The operating system provides a generic set of application run-time support services as specified by the APOS interface standard. These services are hardware-independent and include scheduling, communications and the distribution of system time.

The module support layer provides a hardware-specific implementation of the set of services specified in the MOS interface standard. Examples of services provided by this layer include software loading, built-in-test and run-time context switching.

IMA SYSTEM DESIGN TRADE-OFFS

There are some fundamental trade-offs involved in the development of IMA systems (and flexible real-time, distributed systems in general) which have a direct impact on life-cycle cost.

Flexibility v Predictability : An over-riding requirement in the design of real-time systems (in the aerospace domain or otherwise) is the need to demonstrate predictability by off-line analysis. This has driven the design of many industrial solutions for real-time systems, especially those in the safety-critical domain, towards static, cyclic scheduling. Such solutions are deterministic but offer little flexibility (Locke 1992). In particular, they can be difficult or impossible to modify without considering the impact of localised changes on the system as a whole. In contrast, future aircraft systems, exemplified by IMA, aim to provide a high degree of flexibility in order to meet requirements such as :

- Cost-effective modification of the system to incorporate new hardware technologies, system requirements or software components, both during development and in-service. All modifications in-service must occur without the need for complete system re-certification.
- Increased system availability in-service through run-time fault-tolerance and flexible 1st-line maintenance policy[1];

[1] The 1st line maintenance policy is that which is implemented directly on the aircraft by regular maintenance crew, without involvement of the original system integrators or component suppliers.

From a timing perspective, this means that the scheduling and timing analysis solution must be able to cope with changes in software requirements, implementation details, computation times, communication characteristics and software-hardware allocation details. This must be achieved in an incremental manner, *ie.* without complete re-design and analysis.

The use of static, off-line analysis techniques is required to demonstrate predictability in next generation aircraft systems since the certification problem will be significantly larger and more complex than previously encountered. This is due to increased stringency of certification standards (UK MoD 1997), the highly integrated nature of the system under consideration and the increasing extent of safety critical software in aircraft systems. The use of sound analytical techniques should be backed up by extensive tool support.

Integration v Isolation : The design and analysis of IMA systems is complicated by a desire to achieve a high degree of integration. This is sought from two perspectives :

- Functional integration - Extensive sharing of data between (traditionally independent) aircraft functions, to improve operational performance.
- Physical integration - Extensive sharing of processing, communication and I/O resources, to reduce the physical size/weight of the system and improve system availability (by sharing redundant hardware resources).

From a safety perspective, there is a need to demonstrate that any timing guarantees associated with safety critical applications can not be compromised by the execution (or failure) of less critical applications. Also, in order to meet the flexibility goals described above, the solution must support localised changes to software or hardware design/implementation at any stage of the system life-cycle with minimal re-design and/or analysis. These requirements are in direct conflict with the proposed highly integrated nature of IMA systems, since the sharing of physical and/or logical resources between aircraft functions could easily introduce inter-dependencies between their timing behaviour.

Run-time Efficiency v Technology Transparency : Probably the most demanding requirement for IMA is to support hardware interchangeability (Grigg 1994). This refers to the replacement of a given hardware module by another of the same (standard) specification but different implementation. Module replacement may occur during system development, routine maintenance or in-service upgrade. Two system upgrade options have been identified :

- Limited upgrades - multiple, known hardware targets, where software components can execute on any one of a number of known hardware implementations in their life-time (and can be compiled directly for each).

- Unlimited upgrades - multiple, unknown hardware targets; where software components can execute on any hardware implementation that conforms to a standard specification.

To maximise upgrade potential, there is a need to avoid technology-specific details as far as possible in the design and implementation of IMA software components. The extent to which technology transparency is adopted has impact on the run-time efficiency of the system. It is often suggested that efficiency is not a major concern in IMA systems. We suggest, however, that efficiency *is* a concern from the perspective of hardware resource usage. In terms of the overall aircraft design, it is important to maintain the physical size/weight of the computing platform within prescribed levels. In terms of meeting in-service availability targets, it is important to make effective use of available processing capacity and communication network bandwidth.

