Product-Line Technology Recommendations for Integrated Modular Systems

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

Keywords: Product line, safety-related, evolution, staged configuration

Abstract

Product lines (groups of similar products) have been well-received in organizations that develop products that contain embedded software, yielding improvements to change management and reuse. These techniques, however, only begin to address the wider issues of integrated systems engineering, and certainly do not yet address highly complex configuration and reconfiguration scenarios such as those found in Integrated Modular Systems (IMS).

We believe that there is benefit to be gained by applying the successful disciplines of software product-line engineering to IMS evolution. An IMS is a networked computer systems providing (potentially safety-critical) embedded control and monitoring functions, and as such may eventually use highly complex dynamic reconfiguration techniques. These techniques will evolve and mature over time, and it would be useful to put a support environment into place to deal with configuration throughout the evolution of the technology.

The impact of the different possible configuration and reconfiguration schemes can be assessed by instantiating them as staged product-line configuration processes. Our assessment highlights areas that will be affected as reconfiguration decisions (bindings) are deferred to run-time, resulting in a number of technology recommendations for the implementation of IMS s – in particular, for a simple fire and smoke detection system.

Introduction

Embedded software systems are nearly always good examples of product lines – groups of products that share enough similarity that it is useful to consider them as variants on a single common core (refs. 1–4). They tend to share technology, functionality and consumer markets, leading to significant savings when these shared aspects are used as a basis for organization and reuse. Indeed, some of the most successful reported uses of product line technology (refs. 5–6) have been in the context of embedded systems.

So far, product line techniques have been applied successfully in relatively few domains. As product line technology continues to mature, it becomes capable of addressing more and more of the issues that are relevant to a wider range of domains. The advances in the last few years include ever-improving tool support for product line scoping, definition and configuration (refs. 7–8), concepts of staged configuration and customization (ref. 9), and more rigorous formal semantics for product derivation models (ref. 10).

Among these advances, very little effort has gone towards the use of product line technology to manage complex safety-critical and safety-related control systems. These systems have specific characteristics such as control loop behavior, safety analysis and maintenance scheduling that must be considered together with more general characteristics of functional behavior, timing, throughput, weight and size that are commonly associated with embedded devices.

In product lines, the three main phases are scoping, asset engineering, and product configuration. The scoping phase identifies the products that it must be possible to create, based on various marketing and technology factors. Asset engineering processes create the frameworks, components, generators, tests and other assets that are needed to support the scope of products. The product configuration phase responds to a particular customer or market need by assembling a consistent set of assets into an appropriate product. This paper is principally concerned with this last phase, and its application in safety-critical systems engineering.

In the safety-critical domain, Integrated Modular Systems (IMS) technology is most appropriate for product lines. An IMS is a networked computer system providing embedded control and monitoring functions within a platform such as a car or an aircraft. The use of such systems in vehicles means that the control and monitoring functions are potentially safety-critical, so 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. 11–12). 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. In this paper, the focus is on the calculation of an appropriate "next" configuration. We use an example fire and smoke detection system (ref. 13) as shown in figure 1 as a typical IMS application. Fire/smoke detector appliances are distributed throughout a passenger vehicle – in this case, an aircraft – and connected to a detection system with a standard CAN bus. The detection system also communicates over various other channels with navigation, control and maintenance systems. It may be implemented on any number of networked IMS processors, although the original example uses only a single detection system.

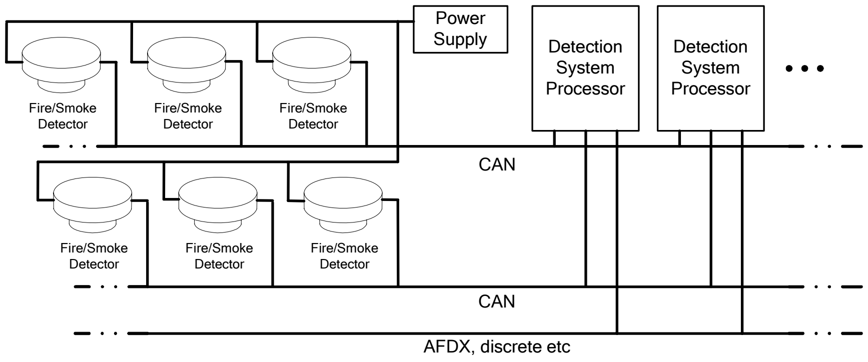

Figure 1 – Example IMS Fire/Smoke Detection System (adapted from reference 13)

