

## Flexibility and Manageability of IMS Projects

Zoë Stephenson, Mark Nicholson, John McDermid; University of York Department of Computer Science;  
Heslington, York, YO10 5DD, UK

Keywords: Safety-related, evolution, standards

### Abstract

In the past few years, Integrated Modular Systems (IMS) have been increasingly seen as a way of reducing the cost associated with computing in high-integrity control applications. An IMS is a network of computational nodes, sensors (with redundancy) and actuators. This type of system uses specific software architectures and configuration processes to allow the deployment of the control application in different arrangements. Eventually, it will be commonplace for IMS software to reconfigure during operation - an eventuality for which it would be prudent to prepare.

Previous work (ref. 1) compared IMS concepts with the idea of staged product lines (ref. 2). The product line model shows particular issues for technology support, processes and change management; our analysis indicates ways in which IMS software and configuration processes could be modified so that they are able to support the full range of possible IMS characteristics from fixed-configuration systems based on hardware-specific integration to fully-reconfigurable logical partition-based systems.

To continue this strand of work, we assess representative processes, architectures and standards for IMS development against these particular recommendations. The results are able to show any limitations that are imposed on these projects with respect to the flexibility of the IMS support infrastructure. From these results, further specific recommendations can be made regarding the future of IMS development and accompanying processes and standards.

### Introduction

In the safety-critical domain, Integrated Modular Systems (IMS) technology is increasingly being used as an implementation platform. An IMS is a networked computer system providing embedded control and monitoring functions within a platform such as a car or an aircraft (ref. 3). The use of such systems in vehicles means that the control and monitoring functions are potentially safety-critical, so system health monitoring and reconfiguration are primary concerns in the IMS infrastructure. The networked computing environment provides many different opportunities to reconfigure in response to faults and errors arising in the system. Reconfiguration is typically handled by a program manager and a database of available configurations within each execution node in the network; the configurations are generally calculated offline and their identification may involve a number of search techniques (refs. 4–5). Reconfiguration requires the trigger to be identified (such as a failure), the appropriate next configuration to be identified and a safe transition process to be enacted.

When developing a safety-critical system, one of the most challenging trade-offs is between flexibility, safety and cost. An increase in flexibility can reduce design cost, but carries with it a risk of increased cost in assessing for safety. For example, a change to a component that adds a new dependency on information from another component can radically alter the propagation of failure modes and the presence of common cause failures. A technology such as IMS leverages reuse to improve flexibility and reduce cost while maintaining levels of safety. A previous paper (ref. 1) identified a way of analysing the IMS configuration process to identify flexibility requirements so that particular types of change that were deemed likely in the evolution of IMS are already catered for in the supporting configuration and application infrastructure. This reduces the risk that changes to the configuration technology will be needed, and hence reduces the overall cost associated with reusing the configuration technology on multiple projects. Reuse of configuration technology also has associated benefits in reducing the cost of training and increasing the opportunities to transfer skills between projects. The recommendations are reproduced here for convenience:

- The same configuration process and language should be used, regardless of the functional integration approach.
- The initialization interface for all software components should support initialization within any planned partitioning scheme.
- The system should always carry a configuration graph component, with an interface to the program manager that it is not sensitive to the number of available configurations; it should also not be overly inefficient to use the interface when there is only one configuration.
- A mode-sensing component should be included to map between operational modes and failures and the types of configuration calculation and authorization that may be used. This encapsulates knowledge about the relationships between particular modes and failures and the reconfiguration process.
- The description of a configuration must include partition failures and restarts, and the configuration calculation process must have an interface to access the partition restart history of the system.
- The interface to the configuration graph structure should cater for different (preconfigured vs. online calculation) approaches to configuration calculation; in particular, it should be possible to specify a time-limit for any calculation.

In this paper, the focus is on the comparison of these recommendations with established standards and projects. In the following section we briefly introduce the scope and applicability of the various standards that affect the development of an IMS project. The subsequent sections compare the standards with the recommendations given above, describe some aspects of an actual IMS implementation with respect to the recommendations, and then summarise with ideas for further work. Throughout this paper, the focus will rest on avionics standards, although the recommendations listed above should also apply to other domains.