LIMITATIONS OF TRADITIONAL REAL-TIME SYSTEM DESIGN/ANALYSIS

In this section, we discuss the merits of fixed priority scheduling solutions and their associated timing analysis for application to next generation avionic systems. We focus on *fixed priority* solutions (Burns 1995) as these appear at first sight to be the most applicable, providing flexibility at run-time whilst also supporting off-line analysis. We shall see, however, that the degree of flexibility offered may not be adequate to meet currently proposed requirements for IMA systems. With regard to other classical scheduling solutions; it is widely accepted that static, time-driven solutions (incorporating cyclic executives) are too inflexible and that dynamic solutions (based on earliest-deadline-first) are too unpredictable for safety critical systems.

Traditionally, timing analysis is only performed after all system timing requirements, software implementation details and target hardware details have been finalised. The overall approach is depicted in Figure 2 and discussed below.

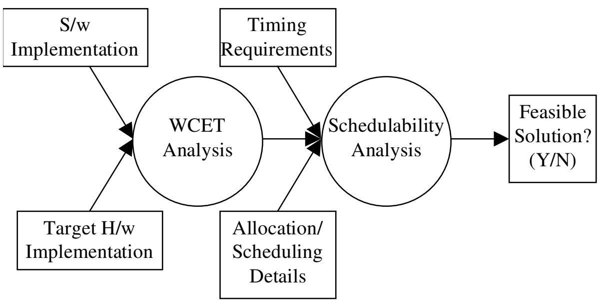

Figure 2. Traditional Real-time System Design/Analysis

The first step is to determine the worst-case execution time (WCET) for each software component in the system. This is the time that a given software component will take to execute given sole access to its required processing and communication resources, *ie.* independent of other software components. The WCET is dependent upon both software and target hardware implementation details.

Schedulability analysis is then performed (off-line) to determine whether the overall system timing requirements will be met at run-time. Schedulability analysis is dependent upon the results of WCET analysis and the scheduling regimes employed throughout the system. Typically, schedulability must be re-assessed for any software component upon change to <u>any</u> of the following :

- its own timing requirements, WCET or communication requirements;
- the timing requirements, WCET or communication requirements of any other software component allocated to the same shared resource(s);
- the hardware implementation of any required processing or communication resource;
- the overall allocation (mapping) of software components to hardware resources.

In fixed-priority scheduling, a priority value is assigned (off-line) to each software component and the run-time scheduler simply executes the active component with the highest priority at any given time. Notably, changes to the requirements or implementation of one software component can impact on the behaviour of another, even when there is no logical relationship (*eg.* communication) between them. Hence, such solutions can be very sensitive to changes in timing requirements and WCETs (Punnekkat 1997). The sensitivity of fixed-priority solutions in distributed systems is particularly dependent upon the manner in which end-to-end transactions are decomposed. Tindell's solution uses event-driven components within transactions (Tindell 1994) to give good end-to-end responsiveness but cannot, in general, support localised changes without system-wide re-assessment of schedulability. Bate's solution uses time-driven components within transactions (Bate 1997) to limit the extent of re-analysis required but to the detriment of end-to-end response times.

Under most scheduling regimes, changes in hardware implementation details invariably enforce re-analysis. This is because worst-case execution times are linked to the target hardware details and worst-case communication delays are linked to the class and implementation of the medium (shared memory, bus, network). The best we can do to support such changes is to limit the extent of re-analysis to the hardware components in question with minimal disturbance on the rest of the system.

The 'minimal disturbance' goal arises when one or more software components allocated to the hardware concerned becomes unschedulable and must be re-allocated with minimal re-analysis. In this context, Bate's fixed-priority solution supports limited hardware changes, where all software components allocated to the changed hardware component(s) remain schedulable, but otherwise does not support re-allocation without re-analysis of entire processing sites and potentially the whole system.

In order to support flexible re-allocation/re-configuration with minimal re-analysis, we need to provide allocation-transparency at whatever level we wish to apply schedulability analysis, *eg*. individual software 'processes'. The provision of common, location-independent communication services via an operating system supports this from the functional perspective, but specific consideration is also required from the timing perspective. Solutions which distinguish between communications via shared memory and networks, at the application level, inherently impose restrictions on future changes to the system.

