



EVALUATION OF REALTIME OPERATING SYSTEMS - THE ROLE OF STANDARDS*

0 EXECUTIVE SUMMARY

This report has been prepared for the ASSC Processing Working Group as part of work to evaluate commercial real-time operating system (RTOS) products for avionics applications. A previous report ASSC/330/2/136 proposed a list of candidate RTOS products. The objectives of this report are to establish the role of operating system standards in the evaluation of RTOS products, to generate evaluation criteria and to determine whether sufficient information is available to carry out an evaluation against the criteria.

Standardisation initiatives relating to operating systems are described in Section 2. It is concluded that there are four categories of standards in existence or in preparation. Operating system standards such as ARINC 653 APEX assume a specific highly-integrated system architecture. In comparison, POSIX is an example of a standard which does not depend upon a specific systems architecture. A third category includes the SAE's GOA standard which specifies a system architecture framework, in which some interface could be implemented by an operating system. Finally, there is an indication that the US Navy is proposing a loosely-integrated system architecture with a high use of standards for both software and hardware. As a result of the range of standards relating to operating systems and their differing scope, there are several different evaluations of RTOS products which could be carried out.

* **This report is published by the Avionic Systems Standardisation Committee (ASSC) to advance the role of standardisation in avionics. The use of this is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom is the sole responsibility of the user.** Copies of this paper are obtainable from the ASSC Agency as an Avionic Systems Standardisation Committee publication.

Criteria for the general evaluation of RTOS products for use in avionics applications are proposed. To avoid the danger of aggregating all possible operating system features, the criteria focus on the abstract requirements necessary for using a commercial RTOS product in avionics. These include both functional criteria such as deterministic performance as well as the non-functional criteria such as safety, security and supportability. It is likely that the non-functional criteria will be harder to evaluate or to satisfy than the functional criteria.

It is concluded that detailed evaluation of RTOS products cannot be carried out without specifying the architecture within which it is proposed that the RTOS is to be used. POSIX is the only standard with which an RTOS product could directly comply. An RTOS product, or the POSIX standard itself, could be evaluated for compatibility with an architecture-based standard such as ARINC APEX, in order to determine whether the RTOS could be used in the implementation of an APEX compliant system. The functionality provided by an RTOS product could be classified within an architectural framework such as GOA: this could assist in proposing system architecture which maximise use of commercial software and standards. The relationship between Ada95, which has realtime extensions, and RTOS products is also complex since Ada can be used either with an RTOS or instead of one.

It is recommended that future work should focus on the evaluation of RTOS products, especially those which offer POSIX compliance or Ada compatibility, within architecture frameworks which promote the use of other standards and commercial software products. Further evaluation of non-commercial operating system standards or products, such as VRTOS or RTEMS from the US Army is also recommended. Non-functional criteria, especially long-term support and safety, should be considered in preference to more detailed consideration of functional criteria. Finally, emerging standards for interfaces between software objects, such as CORBA, should also be considered.

Table of contents		Page no.
0	Executive summary	1
1	INTRODUCTION	7
1.1	Task Background	7
1.2	Objectives of this Report	7
1.3	Structure of the Report	7
2	Standards Relating to RTOS	7
2.1	Introduction to OS Standardisation	8
2.1.1	System Architecture Models and RTOS Standardisation	8
2.1.2	Commercial Standards	8
2.1.3	Architectural Framework Standards	9
2.1.4	Federated Architectures	9
2.1.5	A Classification of Operating System Standardisation Initiatives	9
2.2	Possible Contents of an RTOS Standard	10
2.3	ARINC 653 - Avionics Application Software Standard Interface (APEX)	12
2.4	ASAAC	13
2.5	F22 Common Integrated Processor (CIP) Software	14
2.6	Avionic Modular Component Acquisition (AMCA)	16
2.7	TARDEC (TACOM) Vetronics Real Time Operating System	17
2.8	POSIX	18
2.8.1	ISO/IEC 9945-1 ANSI/IEEE Std 1003.1	20
2.8.2	Profiles	20
2.9	Ada 95: Real Time Extensions	21
2.10	Society of Automotive Engineers (SAE) Generic Open Architecture (GOA)	22
2.11	Other Initiatives from the Society of Automotive Engineers (SAE)	24
2.12	US New Attack Submarine (NSSL)	27
2.13	The Role of RTOS Standards in Evaluating RTOS Products	27
2.13.1	Compliance of RTOS Products with Standards	28
2.13.2	Using RTOS Products in Standard Compliant Systems	28
2.13.3	Evaluating RTOS Products against General Avionics Requirements	29

Table of contents (cont)	Page no.
2.13.4 Evaluating the Standards for Applicability to Military Avionics	29
2.14 The Standardisation Process	29
3 Evaluation of RTOS PRODUCTS for Avionics Applications	30
3.1 Why Use an RTOS	30
3.2 Different Approaches to the Choice of an RTOS	31
3.3 Types of Criteria for Evaluating an RTOS Product	32
3.3.1 General Criteria: Avionics Data Management	34
3.4 Functional Criteria	34
3.4.1 Scheduling	34
3.4.2 Communication	35
3.4.3 Synchronisation	35
3.4.4 Interrupt Handling	35
3.4.5 Time Management	36
3.4.6 Memory Management	36
3.4.7 Supported Interfaces	36
3.4.8 Supported Processors	37
3.4.9 Performance	37
3.4.10 Health Monitoring	37
3.4.11 Application Languages	38
3.5 Non-Functional Criteria	38
3.5.1 Support and Maintenance	38
3.5.2 Portability	38
3.5.3 Standardisation	39
3.5.4 Supplier Commercial Stability	39
3.5.5 Safety Certification	39
3.5.6 Development Support	39
4 Information Available to Evaluate RTOS Products	39
4.1 Introduction	39
4.2 Commercial RTOS: VxWORKS	40
4.2.1 Available Information	40
4.2.2 Comparison with Information Required	40
4.3 Non-commercial RTOS: RTEMS	42

Table of contents (cont)		Page no.
4.3.1	Available Information	42
4.3.2	Comparison with Information Required	42
4.4	POSIX	43
4.4.1	Available Information	43
4.4.2	Comparison with Information Required	43
4.5	Ada95 Realtime Extensions	44
4.5.1	Available Information	44
4.5.2	Comparison with Information Required	45
4.6	F22 AOS	46
4.6.1	Available Information	46
4.6.2	Comparison with Information Required	46
5	Conclusions and Recommendations	48
5.1	Conclusions	48
5.1.1	Relationship of Standards and RTOS Products	48
5.1.2	Criteria for Evaluating RTOS Products	48
5.1.3	Information Available to Evaluate RTOS Products	49
5.2	Recommendations for Further Evaluation of RTOS Products	49
6	References	50
 Figures and tables		
Figure 1	ARINC 653 System Architecture	12
Figure 2	ASAAC Software Concept	13
Figure 3	F22 AOS and ASM Interfaces	15
Figure 4	GOA Framework Architecture Model	23
Figure 5	Relationship of RTOS Products and Standards	28
Table 1	Classification of OS Standardisation Initiatives	10
Table 2	PASC Projects	19
Table 3	Selected Interface Classes in the GOA Framework	24
Table 4	Additional Models Considered in SAE AIR 4885	26
Table 5	Functional Criteria Categories	33
Table 6	Categories of Non-functional Criteria	33
Table 7	Comparison of information required	41

Table of contents (cont)		Page no.
Table 8	Estimated availability of information for evaluation against the non-functional criteria	42
Table 9	Estimated availability of information for evaluation against the functional criteria	43
Table 10	Estimated availability of information for evaluation against the non-functional criteria	43
Table 11	Estimated availability of information for evaluation against the functional criteria	44
Table 12	Estimated availability of information for evaluation against the functional criteria	45
Table 13	Estimated availability of information for evaluation against the non-functional criteria	46
Table 14	Estimated availability of information for evaluation against the functional criteria	47

1 INTRODUCTION

1.1 Task Background

Future avionics systems are likely to make use of real-time operating systems (RTOS). An RTOS is a software kernel providing infrastructure functions such as task scheduling, task communication and input and output. Since the functionality of the RTOS is not specific to the application, it is possible that a commercial RTOS product could be used. ERA has previously reported to the ASSC in ASSC/330/2/136 (Ref. 1) on real-time operating systems: a list of candidate RTOS products was proposed in this report.

A number of standardisation initiatives for real-time operating systems are in progress. These include the ARINC Project Paper 653 'Avionics Application Software Standard Interface' (Ref. 2) (see Section 2.3), the POSIX standardisation being carried out by the IEEE (Ref. 3) (see Section 2.8), and the SAE's Generic Open Architecture (GOA) initiative (Ref. 4) (see Section 2.10). In addition, real-time extensions have been defined for Ada95 (Ref. 5) (see Section 2.9).

1.2 Objectives of this Report

The overall objective of the investigation reported in this document is to work towards a guidance document for the MOD on the selection of RTOS products for avionics and other military embedded applications. Specific objectives are:

- 1) to establish the role of OS standards such as POSIX and ARINC APEX in the assessment of RTOS products
- 2) to generate a list of criteria for evaluating RTOS products
- 3) to determine whether there is sufficient information available to evaluate RTOS products listed in the previous report (Ref. 1) and how additional information could be obtained.

1.3 Structure of the Report

In Section 2, an overview of the different standardisation initiatives is given and the possible relevance of each of the standards to the evaluation of RTOS products is discussed. In Section 3, the range of approaches for the choice of an RTOS are described and criteria for evaluating RTOS products are established. In Section 4, the information available to evaluate RTOS products is considered. Conclusions and recommendations for further work appear in Section 5.

2 STANDARDS RELATING TO RTOS

A number of standardisation initiatives covering real-time operating systems (RTOS) are in progress. In Section 2.1 the different initiatives are introduced. In Section 2.2 the contents of a typical Operating System standard are described. Details of particular initiatives are given in Section 2.3 to Section 2.12. Section 2.13 concludes this Chapter by describing the possible roles of the different standardisation initiatives in the evaluation of RTOS products.

2.1 Introduction to OS Standardisation

In this section the different types of OS standardisation initiative are compared by considering their origins and objectives. A simple classification of the standardisation initiatives is presented.

2.1.1 System Architecture Models and RTOS Standardisation

In the military and civil avionics sector, the requirement to standardise operating systems has arisen from the need to reduce equipment lifecycle costs by increasing use of standard and commercial components and subsystems in the design of avionics systems. Standardisation of interfaces is a vital part of these efforts. To permit interfaces to be standardised, an architectural model for the system is required, within which the interfaces can be specified. Within most architectural models, an operating system has two roles:

- 1) to hide details of the hardware from the application software so that hardware commonality can be achieved
- 2) to allow interaction between different modules of the application software so that software modularity and reuse can be achieved.

It is important to note that the requirements on the operating system depend on the role required of the operating system within the overall architectural model. Therefore, the operating system requirements cannot be considered independently of the architectural model.

One important difference between architectural models is the degree of integration. The Integrated Modular Avionics (IMA) concept, followed in varying degrees in the ASAAC project (see Section 2.4) and the proposed ARINC 653 standard (Ref. 2) (see Section 2.3), proposes a highly integrated architecture with application software portable across an assembly of common hardware modules. An IMA architecture imposes multiple requirements on an operating system.