### Standards

ARINC 653: The ARINC 653 standard (ref. 6) is the Avionics Application Software Standard Interface. It consists of three parts, detailing Required Services, Extended Services and Conformity Test Specification; the work in this paper considers the Required Services part, ARINC 653-1. The standard defines a uniform “Application Executive” (APEX) interface between the application and operating system of an avionics computer system. It defines both the calling mechanism and the services to be provided through that mechanism.

The part of ARINC 653 most often referred to in IMS literature is the specification of time and space partitioning measures. The application is divided into partitions that are allocated to different memory blocks and scheduling time slots. The APEX interfaces provide services to the applications such as inter-application communication, error handling and hardware device communication.

ARINC 653 is designed from a safety viewpoint; hence its structure is generally one of non-interference and analysability. It is generally applicable to modular aerospace control software, but there is a planned progression from highly critical applications using only essential services to less critical, more general-purpose applications using a variety of operating system services.

ASAAC: The Allied Standard Avionics Architecture Council is a consortium of aerospace organisations from the UK, France and Germany. The ASAAC standard (ref. 7) is published as UK Ministry of Defence Standards and NATO standards; it is divided into software, communications, common functional modes, packaging and architecture. Each document describes interface standards to help alleviate problems of complexity and obsolescence, ranging from physical concerns such as the rack size and electrical connections through open network standards all the way to software programming interfaces.

The ASAAC software API considers many features that are not described in ARINC 653-1, such as file handling, threads and debugging. Its interfaces may be translated into calls to ARINC 653 services where appropriate. The standard applies across a range of avionics equipment and applications.

DO-297: Guidance on the certification issues of integrated modular systems in avionics applications is available from RTCA/EUROCAE as DO-297, “Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations” (ref. 8). This guidance has been drawn up to inform those involved in the design, implementation, approval and continued airworthiness of integrated modular systems in civil aerospace certification projects. The guidance clarifies how the use of IMS technologies alters the way in which civil aerospace products are planned, developed and assured.

### Comparison of Standards

Language: “The same configuration process and language should be used, regardless of the functional integration approach.”

Motivation: The integration approach is concerned with the dependencies between the application and the various hardware devices that are used. The diagram in Fig. 1 shows two possible arrangements. Fig. 1(a) is a structure in which application components are largely associated with specific hardware items, and Fig. 1(b) is a more functional scheme where functions communicate with hardware as necessary. The diagram indicates through shape and shading the actual provision of end-user functionality. For example, an engine controller’s thrust limitation functionality would traditionally be spread across several fault-accommodation components and the thrust reverser control components; in a functional integration scheme, that responsibility could be encapsulated in a single application. The configuration language should remain independent of the integration approach used.

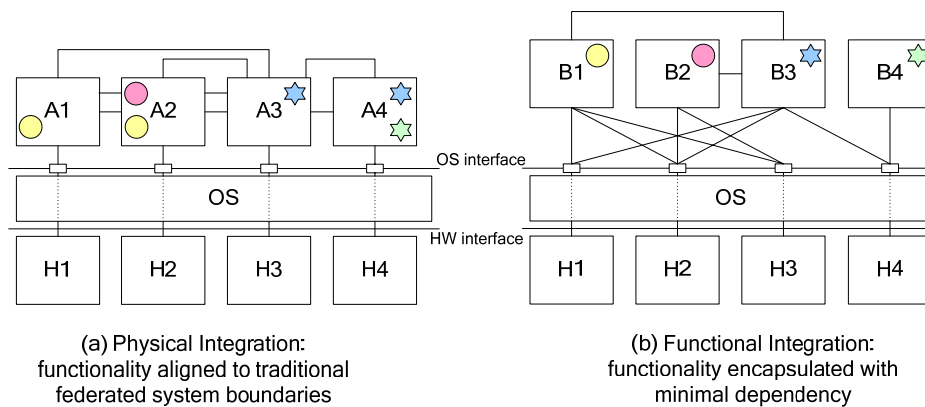


Figure 1 – Physical versus Functional Integration  
Coloured symbols mark provided functionality

ARINC 653: The standard identifies a System Integrator role, with responsibility for ensuring that configuration tables for message routing, memory and time slicing are properly defined. However, the standard does not specify the implementation of the configuration process. From the perspective of the APEX interfaces, the difference between physical and functional integration is simply a difference in the communication between the physical devices and the applications – it is expected that there will be fewer such connections for physical integration, and more connections for a more logical integration scheme. This effect can be seen at the “OS interface” boundary, where there are more connections in Fig. 1(b) than in Fig. 1(a).