In conclusion, we suggest that a fundamentally different approach is needed for systems which require the degree of flexibility of IMA.

RESERVATION-BASED DESIGN/ANALYSIS

A number of scheduling and timing analysis solutions in the broad class of reservation-based methods have been proposed for distributed, multimedia applications. Notable results can be found in (Mercer 1994), (Lee 1996), (Stoica 1996), (Jeffay 1995) and (Yau 1996). Schemes have also been identified for communications network design/ analysis, including (Ferrari 1995), (Ermdahl 1997), (Zhang 1993) and (Karlsson 1996). A common characteristic of these solutions is a highly abstract interface between application software and the scheduler (or communication protocol). This interface allows applications to specify their processing and other resource requirements generally in terms of some utilisation-based or execution rate-based specification. This gives the scheduler a high degree of flexibility in terms of how software processes are subsequently executed such that their timing requirements are met.

In general, resource reservation schemes offer *minimum* resource allocation guarantees and can theoretically be applied in the same manner for both *hard* and *soft* classes of real-time systems. The difference being that for hard real-time systems off-line analysis must show that all timing and resource requirements can be met within such guarantees.

The applicability to next generation avionic systems of existing resource reservation schemes must be questioned from the aircraft certification viewpoint. These have typically been designed to support systems with more dynamic timing requirements, *ie*. where the timing requirements of the system can change at run-time depending on the state of the environment or the system itself. For example, some of these schemes support on-line 'negotiation' over shared resource access in order to allow applications to adapt their rate or execution time according to particular environmental conditions. However, there are many desirable characteristics of these schemes which we aim to incorporate into a solution for IMA :

- independent timing guarantees for individual software components;
- simple schedulability tests;
- end-to-end responsiveness and scheduling flexibility through event-driven behaviour.

There are two main challenges in developing a reservation-based design/analysis solution for IMA :

- How to preserve the desirable properties of such solutions in a scheme which provides *off-line* predictability of worst-case resource utilisation and application responsiveness?
- How to support the demanding system-level flexibility requirements of IMA without delivering unacceptable utilisation levels?

Our proposed solution is based on the *refinement* of timing information from system requirements definition through to software execution on a specific hardware platform. The solution is illustrated in Figure 3 and can be described in two main stages :

Stage 1 : Abstract system design/analysis - The first aim is to capture true timing requirements at a system-level, ie. the interface between the system and its operating environment. The logical architecture of the system can then be developed including the entire application software suite and its associated timing requirements. The notion of resource reservation is introduced and off-line analysis is performed on the basis of implementation-independent assumptions about the underlying hardware.

Stage 2 : Target-specific implementation/validation - This stage supports the introduction of hardware implementation details in conjunction with some scheduling regime which preserves the assumptions made during stage 1. Resource reservations must be shown to be valid for the given target hardware prior to software execution.

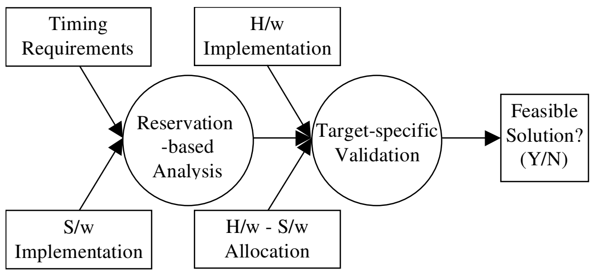

Figure 3. Reservation-based Design/Analysis

STAGE 1 : ABSTRACT SYSTEM DESIGN/ANALYSIS

Abstract system design/analysis can be broken down into a number of shorter stages, each of which is described in more detail below :

- System-level timing requirements;
- Logical architecture definition;
- Reservation-budget assignment;
- Reservation-based analysis.

System-level Timing Requirements : The objective of the first stage is that all true system-level timing requirements are captured. These should provide a complete description of all desired interactions between the system and its perceived operating environment and should be devoid of any technology-dependent details. In order to retain and refine these timing requirements during system development and maintenance, we associate these with specific end-to-end *transactions*.