Section 2.5 contains information about the operating system using the USAF F22. This operating system has features of IMA systems: although it is not a standard, it is of interest since it is an existing system while the standards such as APEX and ASAAC are still evolving and have not yet been proven in use.

2.1.2 Commercial Standards

An impetus for standardisation of operating systems has arisen as a result of the divergence between different versions of UNIX combined with very strong competitive pressure within the operating system market. UNIX standardisation work is being carried out by the IEEE, under the POSIX title (Ref. 3). This standardisation effort has a wide scope and much of it is not relevant to embedded systems. Further information on POSIX is given in Section 2.8 below.

The POSIX standard is not based on a strongly defined architecture, though there may be some implicit architectural assumptions, such as the use of virtual memory. Actual

UNIX implementations have been developed for micro-computers, mainframes and super-computers. Another standard with weak architectural assumptions is Ada95 (Ref. 5). Although Ada95 is primarily a language standard, it incorporates within the language definition, and its extensions, features of a real-time operating system. An Ada runtime system can either be built-on a conventional operating system, or it can be used instead of an OS or it can be used as well as an OS. The Ada runtime system is discussed in Section 2.9.

2.1.3 Architectural Framework Standards

A general consensus in defence procurement exists in favour of greater use of commercial standards. However, there are a large number of different standards which apply to different aspects of a system. This is most familiar in the context of communications, where layered models of standards, such as the OSI seven layer model, have been adopted. The idea of an architectural framework, such as GOA (Ref. 4) (see Section 2.10), is to allow the role of different standards to be described, so that a particular project can adopt a compatible set of standards, for all the interfaces within the chosen architecture.

2.1.4 Federated Architectures

The US Navy has proposed a highly federated architecture for use in future submarines. A federated architecture includes a separate subsystem for each sub-function: this approach contrasts with the integrated approach of IMA. A federated architecture does not require a system level operating system, so that different operating systems could be used in each subsystem and the requirements placed on an operating system are less complex than those which applied for IMA. However, a federated architecture is only attractive to a procurement agency if the interfaces external to each subsystem conform to well-defined standards, allowing subsystem refinement or replacement with minimum impact on other parts of the system. This approach is described in Section 2.12.

2.1.5 A Classification of Operating System Standardisation Initiatives

Table 1 proposes a classification of OS Standardisation initiatives based on the extent to which the standard assumes or specifies the system architecture.

Table 1 Classification of OS Standardisation Initiatives

Classification	Description	Examples
Integrated Architecture	The standard specifies or assumes an integrated architecture. At the application program interface to the operating system, the individual processing elements are not distinguished. Communication between processes does not depend on the identity of the processor executing the process.	APEX, ASAAC, F22 APOS, AMCA, and others, (VRTOS)
Architecture Independent	The standard makes few assumptions about the architecture of the processing system. The existence of shared memory, or the ability to emulate it, may be assumed.	POSIX, Ada95
Architecture Framework	The standard does not specify the architecture, but describes a framework within which a particular system can be described. The framework may be wide, so that it encompasses a range of system architecture or narrow, so that only a few architectural variations are covered.	GOA, (VRTOS)
Federated Architecture	The standard specifies or assumes a federated architecture. Since a federated architecture has interfaces which are different from those in an integrated architecture, these standards may have a different scope to the standards for an integrated architecture.	US Navy NSSN

2.2 Possible Contents of an RTOS Standard

In this section, an overview of the topics which could be covered in an operating system standard is given.

Application Programming Interface (API)

This includes the operating system features provided to the application software programmer. Typical features include creation and termination of processes, communication and synchronisation between processes, handling of interrupts, timing functions and access to hardware resources such as memory. The definition of the API is

similar in difficulty to the definition of a programming language, such as Ada. Both the syntax of the interface and its semantics, or meaning, must be defined. The syntax covers the number and type of the parameters to each call in the API while the semantics describes the behaviour of the operating system for all possible sequences of calls.

Process Priority and Scheduling

A real-time operating system supports application software divided into multiple concurrent processes (also called tasks or threads). The operating system must share the processing resource between the active processes. There are different scheduling strategies which can be used. In real-time operating systems, the strategy should allow some processes to be given higher priority than others and should minimise the delays to a high priority process caused by the execution of one of a lower priority. It is also desirable for the scheduling to be deterministic, so that the behaviour of the system is predictable.

Configuration Interface

In addition to the programming interface, an operating system may have a configuration interface, which allows certain parameters to be set, which cannot be changed from the API. Configuration can occur either when the software is compiled or when the system is initialised. Configuration covers the memory map, the number of interrupts and properties of the hardware, such as the communications interfaces present. A simple operating system may require features such as the number of processes to be configured whereas a more complex operating system allows this to be changed at runtime using the API.

Performance

For a real-time application with timing requirements, the designer must ensure that the performance and throughput of the operating system are adequate. An operating system standard can specify the maximum and minimum time required for each call to the API, though clearly the absolute times depend on the performance of the processor itself. There are many interactions between the performance of an operating system and its other properties, such as the scheduling strategy, and features of the processor itself, such as the use of a cache. These interactions make it difficult to specify performance standards.

Supported Hardware and Portability

A standard could specify the hardware on which the operating system can be used or, in a more general way, the mechanism to be used to extend the range of supported hardware, for example by the addition of a new device driver. The hardware support specification should cover the processor, including FPU and DSP co-processors, memory management, bus specification and device drivers, including graphics cards and communications interfaces.

2.3 ARINC 653 - Avionics Application Software Standard Interface (APEX)

ARINC Project Paper 653 (Ref. 2) is one of a family of proposed standards for different aspects of Integrated Modular Avionics (IMA) for use in civil aviation. ARINC Report 651 (Ref. 6) is a top-level design guide for modular avionics systems. ARINC Report 652 (Ref. 7) is a management guide for software covering both IMA and traditional equipment. ARINC Specification 659 defines a controlled impedance backplane bus for use in IMA equipment. Other specifications, such as ARINC 629, define data buses which could be used in IMA systems

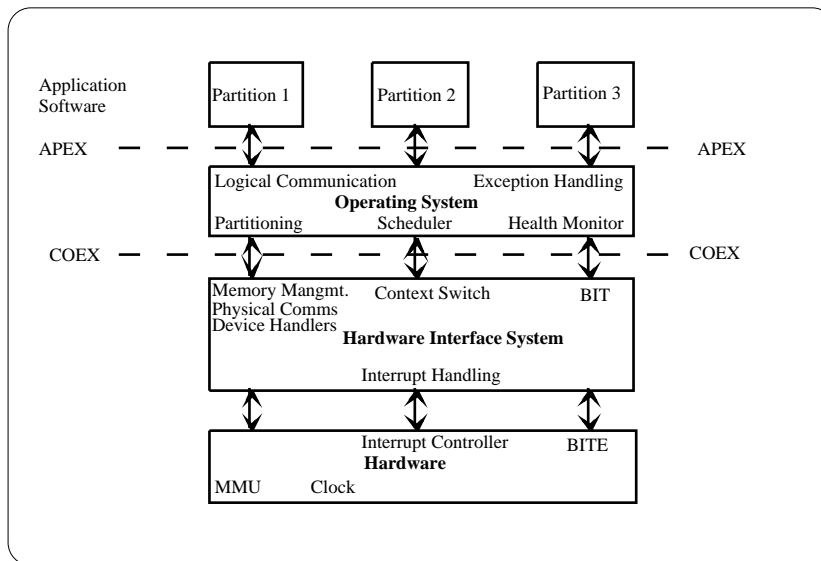


Figure 1 ARINC 653 System Architecture

Figure 1 shows the system architecture specified in ARINC 651 and 653 (Refs. 6 and 2). The architecture identifies two interfaces: APEX and COEX. The APEX interface is between the application software and the operating system while the COEX is the hardware interface used by the operating system. Apart from introductory material, ARINC Report 653 contains only the APEX specification. No specification of the COEX has yet been published.

At the APEX interface the division of the hardware into core modules and cabinets is not visible. However, the application software is divided into partitions. A partition is a processing environment which is self-contained in both memory and time. Since partitions are self-contained, they provide support for applications at different levels of safety-criticality. Partitions are mapped to a hardware processing module but may not be split between more than one module. A module may host more than one partition, but each partition has a fixed, predetermined allocation of time and memory, ensuring that the module is self-contained. Most of the attributes of the partition are determined during system configuration, including the memory and processing allocation and the channels which are used for communication between partitions.

The software within a partition may consist of multiple processes. Processes may be scheduled to execute periodically or aperiodically, with pre-emption of a lower priority process by a higher priority one. In contrast to a partition, most attributes of a process are controlled at execution time. The APEX interface specifies services for process creation, activation, delay and termination, communication between processes, synchronisation of processes to ensure orderly access to shared resources and the allocation of resources to processes.

An important part of the APEX specification covers health management, which includes both the detection of faults and the determination of the recovery action. Errors are distinguished both by their cause and the level at which they occur: thus memory violation is a process level error while power fail is a hardware module level error. An error during error handling is a partition level error. In ARINC 653 (Ref. 2), health management is carried out at module, partition and system levels. At each level, health management tables, which are determined by the system integrator, specify the recovery action. A possible recovery actions are to re-initialise a part of the system.

2.4 ASAAC

The Allied Standard Avionics Architecture Council (ASAAC) programme has been established by France, Germany, UK and USA to develop and demonstrate a standard for modular avionics. The architecture concept (Refs. 8, 9), is use a small set of common hardware modules with modular software. The software concept (Refs. 10, 11, 12) is illustrated in Figure 2.

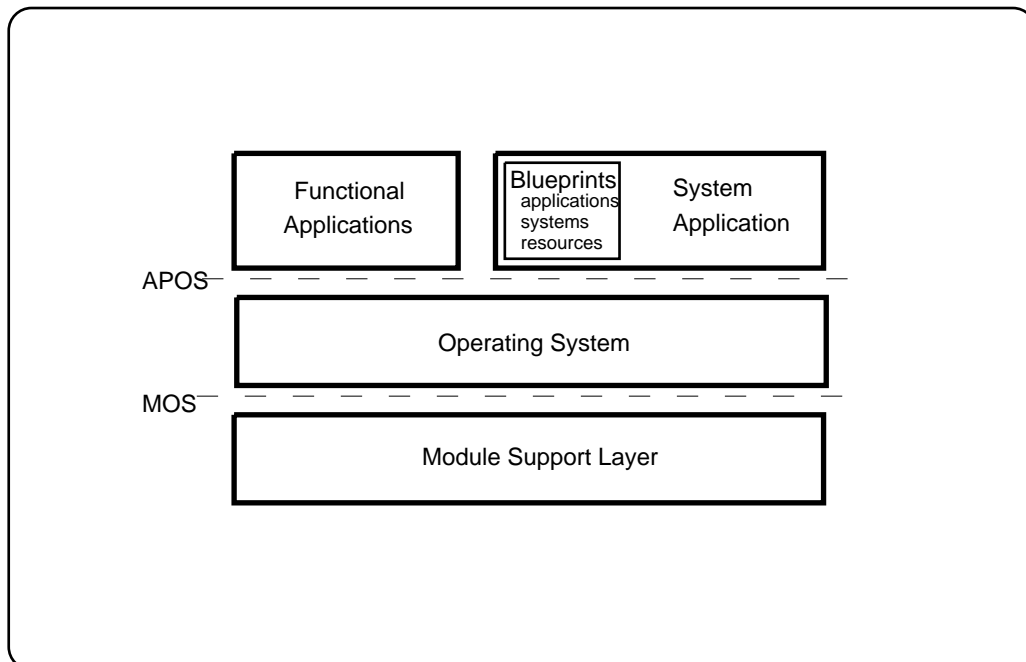


Figure 2 ASAAC Software Concept

Application programmes are written to the APOS interface. At this interface, the system is presented as a network of communicating software processes: the existence of a communication medium such as a data bus or of separate processing elements is not visible to the application software. Application processes communicate via virtual channels in a uniform way which does not depend on the locations of the processes. In this architecture, many properties of the system are determined by configuration. All the scheduling is determined in advance and specified at the configuration interface; data transfer also occurs at pre-determined times rather than as soon as the data is available. The configuration also specifies the mapping of processes to hardware processing elements, including alternative assignments in the event of faults. As a result of the extensive use of configuration, the APOS interface, which corresponds to the API of Section 2.2, is restricted to a few functions.