ASAAC: ASAAC considers configurations in terms of blueprints describing the allocation of threads, processes, schedules and virtual channels (ref. 10). There is no specific configuration language specified for blueprints within the standard. The treatment of processes within the standard will not be affected by a change from physical to functional integration.

DO-297: The standard is concerned with the qualification of tools that generate configuration data and the processes by which the configuration is loaded into the system and shown to be a correct configuration for which certification has been obtained. From the viewpoint of the configuration language, the tools that process such languages should be qualified to work on any integration approach so that changes to the approach do not require requalification of the

tools – although the upcoming DO-178C update to the DO-178B standard (ref. 9) does include new considerations for the validation of tool-generated suggestions that would also mitigate against this phenomenon. The process of demonstrating correctness of the configuration is expected to be unaffected by the type of integration approach used.

**Best Practice:** An organisation deploying IMS should ensure that the current and future integration approaches have been investigated and that any configuration technology designed for long-term use is able to handle any of those planned integration approaches. Most existing configuration tools operate on generic component concepts without specific classification according to hardware dependency, so this recommendation is expected to be met in a majority of IMS projects.

**Initialisation:** *“The initialization interface for all software components should support initialization within any planned partitioning scheme.”*

**Motivation:** There is a small possibility that the initialisation interface of the component and the context within which it is initialised – items such as process basepages, parameters, environment definitions or stack structures – may vary depending on the way in which partitions are created. If a physical partitioning scheme is used, the process could be initiated directly from the operating system of the partition; with a logical partitioning scheme, other software would have been invoked to set up the partitions, so it may use a different loader and invocation to start processes. This recommendation is in place to check that there is no such dependency in the deployed system.

**ARINC 653:** Partitions are started using a single main program; the program is responsible for announcing further processes within that partition. This reduces dependency on an OS-specific process description data structure. The standard discusses various mechanisms for storing and configuring the process attribute data before its use in announcing processes. This should result in a situation where the partitioning scheme influences configuration data but not the main program itself. The actual interface through which the main program is initialised is not specified, and is assumed to be specific to the linker and libraries for a particular platform.

**ASAAC:** The standard does not supply a specific interface for the initialisation of software by operating system services, except for a general-purpose ‘start thread’ call as part of the management of multi-threaded processes. It is assumed that process initialisation will be specific to the tool chain used to deploy the software.

**DO-297:** There are no certification concerns in DO-297 that are specific to the context of the component initialisation interface. There are concerns that deal with the correct initiation and termination of processes and the ability of processes to influence one another’s execution, especially when considering cross-partition processes that are able to initialise partition boundaries. None of these issues define the linking mechanism by which the process is loaded and started, however.

**Best Practice:** IMS standards do not currently address this issue. The organisation should ensure that a standard initialisation context and interface is defined for the components deployed on their IMS systems, and either the organisation or its suppliers should build the necessary adapters to isolate the software processes from dependencies on the initialisation interface.

**Graph Component:** *“The system should always carry a configuration graph component, with an interface to the program manager that it is not sensitive to the number of available configurations; it should also not be overly inefficient to use the interface when there is only one configuration.”*

**Motivation:** There is a general trend towards more adaptable and dynamic software, and eventually this will be reflected in the configuration of IMS, with online configuration calculations using advanced optimisation and intelligence techniques being employed. To ensure that the configuration infrastructure is able to cope with this eventuality, the configuration specification should be encapsulated within an active component that is accessed through an interface that does not impose undue overhead in situations where there are relatively few configurations.