Definition : A *transaction* E_i, from a system of n^E transactions $\{E_i \; ; \; i = 1,..,n^E\}$, describes the temporal relationship between :

- a set of *input events* $\{ e_k^{in}(E_i) \; ; \; k = 1,..,n^{in}\}$ from either the external environment of the system or some other transaction(s);
- a set of *output events* $\{ e_k^{out}(E_i) \; ; \; k = 1,..,n^{out}\}$ to either the external environment of the system or some other transaction(s).

Transaction timing requirements should be captureed in terms of the following parameters :

- let $\Omega^{inTOL}(E_i)$ denote the required worst-case separation in time between input events (transaction input window or input event *phase*); similarly, let $\Omega^{outTOL}(E_i)$ denote the required worst-case separation in time between output events (transaction output window or output event *phase*);
- let $\hat{d}(E_i)$ denote the required separation in time between transaction input and output event windows;
- let $\{ J_k^{inTOL}(E_i) \; ; \; k = 1,..,n^{in}\}$ denote any defined tolerance on arrival jitter (variability) for each input event; similarly, let $\{ J_k^{outTOL}(E_i) \; ; \; k = 1,..,n^{out}\}$ denote any tolerance on response jitter (variability) for each output event;
- let $\hat{a}(E_i)$ denote the required minimum time interval between successive instances of the transaction.

At this stage, there is no restriction on sharing system-level inputs and outputs between transactions. A single system input value could contribute to more than one transaction or a number of transactions could contribute to the same system output.

Figure 4 illustrates the above timing requirements for a single transaction.

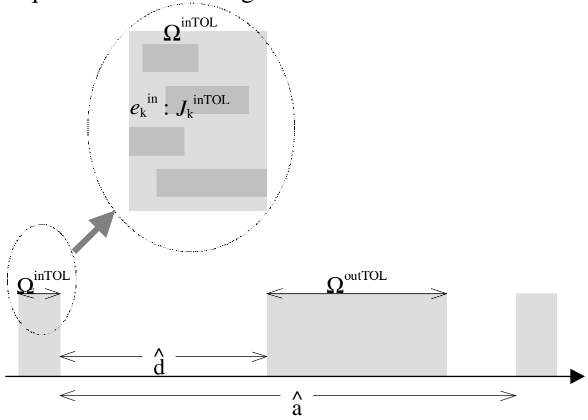

Figure 4. Transaction Timing Requirements

Logical Architecture Definition : At this stage, we address the refinement of end-to-end timing requirements through the development of the software architecture in a manner which preserves hardware technology transparency. This supports deferred selection of hardware components during system development and the interchangeability of hardware components in-service.

Whatever the programming language and method chosen for software development, each transaction identified will typically embody a number of inter-related software components (which could be referred to as 'tasks' or 'processes', depending on the methodology or language). For our purposes, however, we adopt a very specific object of analysis, which we term an *activity.*

Definition : An *activity* is any stage in an end-to-end transaction with static resource requirements.

Transactions are thereby described in terms of precedence-constrained activities with the following characteristics :

- a set of *input* activities to service transaction input events;
- a set of *internal* activities which implement the main body of the transaction;
- a set of *output* activities to service transaction output events.

Complex transactions may be defined with concurrent or conditional execution of activities. If we denote the set of activities resulting from decomposition of transaction E_i by $\{\lambda_{i,j} \; ; \; j = 1,..,m_i\}$, then for any activity $\lambda_{i,j}$:

- let $\hat{a}_{i,j}$ denote the minimum time between successive instances (arrival for execution);
- let $V_{i,j}$ denote its execution-time resource requirement, termed its *reservation budget*, equal to some real value in the interval (0, 1];
- let the function $\boldsymbol{Q}_{i,j}$ express any release conditions in terms of its predecessors (if any).

The *reservation budget* $V_{i,j}$ for each activity $\lambda_{i,j}$ gives a simplistic measure of the minimum resource bandwidth required. This value can be specified early in development then refined as more information becomes available. The assignment of reservation budgets is discussed further in the next sub-section. Note that $V_{i,j}$ is **not** the traditional worst-case execution time (or analogous communications resource requirements), although such values do form the basis for budget validation at a later stage.