The objective of the ASAAC project is to produce a set of STANAGS for modular avionics: these will include standards for the operating system. According to Ref. 8, the working baseline for the operating standard was contained in the 1994 Harmonised Core Architecture Concept Definition. The current status and availability of this document is not known.

2.5 F22 Common Integrated Processor (CIP) Software

The F22 CIP is an existing hardware and software design with IMA features. It is significant in the context of this report both as a source of lessons learned in an actual project and because Hughes (Refs. 13, 14) are promoting elements of CIP as an open standard (see Section 2.6).

The infrastructure software of the CIP consists of the Avionics Operating System (AOS) and the Avionic System Manager (ASM). The AOS (Ref. 13) provides a set of services to allow multiple programs to share the resources of a single Data Processing Element (DPE). A single copy of the AOS runs on each DPE. The scope of the AOS is limited to an individual DPE; the ASM is responsible for co-ordination between multiple DPE's. Figure 3 illustrates the software interfaces.

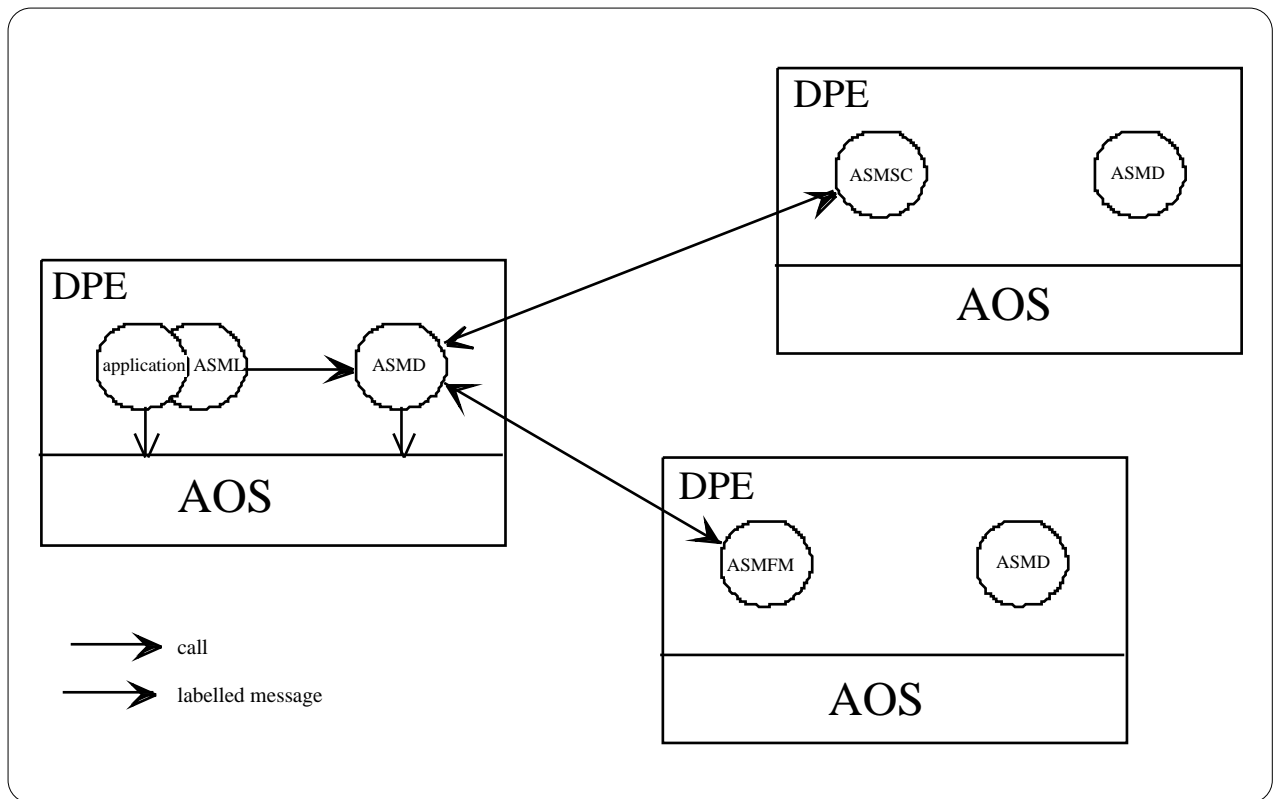


Figure 3 F22 AOS and ASM Interfaces

AOS supports an Ada runtime system: application running in the DPE and using the AOS services are Ada programs. The AOS services include:

- 1) I/O interface control, including parallel bus access, global bulk memory access, backplane signalling, interrupt access, test and maintenance bus access, module maintenance controller access, backend device access
- 2) program loading and control
- 3) process synchronisation
- 4) timer services.

The ASM consists of multiple Ada programs and a linkable interface layer:

ASMD	ASM Distributed: responsible for interfacing with other ASM elements. There is one instance on each DPE.
ASMFM	ASM file manager: manages Data Transfer Elements (DTE) and files in Global Bulk Memory (GBM). There is one active ASMFM in each CIP and one backup.
ASMSC	ASM System Controller: responsible for system and fault management. There is one active ASMSC in each CIP and one backup.
ASML	ASM Linkable: this is linked to the application program to provide the programmers API to the ASM functions.

The AOS includes some features which are for the use of the ASM only. These include program loading and control, configuration of parameters of the hardware module and error reporting.

Privilege Control Tables (PCT)

The rights of an application to call AOS services is determined by entries in a PCT. The PCT is a static table which is generated off-line. The table generation tool checks conformance with the system security policy. Each request for an AOS is checked in the table to ensure that the requesting process has the necessary access rights. If the check fails the calling program is terminated, or in pre-defined circumstances, may be passed an exception. The ASM uses an AOS service to perform PCT lookups to check requests for ASM services in a similar way.

Debug Support

The AOS includes features to support software debugging. The full debug environment provides symbolic debugging of Ada and support for a tool for real time debugging. The tactical debug environment supports a restricted set of capabilities.

2.6 Avionic Modular Component Acquisition (AMCA)

The Avionics Modular Components Acquisition (AMCA) program is a collaboration funded by MOD FS(Air). According to Ref. 15, the team for phase one consisted of Hughes Aircraft, EDS Defence Ltd, Smiths Industries and DRA. An evaluation of the Hughes common integrated processor (CIP) was carried out, with the overall goal of

developing IMA hardware and software. The CIP hardware and software was developed for the F22 project: the evaluation developed a sample application and benchmarked the processor performance. The performance of the CIP, which uses 25MHz Intel i960MX was found to be approximately equal to a 25MHz Motorola M68040. The evaluation concluded that the F22 architecture contained a full set of features for promoting integrity and reliability, which have been shown to be effective. However, it also concludes that further work on IMA is required before safety critical applications can be incorporated. No information on the progress of this project since the publication of Ref. 15 has been found.

It is anticipated in Ref. 14 that the CIP interface specification are to be publicly released and placed under the control of independent bodies for maintenance. This move is proposed as part of a plan to 'leverage (the) strategic investments in the CIP as an open system'. Plans to evolve the CIP architecture to allow technological advances to be incorporated are being prepared. If this strategy succeeds, CIP may become an architecture standard, defined by the software architecture and the communication models, with other physical realisation in future.

2.7 TARDEC (TACOM) Vetronics Real Time Operating System

The Vetronics (vehicle electronics) System Architecture (Ref. 16) is an architecture model developed by the US Army. A system consists of a number of vetronics stations connected by distribution networks for data, including audio and video. The system architecture specifies the standards which may be supported for different functions within the architecture. Since there are some choices in the standards specified, the system architecture is, in some respects, a framework; however, it does not have the same aims or generality as GOA (see Section 2.10).

The Vetronics System Architecture specifies that each processing element shall support:

- 1) the use of Ada for application programming, including support for a multi-tasking Ada runtime system
- 2) the Vetronics Real Time Operating System (VRTOS) services (Ref. 17)
- 3) the Vetronics Graphical User Interface (VGUI) services.

VRTOS is an operating system API standard for application programs written in Ada. The network protocol is required to comply with the User Datagram Protocol (UDP) service on Internet Protocol (IP). UDP provides task to task message transfer and is independent of the processing element on which the tasks reside. However, it is not clear that the VRTOS API abstracts completely from separate processing elements.

Application Interfaces

The VRTOS API provides primitive functionality for communications, mutual exclusion, synchronisation, timing, bit manipulation, memory operations, error logging and circuit

card identification. Service requests from each application program are processed independently.

The VRTOS should be implemented as a classical layered operating system where the lowest level is the kernel. Generally, the only uninterruptable operations that an operating system performs are those executed at the kernel level. This minimizes preemptive latency and makes the VRTOS more predictable and consistent in its behaviour. The kernel functions include a binary semaphore primitive, Ada task priority setting, interrupt service routine wrappers and vector installation.

The VRTOS memory utilities provide the application with basic memory operations. These operations include the ability to copy blocks of memory, read and write blocks of memory as well as a test and set, mutual exclusion primitive. These utilities are provided because the Ada programming language does not afford the easiest or fastest methods of doing these low-level operations.

The VRTOS error logging functions provide a facility for storing and retrieving software errors that may occur both in VRTOS and in application programs. This capability allows software to record details of errors or problems encountered during execution, especially in cases where propagation of an error status or an exception is not appropriate or even possible.

The VRTOS event functions provide the application with a synchronisation primitive. VRTOS events can be used to signal the occurrence of a synchronous or asynchronous activity. VRTOS events are to be used for synchronisation in a similar manner as semaphores, but behave differently from a semaphore. When using VRTOS events, several tasks can wait on an event, and when this event is set, all of the tasks are freed at the same time and made available to run. This differs from the behaviour with a semaphore where only a single task can be freed.

The VRTOS file services provides a standardised interface to mass media storage devices. Physical information concerning the types of devices and their location within the system are contained within the VRTOS configuration file. If a storage device is not directly accessible by a processor, then a request is sent to a processor which can access the storage device on the users behalf. It is transparent to the user whether the file services are performed by a local or remote processor.