**ARINC 653:** The standard uses the concept of a configuration table, a static data object accessible by the operating system but not built into the operating system, and hidden from applications. It specifies operating system configuration for a module and its partitions such as the ports, channels and processes that are in use. The combination of modules to make a system also requires a system-level configuration process, and that is outside of

the scope of the standard. From this perspective, ARINC 653 provides a separate configuration graph component with constant-time access to an arbitrary number of configurations, but only as a passive memory-based lookup.

ASAAC: The configuration graph is accessed through the active SMBP (System Management to Blueprints) interface, abstracting away from the storage of the data. The standard does not specify any constraints on the computational complexity of the access through this interface.

DO-297: The DO-297 standard identifies the need to demonstrate that the configuration installed on the aircraft is that for which the certification was issued, which implies an identifiable component that records that configuration. Once dynamic reconfiguration options are considered, this may invoke the parts of DO-297 that deal with the qualification of reusable components.

Best Practice: The organisation adopting IMS should be aware of the computational complexity associated with accesses to configuration data, and take measures to ensure that the complexity is bounded – ideally within a linear-time bound. For ARINC 653-1, the organisation should also invest in a separate software component to encapsulate the access to the configuration data, isolating changes to the format of the data and the methods by which a configuration is computed from the data.

Mode Sense: *“A mode-sensing component should be included to map between operational modes and failures and the types of configuration calculation and authorization that may be used. This encapsulates knowledge about the relationships between particular modes and failures and the reconfiguration process.”*

Motivation: Consider the situation where an inefficiency is detected in a number of sensors, and a new configuration is sought that makes the best possible use of the combination of efficient and inefficient sensors to maintain the best possible fuel efficiency and the lowest possible emissions. This is an optimisation calculation that could take some time, but the move to this new configuration is not critical – the system can function for quite some time before moving into this new, more efficient configuration. Now consider the situation where a fire is detected in an engine. A new configuration is sought in which fuel is no longer supplied to the engine, and this is a calculation that is required immediately. To manage an advanced IMS product in which a range of events can prompt a reconfiguration, there must be some knowledge of the relationship between reconfiguration-triggering events and the allowed methods of calculation. This is encapsulated in a mode-sensing component.

ARINC 653: ARINC 653 identifies in its system architecture the presence of health monitoring and built-in test equipment. It also calls for a system health management table under the control of the system integrator. Provision is made for process, partition and module-level errors, and the standard identifies the need to have specific recovery actions such as starting and stopping processes or restarting partitions. No definitive list of responses to failures is given; the XML schema for configuration data refers to an `Error_ID_Action` containing an Action of type `ModuleActionType`, but this type is not defined further in the standard. It would be possible to use a separate data item to define the coupling between events and reconfigurations, and to make this available to health management callbacks in the various partitions, effectively encapsulating the control logic in a small number of components. The standard does not mandate any such system, however.

ASAAC: ASAAC provides a set of calls that manage dialogue between the application-level configuration management (AM) and the configuration management for the module (GSM). The configurations are represented with a set of static identifiers. There are two options for the placement of mode sensing – either within the GSM to process configuration identifiers and optimise the configuration dynamically or within the blueprint database access mechanisms themselves. Neither mechanism is constrained by the standard.

DO-297: There are no specific constraints in the standard that affect the use of mode-specific configuration calculations, except that the standard is written with field-loading of configuration data during maintenance as the most dynamic and flexible means of reconfiguring the system. Any component that takes on the responsibility for dictating the type of reconfiguration calculation to make also takes on a measure of authority for reconfiguration, which should be considered when addressing the safety of the system. Some initial ideas in this area have already been considered (ref. 10).

Best Practice: There should be provision within the software architecture of an IMS for the inclusion of a mode-sensing component. The organisation should identify where such a component would be located and analyse its

dependencies. The relationship between the authority held by such a component and the overall authority for reconfiguration should be made explicit and taken into account during safety analysis.

Configuration History: *“The description of a configuration must include partition failures and restarts, and the configuration calculation process must have an interface to access the partition restart history of the system.”*

Motivation: There is a possibility with reconfiguration of either getting stuck or oscillating between configurations. For example, there may be one configuration in which control is severely degraded, making use of only a few sensors. A reconfiguration to a more sophisticated control mode makes use of a larger set of sensors, including one which is faulty. The faulty sensor then indicates a failure and the system is reconfigured back into the degraded state, where it now loses information about the faulty sensor. This cycle could continue, compromising the actual purpose of the system by thrashing between configurations. Hence, some knowledge of previous configurations, in particular failures and restarts, would be useful in determining a useful configuration.