The need for this form of decomposition arises due to the focus on system-level, end-to-end behaviour. In a traditional 'process-oriented' design, a given software process will typically contribute to more than one end-to-end transaction. The approach proposed here is more a real-time data-flow model than a process-oriented model. End-to-end data flows are captured and implemented as transactions, from system-level input to system-level output. Within each transaction, all timing and resource requirements associated with data transformation and communication are captured and implemented in terms of activities. Depending on the software design methodology used and the nature of the application, an activity could encapsulate anything from a few lines of code to many hundreds of lines.

More generally, multiple activities can be mapped to the same piece of code - any unit of code represents a shared logical resource that can be instantiated for different transactions, with appropriate resource sharing restrictions. This supports the functional integration goal of IMA.

At this stage, flexibility of the final system is enabled by avoiding any over-specification of timing requirements - as suggested in (Gerber 1995) and (Fohler 1997). In particular :

- Explicit specification of release times/offsets is discouraged. Output stability requirements are specified as transaction jitter tolerance values rather than specific release times on output activities. This also serves to improve average responsiveness of the system.
- There is no enforcement of related (harmonic) arrival rates on software components for the purpose of simplifying the scheduling solution, as with static, cyclic scheduling solutions.

Reservation Budget Assignment : Reservation budgets are defined (and successively refined) off-line, using some or all of the following information, depending on the stage of system development :

- the timing requirements of the parent transaction;
- the source code (and any intermediate language) representation of the activity;
- the timing behaviour of the same activity on one or more known hardware targets.

Notice that the reservation budget associated with any one activity is independent of :

- other activities which share the same resource(s);
- the underlying hardware implementation;
- the software-hardware allocation.

Reservation budgets form the basis of hardware-independent timing *obligations* on the final target system. Consequently, target-specific validation must be performed when the target hardware implementation is known (either off-line or on-line) to show that these obligations will be met at run-time.

Reservation-based Analysis (RBA) : Given the complete set of timing requirements, reservation budgets and precedence constraints, it is possible to perform an abstract form of schedulability analysis for the system which we call *reservation-based analysis* (RBA). There are, in fact, many different reservation-based schemes we can define upon which to base our abstract timing analysis. A generic set of requirements can be defined to which any such scheme should conform, as follows :

- Discretisation of the time associated with each shared resource into intervals of defined maximum duration - the *reservation period*, denoted by Δ. For any particular scheme, the reservation period can be fixed in duration at run-time or allowed to vary according to actual demand (up to a maximum of Δ).
- A method for allocating (no less than) a minimum resource-capacity to each activity $\lambda_{i,j}$ in each reservation period based on its assigned reservation budget, $V_{i,j}$. For any particular scheme, the time and/or order of resource allocation within each reservation period can be fixed at run-time or allowed to vary according to actual demand.

This gives a consistent framework for timing analysis based on bandwidth reservation across all types of resource in a distributed system. This covers both logical (data, code) resources and physical (processor, communications) resources.

STAGE 2 : TARGET-SPECIFIC IMPLEMENTATION/VALIDATION

At this stage, we impose a number of constraints on the implementation of the system in order to support our top-down development philosophy. These constraints can be described in terms of a standard implementation model that supports technology transparency. This model is depicted in Figure 5 with each of the layers described below.

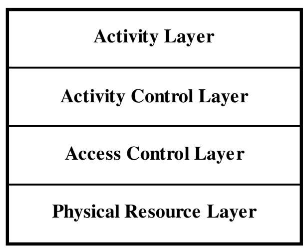

Figure 5. Implementation Model

- Activity layer - Contains the set of activities and their associated timing requirements which comprise the abstract system model.
- Activity control layer - Provides any on-line mechanisms needed to support :
  - validation of reservation budgets on a given target;
  - control of activity release times/rates, *eg.* to counteract undesired effects of jitter;
  - reconfiguration, *eg.* driven by mode change, hardware failure or system upgrade.
- Access control layer - Provides reservation-based scheduling mechanisms and communications protocols. The actual run-time scheduling regimes can be implemented on different physical resources in any way that preserves the timing characteristics of the abstract system model.
- Physical resource layer - Contains the entire (heterogeneous) set of physical processing and communication media for the system