2.8 POSIX

The Portable Operating System Interface (POSIX) standards (Ref. 3) is a set of UNIX-like standard operating system interfaces. The POSIX standards are being developed by a standards committee of the IEEE, under the direction of a joint committee of ISO and IEC. The responsible IEEE committee is called the Portable Application Standards Committee (PASC). The POSIX standards will become international standards. The standards are developed by a series of PASC subcommittees or projects, each dealing with one aspect

of the operating system. Table 2 shows a partial list of the PASC projects, including all those relating to realtime systems.

The different parts of the POSIX standard are in different stages of development. An operating system which claims POSIX compliance should specify which of the POSIX standards are met: not all of the standards in Table 2 are relevant to all applications. For example, a number of operating systems comply with 1003.1 and 1003.1b.

PASC Project Number	PASC Project Name
1003	Guide to the POSIX Open System Environment (OSE)
1003.1	System Interface
1003.1a	System Interface Extensions
1003.1b	Realtime Extensions
1003.1c	Threads
1003.1d	Additional Realtime Extensions
1003.1e	Security
1003.1h	Fault Tolerance
1003.1i	Fixes to 1003.1b
1003.1j	Advanced Realtime Extensions
1003.1n	Fixes to 1003.1, 1003.1b, 1003.1c, 1003.1i
1003.5	Ada Binding to 1003.1
1003.5b	Ada Realtime

Table 2 PASC Projects

The operating system services specified on 1003.1 include services, such as file access, which may not be required in realtime applications, though they can be useful for application development. Some systems, such as those based on a micro kernel, allow the functionality supported to be tailored to the needs of a particular application. The POSIX standards specify the operating system interface. Compliance with 1003.1 and 1003.1b is not, therefore, enough to ensure that an operating system is useful in realtime

applications, since a realtime operating system must meet performance as well as functional requirements.

2.8.1 ISO/IEC 9945-1 ANSI/IEEE Std 1003.1

The second edition of the POSIX standard (Ref. 3) has recently been issued. Some of the realtime features of this standard are described below. It should be noted that Ref. 3 specifies the C language application program interface. Ada language bindings to the previous version of the POSIX standard are now out of date. Future work is expected to produce a language independent specification of the POSIX API with bindings for the different programming languages.

Realtime Extensions

The standard (Ref. 3) includes an optional set of interfaces to support portability of applications with realtime requirements. The specific functional areas included are:

- 1) semaphores: a simple synchronisation primitive
- 2) process memory locking: provides a mechanism to lock program areas in memory, preventing transfer to secondary storage
- 3) memory mapped files and shared memory
- 4) priority scheduling
- 5) realtime signal extension
- 6) timers
- 7) interprocess communication
- 8) synchronised and asynchronous input and output.

Many of these features are extensions or realtime improvements to existing capability of historical implementations of UNIX.

The standard also defines an extension for threads, which are a form of task without the overheads of full processes. Annex G of Ref. 3, which is informative, describes performance metrics which could be used by a POSIX implementation to document the performance of the implementation.

Conformance

The POSIX standard does not describe a mechanism for determining conformance to the standard. Some parts of the standard are optional and the standard does not aim to specify all aspects of the operating system. It is unlikely that all aspects of the data exchange format or terminal i/o would be supported in a real-time operating system.

2.8.2 Profiles

A profile describes a combination of open systems standards which are specific to a particular application area or class of users, such as supercomputing, multiprocessors or

realtime systems. A profile for embedded realtime system might specify parts of the POSIX standard.

2.9 Ada 95: Real Time Extensions

Annex D of the Ada Reference Manual (Ref. 5) defines a number extensions to the Ada language to improve support for realtime programming. These extensions take the form of additional library packages and additional pragmas. A number of metrics are defined: these are documentation requirements. The following topics are covered.

Task Priorities	Priority Scheduling
Priority Ceiling Locking	Entry Queuing Policies
Dynamic Priorities	Pre-emptive Abort
Task Restrictions	Monotonic Time
Delay Accuracy	Synchronous Task Control
Asynchronous Task Control	Other Optimisations and Determinism Rules

Task Scheduling

In Ada83, a single scheduling policy was defined in the core language, based on pre-emptive scheduling of tasks with fixed priorities. This approach lacked flexibility and also, since some aspects of the scheduling could vary between implementations, did not maximise portability. According to the Ada95 rationale (Ref. 18), in Ada95 (Ref. 5) scheduling issues have been removed from the core language. The approach defined in Annex D subsumes and improves the Ada83 approach. In addition, a framework is created within which implementations can offer non-standard scheduling policy. The standard policy, which must be offered by all implementations which conform to the annex, is intended to be compatible with one of the scheduling policies specified in POSIX.

Tasking Restrictions

The section on Tasking Restrictions specifies a number of restrictions which can be used within pragmas to enable an implementation to offer a more efficient runtime system. Each restriction specifies a simplification to the Ada tasking model: if the restriction is used in a program, the compiler should check that it is obeyed. For example, one restriction prevents the declaration of a task within another task. When a particular combination of restrictions is in use, an implementation may substitute a runtime system which is simpler and therefore potentially more efficient than the standard Ada runtime system.

Low-level Facilities

The synchronous and asynchronous task control section specifies capabilities which can be used to implement higher-level synchronisation constructs. The synchronous task control takes the form of a semaphore type which can be used with protected objects to implement a suspend and resume mechanism.

The asynchronous task control mechanism is very low-level. A library package provides operations to suspend or resume a task at any time. These operations could be used to implement user defined task scheduling. However, according to Ref. 18 the interaction of asynchronous task control with other aspects of the Ada tasking model is not specified. The intention is that these features will be used with an Ada implementation which does not support most of the standard features.

Interaction Between Ada and an Operating System

According to the Ada rationale (Ref. 18), revisions to the tasking model in Ada95 compared to Ada83 are intended to allow greater use of RTOS products with Ada programs. Possible approaches to combining Ada with an operating system are described below.

- 1) The Ada runtime system is used instead of a conventional operating system. This approach is often used when Ada programs are targeted at bare-board computers. Its principle disadvantage is that operating system features such as debugging, program loading and monitoring which lie outside the Ada language are not available.
- 2) The Ada real-time system operates at a lower-level than the operating system. As an example of this approach, an Ada program consisting of a number of tasks could run as a POSIX process. The POSIX-compliant operating system schedules the processes while within each process, the Ada runtime system schedules tasks. Interaction between tasks in the same process uses Ada language constructs, but interaction between different processes must use operating system primitives. A disadvantage of this approach is a duplication of functionality at the two levels, with some desirable features such as memory protection not necessarily provided at the lower-level.
- 3) The Ada runtime system is implemented using the features of the operating system. In this approach, the Ada compiler generates calls to OS primitives in order to achieve Ada tasking functionality. For example, an Ada task could be implemented using POSIX threads. Unless the operating system conforms to a standard which is strong enough to ensure that the Ada tasking semantics are correct, this approach requires co-operation between the Ada compiler vendors and RTOS product vendors.
- 4) The functionality of an RTOS product is made available in Ada programs using an Ada API, but the interaction between Ada tasking and the operating system scheduling is not fully defined. This approach is clearly unsatisfactory. It is slightly more satisfactory if it is clearly stated that certain Ada constructs should not be used with certain operating system calls.

2.10 Society of Automotive Engineers (SAE) Generic Open Architecture (GOA)

The GOA Framework (Ref. 4) has been developed by the AS-5 GOA Task Group of the SAE. The framework is intended to classify interfaces needed in embedded systems, including military avionics systems. This classification of interfaces is intended to

encourage the use of open standards, such as POSIX (Ref. 3) or the ISO Open System Interconnection (OSI) standards within embedded systems.

The GOA Framework used the Space Generic Open Avionic Architecture (SGOAA), developed by NASA, as a starting point. However, it is intended that the GOA framework will apply in a different application. It is expected that the framework would be specialised for application to a particular domain such as military avionics.

Interfaces can be either logical or direct. A logical interface defines the content of the information exchanged while a direct interface provides the service to accomplish the data transfer. The GOA Framework identifies nine interface classes, four logical and five direct. The interfaces are defined with respect to a hierarchical system model, illustrated in Figure 4.

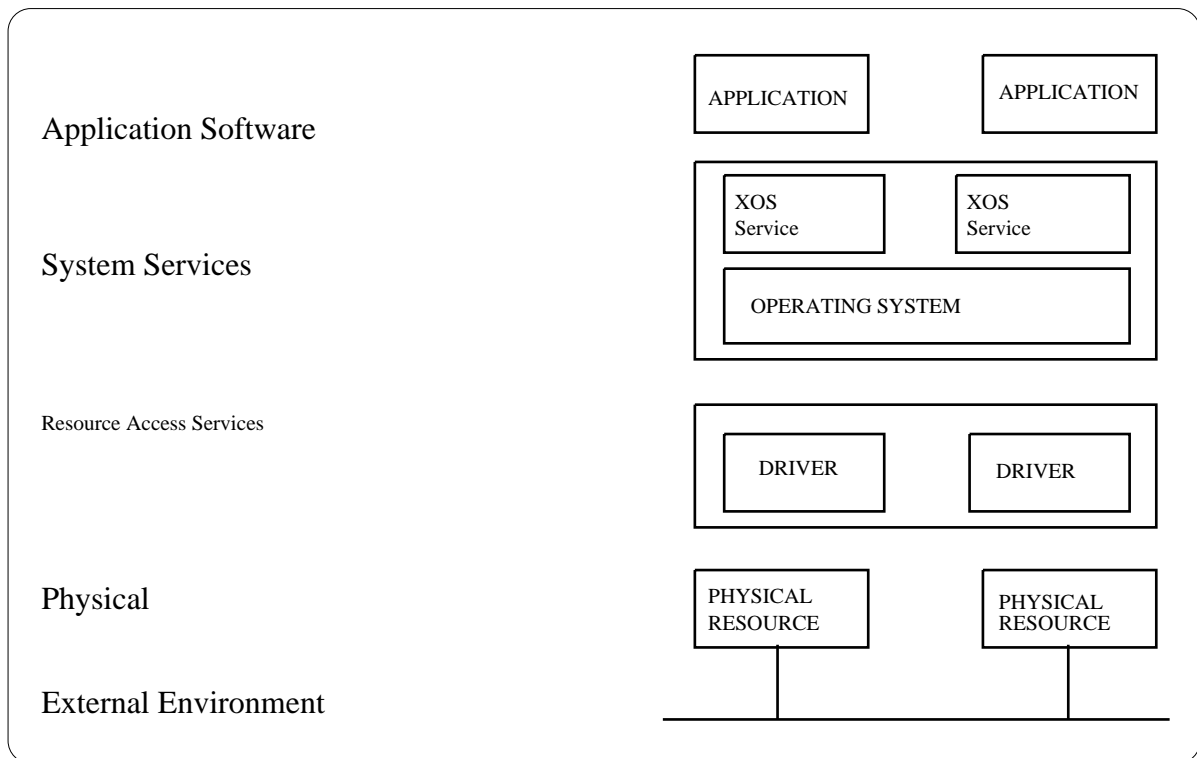


Figure 4 GOA Framework Architecture Model

The layers are application software, system services, resource access services and physical resources. The system services layer consists of two sub-layers: the operating system and the extended operating system services. To illustrate the approach, a partial list of the interface classes is described in Table 3.