ARINC 653: There are no specific provisions in the XML schema for configuration to allow for configuration history, but the standard does allow for the schema to be extended to include new capabilities.

ASAAC: The descriptions given in the grammar for blueprint data make no provision for configuration history.

DO-297: There are no specific considerations that address the precise interface by which configurations are described. If there is a dependency between configuration calculation and the configuration history, this should be documented in advance and the dependency should be included in the overall analysis of failures to show that there is no adverse effect from its presence.

Best Practice: An organisation intending to use IMS for failure-handling reconfiguration must ensure that the problems outlined in the motivation section have been addressed. There must be analysis to show whether the configuration scheme may suffer from this cyclic phenomenon; if so, the agents responsible for reconfiguration must be able to intervene in cases where configurations repeat themselves. This could be an additional authority within the reconfiguration system, or operator authority for those reconfigurations that could suffer from the repeating phenomenon.

Graph Interface: *“The interface to the graph structure should cater for different (preconfigured vs. online calculation) approaches to configuration calculation; in particular, it should be possible to specify a time-limit for any calculation.”*

Motivation: Some types of configuration calculation may take time to arrive at a satisfactory outcome. Other types of configuration are quicker, but not necessarily optimised to meet a range of competing criteria. To allow for the best possible configuration within a particular time period, it must be possible to specify that time period when requesting the calculation. For example, the configuration calculation component could use an approximation or interpolation to arrive at a reasonable configuration immediately, and then spend the rest of the available time searching for an improved configuration.

ARINC 653: ARINC 653 does not specify any interface mechanism other than access to a static configuration table. However, the XML schema for configuration is extensible and could be adapted to identify alternative mechanisms. In this case, partition restarts and timeouts could be specified in the schema, but there would need to be a corresponding provision within the OS itself. Any changes to the schema would also require significant research to demonstrate their correctness and usefulness.

ASAAC: The communication between application, health monitoring and configuration management is limited to single static identifiers to label configurations. The ASAAC standard does not impose any further meaning on these identifiers; some additional interpretation would have to be added to use certain bits in the identifier to encode configuration calculation information.

DO-297: The guidance calls attention to the specification and demonstration of timing issues in modules. The use of time-limited calculations would naturally flow down from the overall timing concerns for the platform. There are no limitations placed by this standard on the interfacing used to achieve this, however.

Best Practice: The organisation using IMS should investigate the interfacing requirements for configuration calculation and include the various options within the interfaces used in IMS products. This may involve the specification of a “wider” interface that is used internally, and mapped onto the narrower interface for configuration look-up. Future changes to the data stored in configurations and the way configurations are calculated would be contained behind this interface.

Summary of Findings: The standards generally give no specific guidance for configuration calculation mechanisms. There is a general assumption within the standards that the configuration will be derived from data tables loaded into the system during installation or on-ground preparation. DO-297 specifies comprehensively those parts of the existing standards that relate to each part of a particular model of IMS; this model also includes no dynamic reconfiguration, so the recommendations under consideration in this paper are mostly outside of its scope. None of the recommendations were found to directly conflict with any of the guidance given in the standards.

### Recommendations in Practice

We were fortunate to be able to study an example of an aerospace project at BAE Systems using IMS as its deployment platform. Commercial considerations prevent the disclosure of many of the details of the IMS implementation, but the general discussion that follows is indicative of the issues being faced in contemporary systems development.

Language: *“The same configuration process and language should be used, regardless of the functional integration approach.”*

The general approach to application specification is to consider a decomposition based on functional concerns rather than necessarily binding particular software and hardware elements together. The data formats used for the representation of the hardware and software elements are general-purpose enough to cope with either type of integration.