A key part of the implementation model is the validation of reservation budgets for a given target, since the RBA model has been developed independently of the target hardware. Validation can be performed whenever the target hardware implementation is known, either off-line or on-line. This depends on the extent to which the system is required to support changes in target hardware within its life-time. For systems which *do not* require to support changes to the hardware platform once in-service, $V_{i,j}$ can easily be validated off-line using the target-specific worst-case execution time of the activity $\lambda_{i,j}$, denoted by $C_{i,j}$, and the end-to-end deadline of its parent transaction, E_i. If we then define $D_{i,j}$ to be some intermediate activity deadline derived from the transaction deadline $D(E_i)$ by distributing slack time fairly across its component activities, *ie.* proportional to their computation times, then the validation test is simple : $V_{i,j} \geq C_{i,j}/D_{i,j}$.

For systems with more demanding requirements to support hardware changes in-service, like IMA, $V_{i,j}$ values must be validated at system initialisation or reconfiguration time using some other method. For instance, in an IMA system, it is proposed that the software implementation is stored and manipulated in some target-independent form (Grigg 1994, Audsley 1998). This could be some portable code format such as ANDF (Macrakis 1993) or Java byte-code (Nilsen 1996). Target-specific validation would then involve portable code instruction times for the given target.

Where the result of validation is negative, *ie.* certain timing obligations cannot be supported, then it may be necessary to reconfigure the system. Where this is still unsuccessful, it may be necessary to support degraded operation of the system. The flexible design/analysis approach we have described supports both of these options.

SUMMARY

We have proposed a method for top-down design/analysis of next generation aircraft computer systems, from timing requirements through to software execution on a specific hardware platform. The method is applicable to real-time, distributed systems in general.

Schedulability analysis is performed off-line on the basis of assumptions about the eventual hardware implementation. When the hardware implementation is defined late in the development phase or modified in-service, target-specific validation is performed. Hence, our approach is supportive of systems where in-service hardware interchangeability is required, where implementation details can play no part in off-line analysis. At each stage of refinement, the model supports localised changes with minimal re-analysis.

Work is continuing to extend the theoretical reservation-based analysis model and to develop a practical demonstration.

REFERENCES

(Audsley 1998) Audsley, NC., Grigg, A., “Portable Code in Avionic Systems,” *Proc. of IEE Colloquium on Real-time Systems*, 1998.

(Bate 1997) Bate, I., Burns, A., “Timing Analysis of Fixed Priority Real-time Systems with Offsets,” *Proc. of Euromicro Workshop on Real-time Systems*, 1997.

(Burns 1995) 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.

(Edwards 1994) Edwards, RA., “ASAAC Phase 1 Harmonised Concept Summary,” *Proc. of ERA Avionics Conference and Exhibition*, 1994.

(Edwards 1997) Edwards, RA., “Exploiting the Potential of Integrated Modular Avionics,” *Proc. of ERA Avionics Conference and Exhibition*, 1997.

(Ermedahl 1997) Ermedahl, A., Hansson, H., Sjodin, M., “Response Time Guarantees in ATM Networks,” *Proc. of Real-time Systems Symposium*, 1997.

(Ferrari 1995) Ferrari, D., Gupta, A., Ventre, G., “Distributed Advance reservation of Real-time Connections,” *Proc. of Int. Workshop on OS Support for Digital Audio and Video*, 1995.

(Field 1997) Field, D., *et al* “Software Techniques for IMA - System Management, System Security and Blueprints,” *Proc. of ERA Avionics Conference and Exhibition*, 1997.

(Fohler 1997) Fohler, G., “Dynamic Timing Constraints - Relaxing Overconstraining Specifications of Real-time Systems,” *Proc. of*

Real-time Systems Symposium (Work in Progress), 1997.