4L	This class is logical interface between software applications to establish a data interchange interface or to request or transmit data. One requirement on this interface is to allow communication between applications on different processors.
4D	This class is direct interface from the application software to the system services, corresponding to the operating system API. Since the services in the system service layer and lower layers in the architecture model are provided for each processor, there is no requirements for the 4D interface to allow application software on one processor to access system services on another processor.
3X	This class consists of the direct interface between the operating system and the extended operating system services, within the system service layer. This interface is intended to support an expansion of the services provided by the operating system. The interface allows access to privileged operating system services, beyond those available to application software.
1L	This interface class covers the logical exchange of data between physical resources.

Table 3 Selected Interface Classes in the GOA Framework

2.11 Other Initiatives from the Society of Automotive Engineers (SAE)

The Avionics System Division (ASD) of the Society of Automotive Engineers (SAE) has established an Operating System Application Program Interface Working Group numbered AS-5B-2 to develop an Application Program Interface (API) standard (Ref. 19). The purpose of the standard is to 'collect avionics operating system application program interface requirements. The set of requirements shall be used to evaluate existing OS APIs (in particular POSIX) and determine the suitability of the interface based on cost, schedule, performance, security, and supportability tradeoffs.'

The latest version of the standard available (Ref. 19) is still far from complete. The requirements are organised into the following categories:

Synchronisation	Special Devices
Program Support	Non Operational Support
Task Control	Configuration
Error Handling	Fault Management
Data Security Authorisation	Data Conversion

Memory Management	Re-initialisation
Communication	Instrumentation
Timer Services	Built In Test
File Management	Bootup/Initialisation
Input/Output	

The relationship between the Ref. 19 and the GOA framework (Ref. 4) is not clear.

Real-Time Model Task (RTMT) Group

The RTMT Group of the AS-2A subcommittee of the AS-2 Interconnect Network Committee has circulated Ref. 20. The mission of the RTMT is to increase understanding of the communication requirements which are specific to a real-time environment. The task group liaises with other standardisation groups including ISO, NATO, IEEE, POSIX, JIAWG and NGCR (Next Generation Computing Resource).

The task group has compared a number of architecture models from a communication perspective. The starting point is the ISO Open System Interconnect (OSI) Reference Model (OSI/RM). Since previous work has shown that OSI/RM is not applicable to the requirements of real-time systems, the RTMT Group has proposed a different model. Other models are compared to the RTMT model. Although the comparison concerns the communication requirements, the models are of potential interest since they also place requirements on operating systems. Table 4 shows the model considered, excluding those which are mentioned elsewhere in this report.

Model	Description
NIST	The National Institute of Standards and Technology (NIST) is defining an Open System Environment (OSE) for inter-operability and portability of applications in distributed systems. The NIST defines a seven layer reference model. An Application Portability Profile (APP) has been defined which outlines applicable open standards for each interface.
TAFIM TRM	The Technical Architecture Framework for Information Management (TAFIM) is a Technical Reference Model (TRM) which has been defined by the Defense Information Systems Agency (DISA). The TAFIM TRM is an adaptation of the NIST model to the DoD's requirements. A profile of approved standards has been identified.
WWSS TRM	The Weapon and Warfare Support Services (WWSS) Technical Reference Model is a further refinement of the TAFIM TRM. Specific standards are identified in conforming systems.
C ⁴ I	The Command, Control, Communications, Computers and Intelligence (C ⁴ I) architecture has been developed to provide a foundation for system inter-operability. Three concepts of architecture are identified: the technical architecture includes standards and policies for building a system; the system architecture defines current systems and the migration to future systems; the operational architecture defines the integration of a system into the overall battlefield architecture. The technical architecture specifies the use of POSIX and the TAFIM model for information processing; communications are based on internet protocols.
GAM-T-103	This standard was developed in the 80s by the French MoD and industry. It specified a communications reference derived from the seven layer ISO model, but reduced to four layers. The user layer is the highest level where application programs produce and use data; below the user layer are transport, data link and physical.
ATM	Asynchronous Transfer Mode (ATM) is a telecommunication for multi-rate communication with efficient allocation of bandwidth.
SCI	Scalable Coherent Interface (SCI) is a communication protocol based on a shared memory paradigm with a single memory address space. SCI was developed for use in multiprocessor computers but is being considered for real-time applications. There is no specific communication model.

Table 4 Additional Models Considered in SAE AIR 4885

2.12 US New Attack Submarine (NSSN)

The US Navy is proposing, in Ref. 21, two techniques to reduce the software acquisition and lifecycle costs for the NSSN program:

- 1) use of commercial standards and COTS software
- 2) software reuse.

Ref. 22 describes the software reuse plan. No definitive information has been found about the systems architecture within which it is proposed to make use of commercial standards and COTS software. However, it has been reported in the ASSC Processing Working Group that:

- 1) a federated system architecture is proposed
- 2) standards will be applied to the interfaces between subsystems
- 3) COTS equipment will be used to implement subsystems, with a policy of subsystem replacement when obsolescence occurs.

If this is correct, it has implications for the evaluation of RTOS products for use within the subsystems. In general, the evaluation criteria will not require a uniform RTOS to be adopted system wide, since it is the interfaces between subsystems, rather than within a subsystem, which are to be maintained through the system life.

2.13 The Role of RTOS Standards in Evaluating RTOS Products

Four possible roles of the System Architecture and RTOS standards described above can be considered for evaluating commercial RTOS products.

- 1) An RTOS product could be evaluated for compliance with the standard. This role is not likely to be applicable to standards such as APEX or ASAAC which are based on a specific model of the system architecture, whose requirements are not likely to be satisfied by a general purpose RTOS product.
- 2) An RTOS product could be evaluated for compliance with some part of the requirements of a standard based on a system architecture model, for example so that the difficulty of using the RTOS product as a foundation on which to implement a system compliant with the standard could be evaluated.
- 3) Some of the standards contain requirements specific to the use of RTOSs for avionics applications. Since the RTOS products are aimed at a market which is wider than just avionics, a general evaluation of suitability for use in avionics application could be carried out using requirements taken from one or more standards.
- 4) Most of the standards are incomplete and unproven as yet. Only a minority of the standards are specific to military avionics. For both these reasons, the standards as well as actual RTOS products need to be evaluated for applicability to military avionics.

In the following sections, each of these roles is described and the standards which could be used in each role are noted. Figure 5 gives an overview of the possible relationships between the RTOS products and the standards.

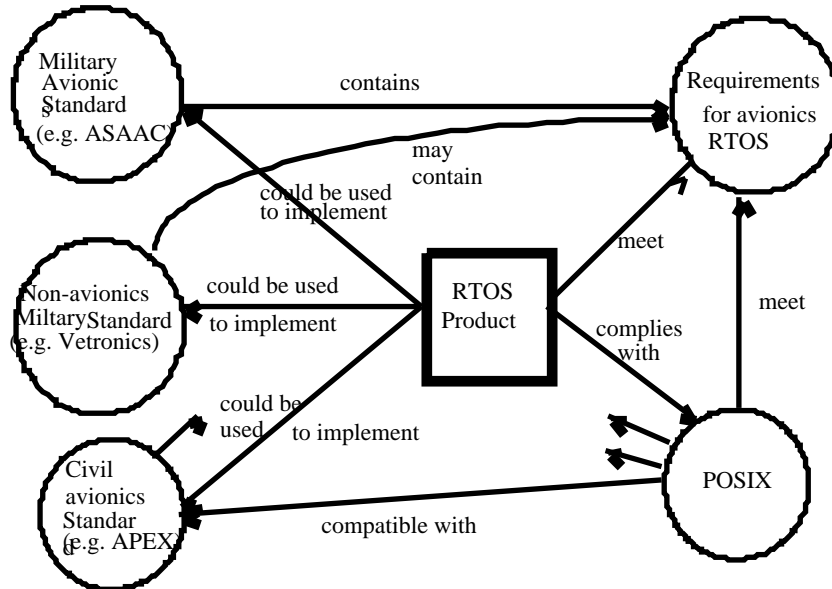


Figure 5 Relationship of RTOS Products and Standards

2.13.1 Compliance of RTOS Products with Standards

Only the POSIX standard can be used directly to assess general-purpose RTOS products, since standards such as APEX, ASAAC and VETRONICS are closely linked to detailed architectural models. As noted above, only parts of the POSIX standard are relevant to real-time systems. The criteria for compliance with POSIX are complex, since some parts of the POSIX standard are optional.

It should also be noted that compliance with relevant parts of the POSIX interface standard is not sufficient to ensure that an operating system will be suitable for real-time applications. In particular, the POSIX scheduling policies do not ensure that the behaviour of the operating system is fully deterministic. POSIX compliant OS products which claim to be suitable for real-time applications are often based on a fully pre-emptive operating system kernel, in order to ensure, for example, that the interrupt latency is bounded.

2.13.2 Using RTOS Products in Standard Compliant Systems

Standards such as APEX and ASAAC are based around a detailed systems architecture; as a result these standards place requirements on the operating system component of the architecture which are not likely to be met by a general purpose operating system. It is not useful, therefore, to evaluate an RTOS product against the overall requirements of APEX or ASAAC. However, as well as requirements specific to the architecture, the standards also contain more general requirements which might be met by a general purpose operating system. It is possible that an RTOS product could be used in the

implementation of a APEX-compliant system, for example. To evaluate the value of this approach, the requirements of APEX which are directly implemented by the RTOS could be investigated and compared with the requirements which would need to be implemented by an additional software layer, extending the RTOS product.

This form of evaluation could be carried out within the GOA framework (Section 2.6), which classifies the interfaces within a systems architecture. In this case, the functionality of POSIX in general, a particular POSIX-compliant RTOS product or of another RTOS product could be placed within the GOA classification.

2.13.3 Evaluating RTOS Products against General Avionics Requirements

Standards such as APEX, ASAAC and SGOAA which have been developed within the aerospace sector contain general requirements which determine the suitability of an operating systems for use in avionics. The requirements of this nature could be collated and reconciled and used to evaluate RTOS products, independently of the architectural model and the other requirements of the standards. This approach is similar to that being taken in SAE ARD 50067 (Ref. 20), but no completed version of this document has been obtained yet. Section 3 below describes general requirements for evaluating RTOS products for use in avionics but does not compare the requirements with those contained in the standards.

It is noted in Section 2.13.1 that not all POSIX implementations are real-time. Assuming that a POSIX implementation is suitable for general real-time use, the requirements specific to avionics could be used to investigate the suitability of POSIX for avionics applications.

2.13.4 Evaluating the Standards for Applicability to Military Avionics

Some of the standards, such as APEX and VETRONICS described in this section have been developed in sectors other than military avionics. The evaluation of these standards for use in military avionics should be considered. However, this evaluation would need to encompass the system architecture proposed in each standard rather than being restricted to the operating system, since the requirements placed on the operating systems are within the context of a system architecture.

The best way to carry out this evaluation is not clear. The difficulty is that while the military avionics architecture proposed in the ASAAC project differs from that proposed for civil avionics by ARINC, both architectures remain unproven. Each architecture represents a consensus formed by a group of experts in the two domains; an independent framework for assessing the applicability of the each proposed architecture in the other domain would need to be established.