Initialisation: *“The initialization interface for all software components should support initialization within any planned partitioning scheme.”*

The general principles of separation of concerns and information hiding combine to ensure that only the operating system needs to know how the system is partitioned. There are no specific details of partition location or sharing passed on to the applications; they simply communicate over the various channels that are provided, and assume that the system has been correctly configured by the system designer. While there are issues here to do with the overall system health monitoring and reconfiguration, they are outside of the scope of the recommendations.

Graph Component: *“The system should always carry a configuration graph component, with an interface to the program manager that it is not sensitive to the number of available configurations; it should also not be overly inefficient to use the interface when there is only one configuration.”*

The ASAAC approach is used on this particular project, with configuration data accessed over a dedicated programming interface. The particular implementation used does not impose a performance penalty for large configuration graphs.

Mode Sense: *“A mode-sensing component should be included to map between operational modes and failures and the types of configuration calculation and authorization that may be used. This encapsulates knowledge about the relationships between particular modes and failures and the reconfiguration process.”*

The existing interface is not capable of requesting a particular type of information, so the project does not currently have extensibility that supports this recommendation. However, the need for such a facility had already prompted research work in the area of configuration calculation methods before the study reported in this paper was conducted.

Configuration History: *“The description of a configuration must include partition failures and restarts, and the configuration calculation process must have an interface to access the partition restart history of the system.”*

There are currently no plans to include configuration history directly into the local configuration process, but the project does use a hierarchical structure of system manager components. This would allow the introduction of configuration history into the system-level configuration management components without necessarily affecting the behaviour of the local components. This level of coordination could ensure that the configuration does not “stick” on a single configuration that causes restarts, nor oscillate between configurations that fulfil competing criteria.

Graph Interface: *“The interface to the graph structure should cater for different (preconfigured vs. online calculation) approaches to configuration calculation; in particular, it should be possible to specify a time-limit for any calculation.”*

The philosophy on the current project is one of static configuration, so the issues of time limits and alternative configuration schemes have not been directly addressed. The underlying technology uses a general-purpose state machine representation, and this is the area that would need to be adapted to accommodate alternative configuration techniques.

### Conclusions and Further Work

This paper has taken a set of recommendations for flexibility in IMS configuration and compared those recommendations with the constraints imposed by relevant standards in the avionics domain. The findings show no major issues that prevent the implementation of the recommendations, although some aspects of such an exercise would not be trivial. The following challenges can be identified from the comparisons discussed above:

Configuration History: The configuration process should have a way to take the configuration history into account. This may be a direct, local coupling between the component responsible for deciding on the configuration and a stored list of previous configurations; it may also be the responsibility of a remote configuration component. The arrangement should be chosen based on safety analysis of the configuration architecture. Of particular concern is the possibility that a local component with responsibility for reconfiguration could produce an unnecessary reconfiguration (in SHARD terms (ref. 11), a commission failure).

Configuration Calculation Method: The eventual outcome of IMS deployment could be a full-calculation configuration manager, using a combination of intelligence and optimisation techniques to adapt the configuration to the environment and system health. If an organisation really embraces the possibilities of IMS, it would be advantageous to consider support for such techniques in the architectures and configuration systems in current use. Deciding on the right trade-off between dealing with present concerns and incurring cost by considering future possibilities is an interesting research challenge.

Precedent: Each advance in configuration capability represents a precedent to be set by the organisation applying for certification. As with any change in the safety-critical area, this represents an additional cost, especially for the recommendations given in this paper that lie outside of the existing scope of the standards. Analysing the costs and benefits of these recommendations will be crucial to their acceptance and deployment in an industrial setting.

In addition to these challenges, the six recommendations were originally produced through an analysis process based on a staged product line model. It would be appropriate to consider ways in which organisations can develop product lines of IMS applications. The recommendations provide flexibility and some degree of insulation from future technology changes in IMS configuration, but changes in required system functionality will still need to be managed.

One possible view is to consider product lines as a way of providing a constrained context within which to reuse application components. The product line scope identifies the exact context within which components will be reused, enabling the rapid construction of a reuse library and maximising return on investment. When developing reusable software components for avionics applications, there is a need to declare the context of reuse in advance as part of the overall development plan. Product line analysis would seem to provide a perfect way of identifying this context of reuse, and enabling the development of reusable software components to improve flexibility while minimising the risk of incurring increased cost. The product line analysis also provides a feature model to show the

different configurations in which component functionality will be inactive, helping to plan the process of demonstrating during integration testing that such functionality is not inadvertently enabled.