(Gerber 1995) Gerber, R., Hong, S., Saksena, M., "Guaranteeing Real-time Requirements with Resource-based Calibration of Periodic Processes," *IEEE Trans. on Software Engineering* 21(7), 1995.

(Grigg 1994) Grigg, A., Field, D., "The Impact of Interchangeability Requirements on Operating Systems for Modular Avionics," *Proc. of ERA Avionics Conference and Exhibition*, 1994.

(Grigg 1997) Grigg, A., Audsley, NC., "Towards a Scheduling and Timing Analysis Solution for IMA Systems," *Proc. of ERA Avionics Conference and Exhibition*, 1997.

(Jeffay 1995) 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*, in Little, TDC., Gusella, R., *Lecture Notes in Computer Science* 1018, 1995.

(Karlsson 1996) Karlsson, G., "Capacity Reservation in ATM Networks," *Computer Communications* 19(3), 1996.

(Lee 1996) Lee, C., Rajkumar, R., Mercer, C., "Experiences with Processor Reservation and Dynamic QoS in Real-time Mach," *Proc. of Multimedia Japan*, 1996.

(Locke 1992) Locke, CD., "Software Architecture for Hard Real-time Applications: Cyclic Executives vs. Fixed Priority Executives," *Real-time Systems Jnl.* 4(1), 1992.

(Macrakis 1993) Macrakis, S., "The Structure of ANDF: Principles and Examples," *Open Software Foundation - Research Institute*, 1993.

(Mercer 1994) Mercer, CW., Savage, S., Tokuda, H., "Processor Capacity Reserves: OS Support for Multimedia Applications," *Proc. of IEEE Int. Conf. on Multimedia Computing Systems*, 1994.

(Nilson 1996) Nilson, K., "Java for Real-time", *Real-Time Systems Jnl.* 11(2), 1996.

(Punnekkat 1997) Punnekkat, S., Davis, R., Burns, A., "Sensitivity Analysis of Real-time Task Sets," *Proc. of Asian Computing Science Conf.*, 1997.

(Stoica 1996) Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, SK., Gehrke, JE., Plaxton, CG., "A Proportional Share Resource Allocation Algorithm for Real-time Time-shared Systems," *Proc. of Real-time Systems Symposium*, 1996.

(Tindell 1994) 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.

(UK MoD 1997) UK MoD, "Defence Standard 00-55 - Requirements for Safety Related Software in Defence Equipment (Part 1: Requirements)," 1997.

(Wake 1996) Wake, AS., Miller, PR., Moxon, P., Fletcher, MA., "Modular Avionics Operating System – Software Concept," *Proc. of ERA Avionics Conference and Exhibition*, 1996.

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

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

BIOGRAPHY

Alan Grigg received an honours degree in Mathematics at Thames Polytechnic (now University of Greenwich) in 1985. He subsequently joined British Aerospace; eventually working on the specification/design of a software architecture for military IMA systems. He was a British Aerospace representative for the multi-national, military IMA programme (ASAAC). In 1996, he joined the British Aerospace Dependable Computing Systems Centre at the University of York to persue research into real-time system design and timing analysis.

Neil Audsley received a BSc (Hons. First Class) and DPhil from the University of York in 1988 and 1993 respectively. His doctoral thesis considered the scheduling and timing analysis of safety-critical systems. He has published numerous technical papers. He has worked with the British Aerospace Dependable Computing Systems Centre at the University of York since 1995, specialising in timing analysis and kernels for safety-critical systems.

Martyn Fletcher has a honours degree and diploma in Applied Physics from the University of Hull and also a masters degree in Integrated Circuit System Design from the University of Manchester Institute of Science and Technology. For the past nine years, he has worked in R&D at British Aerospace. Initially, his work involved investigating techniques for distribution and fault tolerance in computer systems. He has now worked for some time on the IMA software architecture, both on British Aerospace's internal R&D programme and on national and international programmes.

Allan Wake received an honours degree in Engineering from the University of Humberside. He joined British Aerospace in 1986, working initially on aircraft training aids development. He currently leads the British Aerospace IMA software team which is heavily involved in collaborative, multi-national research programmes including ASAAC.