2.14 The Standardisation Process

It is noted that a large number of standards relating to software and communications have been proposed. Standards arise in a variety of ways and in response to various sorts of pressure, both commercial and technical. A review of the process leading to a successful

standard may be useful to the work of the ASSC in relation to the standardisation of operating systems for avionics applications. Some of the factors to be considered include:

- 1) how do standards arise and how long does it take to achieve standardisation?
- 2) how do technical and commercial issues interact in the formation of a standard?
- 3) how does the process of standardisation affect the technical content of a standard?
- 4) what proportion of standards are adopted into use and what factors influence the take up of a standard?

One example of a standard with similarities to an RTOS standard is the Portable Common Tool Environment (PCTE) standard (Ref. 23). PCTE is an interface specification for the integration of different software engineering tools, building on operating system and database concepts. The standard was adopted by ISO/IEC in 1995; it has three parts, including language bindings for both Ada and C. The degree of take up of PCTE by the suppliers and users of software engineering tools is unclear.

A review of the process of standardisation for standards such as PCTE, or other standards such as those described in Section 2.11, with similarities to possible future operating system standards, could be considered so that future work by the ASSC can take account of the lessons learned

3 EVALUATION OF RTOS PRODUCTS FOR AVIONICS APPLICATIONS

In this chapter, the evaluation of RTOS products is considered. The first section reviews the reasons for considering the use of an RTOS product, since the potential advantages of using an RTOS product provide the motivation for evaluating candidate products. In Section 3.2, the influence of the system architecture choice on the evaluation of RTOS products is reviewed. Six approaches to the choice of an operating system are identified. The approach chosen determines the nature of the criteria which should be used to evaluate a candidate RTOS product. The range of possible evaluations is too great for detailed criteria to be established for all possible evaluations, but general criteria for evaluating RTOS products provide a starting point. Such criteria are given in Sections 3.3 to 3.5.

3.1 Why Use an RTOS

To evaluate operating system products, it is necessary to understand the potential advantages of using an operating system product within a real-time system. Off-the-shelf real-time operating systems are not used in many existing avionics systems, which are based instead on a simple bespoke runtime system, such as a cyclic scheduler. The advantages of using a commercial RTOS rather than a bespoke approach are:

- 1) greater functionality, for example for process communication and synchronisation
- 2) improved hardware independence
- 3) reduced development costs, compared with a bespoke development of equivalent functionality

- 4) better development and testing capabilities.

Although the use of an off-the-shelf RTOS product would obviously reduce development costs, it has been argued (Ref. 24) that other costs, in particular certification costs for safety related systems, can increase with the use of an RTOS product. This point is not developed further here.

3.2 Different Approaches to the Choice of an RTOS

There are different approaches to the choice of an RTOS and the approach chosen affects the evaluation of any RTOS products which are being considered. The six approaches identified are:

- 1) use a project bespoke RTOS, as exemplified by the F22 project
- 2) use a standard OS interface such as APEX (ARINC 653) or ASAAC
- 3) use a non-commercial off-the-shelf RTOS, such as RTEMS
- 4) use a commercial RTOS, such as VxWorks, possibly with POSIX compliance
- 5) use an Ada run time system, with real-time functionality
- 6) adopt a federated system architecture which allows each subsystem to determine which RTOS to use.

The example of the F22 project is not altogether applicable, since this project did not use an RTOS product. However, the approach of developing a single RTOS for use throughout the F22 avionics differs significantly from previous bespoke approaches. Moreover, although the F22 operating system was not developed in conformance to any standards, it does include some of the features of emerging standards such as ASAAC. The importance of the F22 example, is that the system is in an advanced stage of development. Alternative approaches such as APEX and ASAAC have not yet been used in a real system. Further information about the F22 operating system is given in Section 2.5.

The choice of one of these approaches depends strongly on the system architecture chosen. For example, the choice to conform to ARINC 651 determines whether APEX will be used. As noted in Section 2.13, none of the commercial RTOS products is likely to be fully conformant to APEX, but a commercial RTOS could be used to implement an APEX compliant interface. If this approach is chosen, the candidate RTOS products would need to be evaluated against the APEX standard, in the way described in Section 2.13.2.

The US Army RTEMS operating system is an example of software which is off-the-shelf, but which is not a commercial product. The choice of RTEMS or another non-commercial system is included as a separate approach because it is likely to result in a very different evaluation result, compared to a commercial product, especially against the commercial criteria given below.

The Ada programming language can be used with an RTOS product. However, it is also possible to use the Ada run-time system instead of an RTOS, as described in Section 2.9. In this approach, the Ada run-time system would be evaluated using similar criteria to those applied to an RTOS product.

The final choice listed above is to select a federated architecture. This is distinguished as a separate approach since in a federated architecture application software is hosted on separate subsystems, rather than on the system as a whole. As a result, the same API is not required in all subsystems, as it is in a system based on ARINC 651/653. While there may be reasons for preferring a single operating system choice, in principle the choice of operating system can be left to the subsystem supplier, rather than to the system integrator. Section 2.12 above describes a particular approach to systems with federated architectures.

3.3 Types of Criteria for Evaluating an RTOS Product

Criteria for evaluating an RTOS product may be functional or non-functional. The functional criteria concern the behaviour of the operating system and the facilities provided to the application programmer. Table 5 shows the categories of functional criteria. General criteria in each functional category are given in Section 3.4.

Category	Description
Scheduling	Different supported algorithms for sharing the processing resource between concurrent activities (processes, tasks, etc.)
Communication	Exchange of data between processes
Synchronisation	Co-operation of processes, for example for mutual exclusion
Interrupt Handling	Response to externally generated interrupts
Time Management	Interval and absolute timing
Memory Management	Allocation and protection of memory
Supported Interfaces	Communications and other interface devices which can be supported
Supported Processors	Processors and boards on which the OS can be hosted

Performance	Response time of operating system functions
Health Monitoring	Detection and management of fault conditions
Application Languages	Languages which can be used for application programming

Table 5 Functional Criteria Categories

The non-functional criteria concern wider issues including commercial, safety and supportability. Categories for these criteria are shown in Table 6. General criteria for each non-functional category are given in Section 3.5.

Category	Description
Support and Maintenance	Availability of expert support for advice and for the correction of detects found in use
Portability	Transfer of applications or of the OS to different hardware
Standardisation	Compliance with an OS interface standard
Supplier Commercial Stability	Assurance that the producer organisation will continue to exist and to support the product
Safety Certification	Level of acceptance of safety by a regulatory body
Security Certification	Level of acceptance of security by a regulatory or advisory body
Development Support	Facilities available to application developers

Table 6 Categories of Non-functional Criteria

Development support is includes as a non-functional criteria since it does not concern the functionality of the product as an operating system.

3.3.1 General Criteria: Avionics Data Management

Instead of the technically-oriented functional and non-functional categories described above, more general criteria could be proposed based on the management of data within the avionics system. Here, we regard the function of an avionics system to be to process the data generated by the sensors and to generate data for output devices, including the pilot displays. Data processing is supported by a 'data management framework', which includes operating system and data communication functions. The framework does not presuppose the architecture to be used in future avionics systems. The criteria are:

- G1 The data management framework allows the data to be processed at a location in the system which is optimal with respect to the user's requirements.
- G2 The data management framework maintains the security of data and does not introduce any hazards to safety.

One of the drivers for the design of future military avionics systems, which may require radical architectures to be adopted, is the development of ever more powerful sensors. For example, in an aircraft with two independent imaging sensors such as Radar and IR, the avionics must fuse the two data streams to provide the pilot with a unified situation picture. This suggests that sensor data should be transferred in an 'raw' form to a central location. On the other hand, other system pressures, such as the weight and maintenance costs associated with cabling, suggest that processing should be distributed to the sensors, so that the sensors become smart and only processed data is transferred out of a sensor. It is likely, therefore, that the data management framework in future systems will need to support flexibility in the association between the sources of data and the location at which it is processed.

Although these general criteria could be useful in future, they cannot be used directly to evaluate RTOS products; therefore, they are not considered further in this report.

3.4 Functional Criteria

There may be more than one realisation of each operating system function. For example, inter-process communication can be by message passing, which is either unbuffered (as in an Ada rendezvous), uses single place buffering (such as a mailbox) or fully buffered (as in a UNIX pipe). Alternatively, shared memory can be used for inter-process communication. The communication can be synchronous or asynchronous, and may or may not have an associated timeout or acknowledgement. It is not desirable for functional evaluation criteria to specify that all the alternatives should be supported, since this increases the size and complexity of the operating system. However, it is also difficult to select particular alternatives at the expense of others. To avoid this problem, the criteria given here are as general as possible.

3.4.1 Scheduling

- F1.1 The OS should provide a mechanism to allocate processing resources to application processes.

- F1.2 If the allocation is not fully deterministic, then facilities should be provided to prioritise some processes. A mechanism of priority inheritance should be supported to minimise the delays caused to a high priority process by a lower priority process.
- F1.3 Periodic invocation of processes should be supported, with the capability to detect the non-completion of a process before the end of the period, unless this can be checked statically.

Notes: Since the most important requirement in avionics applications is for deterministic behaviour, it may not be necessary to have a full complement of dynamic scheduling capabilities: see, for example, the ASAAC approach (Section 2.4).

3.4.2 Communication

- F2.1 The OS should support inter-process communication of sufficiently large bandwidth and sufficiently small latency.
- F2.2 The communication mechanism must be robust, so that corruption of data is prevented.
- F2.3 The non-completion of a communication should be detectable, for example by means of a timeout mechanism.
- F2.4 The communication mechanisms should minimise the information about the process receiving the data which is required to be built into the producer process; broadcast mechanisms which allow the receiver to determine the applicability of data should be supported.

Notes: Inter-process communication using shared memory is not robust unless the access to the shared memory can be shown to be non-overlapping.

3.4.3 Synchronisation

- F3.1 If a dynamic scheduling scheme is supported, there should be a mechanism for synchronising processes.
- F3.2 A means to ensure the absence of deadlock should be supported.

Notes: Synchronisation is closely related to both scheduling and communication. If processes are scheduled on a fixed schedule, then separate synchronisation mechanisms are not required. Unbuffered point-to-point communication mechanisms (such as an Ada rendezvous) can be used to provide synchronisation.

3.4.4 Interrupt Handling

- F4.1 A means to program an interrupt response should be supported, with priorities allocated to the different interrupts.

F4.2 The time when interrupts are disabled should be minimised; it must be possible to determine the maximum length of time for which interrupts can be disabled.

F4.3 The ability to detect the existence of an unserved interrupt is desirable.

Notes: It is not desirable for a real-time OS to disable interrupts whenever an OS service routine is called.

3.4.5 Time Management

F5.1 The OS should provide time functions to application programs, including absolute times and interval times. Both the time resolution (the minimum interval between times) and the range of the time interval must be sufficient for the application.

F5.2 Timed activation of processes should be supported. If dynamic scheduling is used, a function to delay a process may be required.

F5.3 The OS should interface to timer hardware.

Notes: The process delay function may not be required in a system which only requires cyclic scheduling.