#### Acknowledgements

The work presented in this paper was conducted as part of the ongoing research of the Rolls-Royce UTC in Systems and Software Engineering at the University of York, with technical details provided by representatives of BAE Systems. We are grateful to these industrial partners for their support.

#### References

1. Stephenson, Z.; Nicholson, M.; McDermid, J. A. Product Line Recommendations for Integrated Modular Systems Proceedings of the 23<sup>rd</sup> International System Safety Conference, San Diego, September 2005
2. Czarnecki, K.; Helsen, S.; Eisenecker, U. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. Software Process Improvement and Practice volume 10(2). 2005.
3. Wlad, J. A New Generation in Aircraft Avionics Design. ECE Magazine, May 2005, pp14–17.
4. Nicholson, M. Selecting a Topology for Safety-Critical Real-Time Control Systems. Department of Computer Science University of York U.K., 1998.
5. Strunk, E.A.; Knight, J. C. Distributed Reconfigurable Avionics Architectures. 23rd Digital Avionics Systems Conference, Salt Lake City, UT, USA. 2004
6. ARINC. ARINC 653-1 Avionics Application Software Standard Interface. October 2003.
7. ASAAC. ASAAC Phase II Stage 2, Second draft of proposed guidelines for system issues – Volume 2: Fault Management. REF-WP: 32350, 2002
8. RTCA/EUROCAE. Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations DO-297, August 2005.
9. RTCA/EUROCAE. Software Considerations in Airborne Systems and Equipment Certification DO-178B/ED-12B, December 1992.
10. Jolliffe, G. Exploring the Possibilities Towards a Preliminary Safety Case for IMA Blueprints Proceedings of the 23<sup>rd</sup> International System Safety Conference, San Diego, September 2005.
11. Fenelon, P.; Nicholson, M.; McDermid, J. A., Pumfrey, D.. Towards Integrated Safety Analysis and Design SIGAPP Applied Computing Review, 2(1), 1994, pp21–32.

#### Biography

Zoë Stephenson, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432749, facsimile – +44 1904 432708, email – zoe.stephenson@cs.york.ac.uk

Zoë Stephenson graduated from the University of York with a first-class honours degree in Computer Science. She has previously worked as a research student funded through an EPSRC CASE studentship with Rolls-Royce plc., on the Converse project, part of the EPSRC Systems Engineering for Business Process Change managed research programme. Work has been targeted at the development of an embedded systems software engineering process that is capable of operating over a complete product line. She is now working in the Rolls-Royce UTC in Systems and Software Engineering on flexible system and software modelling technology and processes.

Mark Nicholson, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432789, facsimile – +44 1904 432708, email – mark.nicholson@cs.york.ac.uk

Mark Nicholson is a Research and Teaching Fellow in System Safety Engineering in the Department of Computer Science at the University of York. He co-ordinates the Masters and postgraduate certificate programs in System Safety and Safety-Critical System Engineering in the Department. He is a member of EUROCAE WG63 updating the aerospace recommended practices 4754 / 4761. He has been researching safety-critical systems for more than 10 years. His doctoral research focused upon issues surrounding the selection of architectural structures with appropriate reliability, timing and safety characteristics.

John McDermid, High-Integrity Systems Engineering Group, Department of Computer Science University of York Heslington, York YO10 5DD, United Kingdom, telephone – +44 1904 432726, facsimile – +44 1904 432708, email – john.mcdermid@cs.york.ac.uk

John McDermid has been Professor of Software Engineering at the University of York since 1987 where he runs the high integrity systems engineering (HISE) research group. HISE studies a broad range of issues in systems, software and safety engineering, and works closely with the UK aerospace industry. Professor McDermid is the Director of the Rolls-Royce funded University Technology Centre (UTC) in Systems and Software Engineering and the BAE SYSTEMS-funded Dependable Computing System Centre (DCSC). He is author or editor of 6 books, and has published about 280 papers.