3.4.6 Memory Management

F6.1 The OS should provide a means to allocate the physical memory to the application process.

F6.2 A means of preventing (or detecting) access to memory by a process other than the one to which the memory is allocated should be provided.

F6.2 If dynamic allocation of memory is used, the OS should ensure that exhaustion of available memory can be detected. The maximum amount of memory that can be lost, if any, through fragmentation should be defined.

Notes: Criteria F6.2 can be disputed. Some commercial RTOS products may choose to omit memory protection either to support hardware without memory management hardware or to provide faster response times at the expense of less protection.

3.4.7 Supported Interfaces

F7.1 The OS should support an interface to as many as possible of the interfaces required in a particular application.

F7.2 The OS should provide a defined interface for the addition of device drivers.

Notes: It may be possible to add additional hardware interfaces by application level programming, however this approach is subject to some limitations, depending on the nature of the OS. For example, an interface supported only by an application program cannot be used with the built-in I/O facilities.

3.4.8 Supported Processors

- F8.1 The OS should support a wide range of popular processors.
- F8.2 The OS should support the use of all features of the supported processors, including memory management units, input/output lines, hardware interrupts, coprocessors and counter timer hardware.
- F8.3 The OS should support multi-processor hardware.

Notes: Support for multiple processors (F8.3) is not always required.

3.4.9 Performance

- F9.1 The performance of the OS should be deterministic: sufficient data should be provided to determine the time required for, or the delay caused by, all OS functions in all circumstances. Some times may be specified as ranges (minimum to maximum), however an explanation of the cause of the variation (e.g. increases with the number of processes) should be given.
- F9.2 The maximum interrupt latency and the context switching time should be specified.
- F9.3 In a priority based scheduling system, it should be possible to transfer control to a higher priority process even when the current process is executing an OS function (this is called pre-emption). Any exceptions to this should cause a delay of a defined and bounded duration.

3.4.10 Health Monitoring

- F10.1 The OS should provide a means to monitor for, and to control the response to, defined failures in the system. The classes of failures considered should include hardware, interfaces and application software. The possible responses should include re-initialisation or re-configuration.
- F10.2 The greater the degree to which the OS API hides the failure condition, the greater the requirement to build-in the management of the failure.

Notes: It is not necessary for the health monitoring to be built into the OS, providing the OS provides the interfaces to allow health monitoring by an OS application: criteria F10.2 attempts to express this. Health monitoring cannot be handled completely at the single processor level: for example, the response to processor hardware failure must be handled by another processor.

3.4.11 Application Languages

- F11.1 The OS functions should be accessible through an Ada language API.
- F11.2 The relationship between the tasking constructs, the Ada runtime and the RTOS must be clearly described.
- F11.3 The OS should support application programming in a mixture of languages.

Notes: If OS functions are accessible from Ada which can cause data to be transferred between tasks or which can change the currently executing process, there is a potential clash with the facilities in Ada for these purposes. Ada constructs such as tasking should not be considered to be compatible with the OS unless the criteria F11.2 is satisfied. Although Ada is the MOD's preferred programming language, mixed language support (F11.3) may be required for software reuse, including the use of COTS software products.

3.5 Non-Functional Criteria

3.5.1 Support and Maintenance

- N1.1 The OS vendor should provide technical support to enable the application developer to identify and obtain fixes to problems.
- N1.2 The OS vendor should guarantee the availability of support for a specified period, which should be at least sufficient to cover the entry into service of the equipment. In the event that the OS vendor is taken over, the guarantee should apply to the purchasing company.
- N1.3 An arrangement for obtaining continuing support, such as access to the documentation and source code of the OS, should be agreed for the period following the expiry of the support agreement or in the eventuality of the commercial failure of the supplier.
- N1.4 The OS vendor should specify the period during which the hardware platform used for the project will be supported by future RTOS versions.

3.5.2 Portability

- N2.1 The OS should be implemented in a way which facilitates portability to new processors and provides the greatest confidence that future processors will be supported.
- N2.2 The OS should be configurable for different variants of supported processors.
- N2.3 The OS should be configurable for different designs of processing board based on one of the supported processors.

3.5.3 Standardisation

- N3.1 The extent of compliance with OS interface standards should be specified. The way that the conformance has been assessed should be described, including the degree of independence between the vendor and the agency assessing compliance.

3.5.4 Supplier Commercial Stability

- N4.1 The OS vendor company should satisfy criteria for commercial stability.

3.5.5 Safety Certification

- N5.1 The highest safety integrity level to which the RTOS has been accepted by a safety authority should be specified and should be sufficient for the proposed application.
- N5.2 The information offered by the OS vendor to assist is safety certification should be specified. The required information includes the process used in the RTOS development.

3.5.6 Development Support

- N6.1 The OS should be included in a capable development environment.
- N6.2 The OS development environment should support software testing and debugging, with the capability to monitor the execution of the software on the target hardware.
- N6.3 The OS development environment should support the analysis or measurement of performance of applications

4 INFORMATION AVAILABLE TO EVALUATE RTOS PRODUCTS

4.1 Introduction

In this section, the availability of the information necessary to evaluate candidate RTOS products is considered. The evaluation is to determine the suitability of the RTOS for use in avionics applications, taking account of standards relating to operating systems. It has been shown in Section 2 that there are a range of standards relating to operating systems, which differ widely in scope. As a consequence, there is more than one possible evaluation of an operating system which could be carried out, depending on the assumptions made about the systems architecture within which the RTOS is to be used. Given this difficulty, the evaluation criteria presented in Section 3 are general and wide in scope, covering both functional and non-functional properties.

The approach taken in this section is to consider the availability of information for an evaluation of an RTOS product against the criteria of Section 3, without specific architectural assumptions. The candidate RTOS products of ASSC/330/2/136 (Ref. 1) are not considered separately: instead, the following classes of RTOS are considered:

- 1) a commercial RTOS, exemplified by VxWORKS
- 2) a non-commercial RTOS, exemplified by the US Army's RTEMS.

The assessment of the information available is based on the information obtained during the production of Ref. 1: no further data about these operating systems has been obtained during the production of this report.

The assessment adds three further classes of operating system, considering the information available on:

- 1) POSIX
- 2) Ada 95 realtime extensions
- 3) F22 AOS.

In these cases, the objective of an evaluation would be different from that for actual RTOS products such as VxWORKS. For POSIX, the main focus is the suitability of POSIX compliant operating systems for avionics applications. This also needs to be evaluated for the Ada95 realtime extensions, though additional issues also arise. The F22 AOS is included since, as noted in Section 2.5 and 2.6 it is being proposed as a standard and it has been made available to the MOD for evaluation.

4.2 Commercial RTOS: VxWORKS

4.2.1 Available Information

Documentation provided by the suppliers of commercial operating systems covers the functionality of the operating system, the API and the functionality of the development environment. It should be noted that a charge is usually made for documentation: this cost has not been investigated during this study.

The documentation covers the different products produced by the supplier. VxWORKS is the core operating system supplied by Wind Rivers; a number of operating system extensions are available separately: two extensions for VxWORKS are for board-level portability and memory management. Products which support development may be supplied and documented separately. The development environment supplied by Wind Rivers is called Tornado; an operating system simulator called WindView is also available.

4.2.2 Comparison with Information Required

The estimated availability of information for evaluation against the functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Scheduling	Extensive information available from the supplier.
Communication	

Synchronisation	
Interrupt Handling	
Time Management	
Memory Management	Information available from supplier, but the limitations of this extension may need to be investigated in practice.
Supported Interfaces	Some information available from the supplier.
Supported Processors	
Performance	The extent of the information is uncertain. It is likely that independent experiments would be required.
Health Monitoring	The extent of the information is uncertain.
Application Languages	Some information available from the RTOS supplier, but additional information may be required from compiler suppliers.

Table 7 Comparison of information required

The estimated availability of information for evaluation against the non-functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Support and Maintenance	Additional information required.
Portability	Some information available.
Standardisation	Some information available.
Supplier Commercial Stability	Additional information required.
Safety Certification	
Security Certification	
Development Support	Extensive information available from the supplier.

Table 8 Estimated availability of information for evaluation against the non-functional criteria

4.3 Non-commercial RTOS: RTEMS

4.3.1 Available Information

Both documentation and source code for the RTEMS operating system is available from the RTEMS Home Page (Ref. 25). The documentation includes the RTEMS Brochure, the user guide for application programming in C and Ada, a user guide supplement for each supported target processor and for the UNIX-based simulator. The development environment is based on tools from the GNU software project, which include varying amounts of documentation.

4.3.2 Comparison with Information Required

The estimated availability of information for evaluation against the functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Scheduling	Extensive information available.
Communication	
Synchronisation	
Interrupt Handling	
Time Management	
Memory Management	
Supported Interfaces	
Supported Processors	
Performance	
Health Monitoring	The extent of the information is uncertain.
Application Languages	Extensive information is available.

Table 9 Estimated availability of information for evaluation against the functional criteria

The estimated availability of information for evaluation against the non-functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Support and Maintenance	Additional information required.
Portability	Some information available.
Standardisation	Additional information required.
Supplier Commercial Stability	
Safety Certification	
Security Certification	
Development Support	Extensive information is available.

Table 10 Estimated availability of information for evaluation against the non-functional criteria

4.4 POSIX

4.4.1 Available Information

The only immediately available information is the POSIX standard (Ref. 3). Additional information could be obtained from the wider literature or from the suppliers of POSIX compliant OS products. In particular, a supplier survey would be required to establish the take-up of different aspects of the standard.

4.4.2 Comparison with Information Required

The estimated availability of information for evaluation against the functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Scheduling	
Communication	

Synchronisation	Extensive information available in the standard.
Interrupt Handling	
Time Management	
Memory Management	Not applicable to the standard. Additional information could be obtained from a supplier survey.
Supported Interfaces	
Supported Processors	
Performance	
Health Monitoring	
Application Languages	Only the C interface is covered in the standard; information about Ada support could be obtained from product suppliers.

Table 11 Estimated availability of information for evaluation against the functional criteria

The non-functional criteria are applicable to POSIX to some extent, however none of the information required to evaluate the POSIX standard against these criteria is readily available.

4.5 Ada95 Realtime Extensions

4.5.1 Available Information

The readily available information about Ada is Ref. 20 and Ref. 21. Further information could be obtained from the suppliers of Ada compilers.

4.5.2 Comparison with Information Required

The estimated availability of information for evaluation against the functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Scheduling	

Communication	Extensive information available in the Ada standard.
Synchronisation	
Interrupt Handling	
Time Management	
Memory Management	The Ada standard does not provide information on these criteria; further information would be required from Ada compiler suppliers or from other sources.
Supported Interfaces	
Supported Processors	
Performance	
Health Monitoring	
Application Languages	

Table 12 Estimated availability of information for evaluation against the functional criteria

The estimated availability of information for evaluation against the non-functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Support and Maintenance	Additional information required.
Portability	
Standardisation	Information available.
Supplier Commercial Stability	Additional information required.
Safety Certification	

Security Certification	
Development Support	

Table 13 Estimated availability of information for evaluation against the non-functional criteria

4.6 F22 AOS

4.6.1 Available Information

The information available about the F22 AOS is limited to Ref. 13. Other documentation will have been produced by Hughes, for example to describe the Avionics System Manager (ASM), but its availability is not known. It is likely that further information has been made available to participants in the ACMA project (Ref. 15).

4.6.2 Comparison with Information Required

The estimated availability of information for evaluation against the functional criteria is shown in the following table.

Category	Availability of Information for Evaluation
Scheduling	Extensive information available.
Communication	
Synchronisation	
Interrupt Handling	
Time Management	
Memory Management	
Supported Interfaces	Some information available.
Supported Processors	
Performance	The extent of the information is uncertain.
Health Monitoring	

Application Languages	
--------------------------	--

Table 14 **Estimated availability of information for evaluation against the functional criteria**

The non-functional criteria are applicable to the F22 AOS but no information is available for an evaluation against these criteria.

5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

5.1.1 Relationship of Standards and RTOS Products

- 1) The OS standards surveyed in Chapter 2 vary in their scope and purpose. The POSIX standard specifies the functionality of an operating system which resembles UNIX and does not make many assumptions about the system architecture. On the other hand, standards such as ASAAC and Apex are based around a specific system architecture. A third class of standard, including GOA, describe system architecture frameworks within which interfaces standards can be classified. These different classes of standards are not directly comparable.
- 2) POSIX is the only standard with which an RTOS could comply directly. Even in this case, compliance is not simple, since POSIX includes functionality which may not be required in a realtime application and the POSIX standard also includes optional features. As well as evaluating RTOS products against the POSIX standards, it would also be possible to evaluate the functionality of POSIX against the requirements of avionics applications. Other properties of the POSIX standard could also be evaluated, such as whether it is strong enough to ensure application portability in a realtime environment.
- 3) Ada95 and the realtime extensions defined in the Ada Reference Manual have been regarded as a form of OS standard in this report. The relationship between Ada and RTOS products is complex. Since the Ada runtime system can substitute for an RTOS product, the Ada runtime system could be evaluated as an operating system for avionics applications. An RTOS product can also be used to implement the Ada runtime system, though it is not known how much this approach has been followed. Finally, an RTOS product may offer an interface to Ada applications programs, though in this case the interaction between the Ada runtime system and the RTOS needs to be defined.
- 4) RTOS products could be used within an implementation of the application interface, such as APEX, within an integrated architecture. It is not likely that an RTOS product would offer all the functionality required at such an interface, but the extent of the requirements met and the detailed compatibility between the RTOS and the requirements of the standard could be evaluated.
- 5) The functionality of an RTOS product could be classified within a framework standard such as GOA. The RTOS API functionality could be evaluated against the requirements for the interfaces from the application program to the system service level. When some other components in the framework have been specified, as in the US Army's vetronics system standard, the lower levels in the framework could also be evaluated against the hardware and interfaces supported by an RTOS product.

5.1.2 Criteria for Evaluating RTOS Products

- 1) It is difficult to produce a single set of criteria which cover the wide range of different evaluations which could be made of an RTOS product against the different classes of standard. However, general criteria covering both functional and non-functional properties have been proposed.

- 2) In each aspect of operating system design, a number of different solutions exist. For example, inter-process communication can use buffer or synchronous messages or shared memory. The evaluation of RTOS products should not be based on a maximal set of solutions (message passing, shared memory) but on more abstract requirements.
- 3) A more detailed evaluation of an RTOS product can only be carried out in the context of a defined system architecture. Even non-functional properties such as safety may depend on the system architecture, since in a federated architecture subsystem diversity may be used to reduce the integrity required of the RTOS product.

5.1.3 Information Available to Evaluate RTOS Products

- 1) Readily available information about RTOS products covers most functional criteria. Evaluation criteria relating to performance, health monitoring and Ada support may not be satisfactorily covered by the available information.
- 2) Less information is available to evaluate RTOS products against the non-functional criteria. Particular difficulties are light with respect to the evaluation of supplier stability, long term supportability and safety.
- 3) The availability of information to evaluate different aspects of the Ada realtime extensions was also considered. Little information is available at present, but it could be obtained from Ada compiler vendors.

5.2 Recommendations for Further Evaluation of RTOS Products

- 1) The priority of the different evaluation objectives in relation to the different standards and system design approaches should be established, taking account of the role of the ASSC Working Group to advise the MOD on matters requiring co-operation between industry and the MOD. For example, is the evaluation of the suitability of POSIX a more important task than an assessment of the commercial stability of the different RTOS product vendors? Further work should also take account of work already in progress, such as the ASAAC project. The further recommendations below are based on preliminary judgements of these issues.
- 2) Further work should focus on the use of standards, facilitating the use of COTS software and the reuse of software between military systems. Of the standards considered in this report, POSIX and the Ada95 are immediately applicable. An architecture framework such as GOA provides the best potential for the systematic application of software standards and products, including RTOS products, to avionics applications.
- 3) The realtime features of POSIX should be evaluated for applicability to avionics applications. The portability of POSIX applications in a realtime environment, the specification of the performance of POSIX compliant operating systems and the assessment of POSIX compliance should also be considered.
- 4) The realtime features of Ada 95 should be evaluated for applicability to avionics applications. The extent to which Annex D of the Ada Reference Manual is being supported by compiler vendors also needs to be investigated, together with the compatibility of different Ada implementations and RTOS products.

- 5) The GOA architectural framework should be exploited in two ways. Firstly, the functionality of RTOS products should be classified using the GOA model and evaluated against the requirements placed on each interface class. Secondly, the GOA model should be expanded to introduce more abstraction layers within the application software. The GOA framework, which is very general, should be compared to more specific frameworks such as the US Army Vetrionics System Standard or NASA's SGOAA.
- 6) Further evaluation should focus on the non-functional criteria: especially safety and long term support. Of the functional criteria performance, Ada support and health monitoring should be investigated. It is not recommended that a detailed evaluation and comparison of the functionality of RTOS products such as VxWorks and LynxOS would be of value.
- 7) Further evaluation of the US Army's RTEMS RTOS would be valuable since it offers a significantly different approach to that of using a commercial RTOS product, especially with respect to the non-functional evaluation criteria such as supportability.
- 8) New software standards for object communication such as the Common Object Request Broker Architecture (CORBA) are being created. Although these standards are not supported by existing RTOS products, their use is being considered within avionics systems. The evaluation of standards such as CORBA should be considered in future.

6 REFERENCES

Real Time Operating System. Report prepared for the ASSC Processing Working Group by ERA Technology, April 1996. ASSC/330/2/136.

ARINC Project Paper 653 'Avionics Application Software Standard Interface'. Aeronautical Radio Inc, 2551 Riva Road, Annapolis, Maryland USA.

ISO/IEC 9945-1 ANSI/IEEE Std 1003.1 Information technology - Portable Operating System Interface (POSIX). Part 1: System Application Program Interface (API) [C Language]. Second edition 1996-07-12.

SAE International AS4893 Generic Open Architecture (GOA) Framework, SAE AS5 GOA Task Group, Working Draft, July 11th 1995. Distributed as ASSC/510/3/31 Iss 2.

Ada Reference Manual: Language and Standard Libraries. Version 6.0 21 December 1994. ISO/IEC 8652:1995(E). Online version at <http://www.adahome.com/rm95/>.

ARINC Report 651 'Design Guidance for Integrated Modular Avionics'. Aeronautical Radio Inc, 2551 Riva Road, Annapolis, Maryland USA.

ARINC Report 652 'Guidance on Avionics Software Management'. Aeronautical Radio Inc, 2551 Riva Road, Annapolis, Maryland USA.

Edwards R.A., "ASAAC Phase I Harmonised Concept Summary", 1994 Avionics Conference and Exhibition, Heathrow, UK, November 30th to December 1st 1994, ERA report 94-0973, paper 10.3 (also distributed as ASSC/020/3/56).

ASSC/020/3/57 ASAAC Phase I Concept Summary: Slides.

Field D. and Grigg A., "The Impact of Interchangeability Requirements on Operating Systems for Modular Avionics", 1994 Avionics Conference and Exhibition, Heathrow, UK, November 30th to December 1st 1994, ERA report 94-0973, paper 9.3.

Wake A.S., Miller P.R., Moxon P. and Fletcher M.A., "Modular Avionics Operating System - Software Concept", 1996 Avionics Conference and Exhibition, Heathrow, UK, November 20th to 21st 1996, ERA report 96-1051, paper 10.5.

ASSC/330/2/130 Integrated Modular Avionics, Software Architecture Concept, Alan Grigg, BAe Defence Ltd, Military Aircraft Division.

Software User Manual for the Avionics Operating System of the F-22 Common Integrated Processor. 5139492, Standards Development Version, 22 August 1995, Hughes Aircraft Company. Distributed as ASSC/330/3/342.

Open Systems and the Common Integrated Processor (CIP) M. Rodriguez and R. E. Baeuchler, Hughes Aircraft. 1995 Avionics Conference and Exhibition: Low-cost avionics - can we afford it? Heathrow UK, 29-30 November 1995.

Shore D.J. and Barton G.P., "Avionic Modular Component Acquisition and Evaluation", 1995 Avionics Conference, page 10.8.1.

Military Handbook: Vetronics System Architecture, prepared for US Army Tank-Automotive R&D Center, Vetronics Technology Center by TARDEC Software Engineering Division, DCS Corp. Sept. 27 1994. Distributed as ASSC/510/3/30. Available at <http://www.tacom.army.mil/vsa/vsa.html>.

Vetronics Real Time Operating System (VRTOS): API Specification. Appendix A of (Ref. 18). Distributed as ASSC/330/2/248. Available at <http://www.tacom.army.mil/vsa/vsa.html>.

Intermetrics, Inc. Ada 95 Rationale: The Language, The Standard Libraries. January 1995. Available from <http://www.adahome.com/Resources/References.html>.

Avionics Operating System: Application Program Interface Requirements. SAE ARD 50067, Draft, May 11th 1996. Distributed as ASSC/330/3/314 Iss 2.

Open System Interconnection (OSI) and other Reference Models - User Perspective for Real-time Applications. SAE AIR 4885, Draft 7, 14th May 1996. Distributed as ASSC/510/3/60.

New Attack Submarine (NSSN) Software Acquisition and Reuse Strategy, prepared by Comptek Federal Systems, for NUWC DIVNPT Code 2253, January 2 1996. Available from <http://sw-eng.falls-church.va.us/reuseic/methods/Welcome.html>.

Software Reuse Development Plan for New Attack Submarine, FY95 Report, January 2, 1996, prepared by American Systems Corporation, for NUWC Code 2253 Available from <http://sw-eng.falls-church.va.us/reuseic/methods/Welcome.html>.

ISO/IEC 13719 Information technology -- Portable Common Tool Environment. Part 1: Abstract Specification (also ECMA-149), Part 2: C programming language binding (also ECMA-158), Part 3: Ada programming language binding (also ECMA-162), 1995.

John A. McDermid. COTS: The Expensive Solution? IEE Colloquium: COTS and Safety Critical Systems. January 28th 1997, IEE Digest No: 97/013.

RTEMS Home Page. <http://lancelot.gcs.redstone.army.mil/rtems.html>.

Prepared by:

**D W R Marsh
Software Systems Department
ERA Technology Ltd**