Designs that use IMS technology currently have very limited reconfiguration rules as embodied by recommended practices such as Arinc-653 (ref. 14) and ASAAC (ref. 15). We expect that the maturing IMS technology will be used in more and more systems, establishing a pedigree and a knowledge base; meanwhile, automated tool support and faster processing capabilities will allow the user to search a larger number of configurations, and configurations involving increasingly diverse network elements. The necessary elements for a reconfiguration of an IMS (ref. 12) include:

- The set of applications in the system, and how they are allocated to computer systems.
- The set of viable system configurations and the set of possible transitions between configurations.
- The maximum time any part of the reconfiguration can take.
- The characteristics of the system's operating environment that affect what services are most important in a particular time and place, and how and when services can be altered.
- The possible combinations of those characteristics that need to be considered when determining potential configurations for the system.
- The relative value that each configuration will provide to the user under each potential combination of environmental characteristics.

Not all of these elements are explicitly catered for in current IMS technology; the following factors represent the changes that IMS is expected to undergo as these elements become more important:

- *Functional Integration*: are the applications mapped to the IMS structured according to the boundaries of the physical system or according to the boundaries between different functions?
- *Segregation*: are the applications partitioned using physical means, logical software support or both? What are the implications for segregation of a change in the configuration of a system?

- *Cardinality*: does the system have more than one configuration, or are all modes and failures handled within a single configuration?

- *Reconfiguration*: How does the context of operation affect reconfiguration? Is reconfiguration allowed in-flight? Is the reconfiguration process different in different modes or phases of operation?

- *Recovery*: are partitions allowed to restart? Does that depend on the context of operation?

- *Discovery*: is the database of possible configurations simply a static list, or is there some online calculation involved?

As these factors vary, there may be a wide-ranging impact on the technology and processes used in deriving configuration schemes. There could be a completely new framework for configuration and deployment at each stage of evolution, a situation that could become impractical and costly. This paper is concerned with the derivation of technology recommendations that will help to minimize those changes. This derivation is performed by modeling the IMS configuration process in terms of product line configuration.

The next section reviews related work in product line configuration. The following sections introduce the concept of a staged configuration process, relate this to a generic IMS process, and follow along with the evolution of that process using the variations outlined above. The results are discussed in terms of the case study in fire and smoke detection where applicable.

Product-Line Configuration

Although product line engineering techniques are typical of the software engineering domain, they are mostly concerned with physical products such as network switches (ref. 16), automotive subsystems (ref. 17) or mobile telephones (ref. 18). In software product line engineering, products are described in terms of the features that they exhibit rather than by the components that provide those features. This provides a useful abstraction for requirements elicitation, product marketing and user guidance. Product line features are organized into a feature model that describes their allowed combinations and parameters in any one product. A dedicated product-line engineering process ensures that the scope of variation in the feature model accurately represents the actual scope of variation in the products.

The more usual software configuration management techniques are structured for teams of software engineers collaborating on a single software project as it evolves. They manage the different instances of the project code as it changes from one version to the next, and as different engineers work on different parts of the project. However, configurations in the sense of a product line represent different products at the same stage of development. A product configuration is represented by a selection of features from the feature model; the choice among a number of feature alternatives for a product is known as a binding. In a simplistic product-line engineering environment, a generator-style configuration system will take a feature model and a desired configuration, and bind all of the configured features in one pass (ref. 2). The resultant product description may map to specific implementation artifacts, or it may operate as an input to another generator process; it may also be useful to drive some of the implementation processes directly from the configuration specification.

One particular advancement on this one-shot model of configuration is the notion of staged configurations (ref. 9). Instead of binding from a number of possible sets of product features into a single set of product features, a number of intermediate stages are used; this allows for partial configuration of the feature model for use in communicating between different organizations or for controlling particular subsets of the product line. A simple two-stage product line process is shown in figure 2 using the UML SPEM activity notation (ref. 19). Here, the intermediate stage is

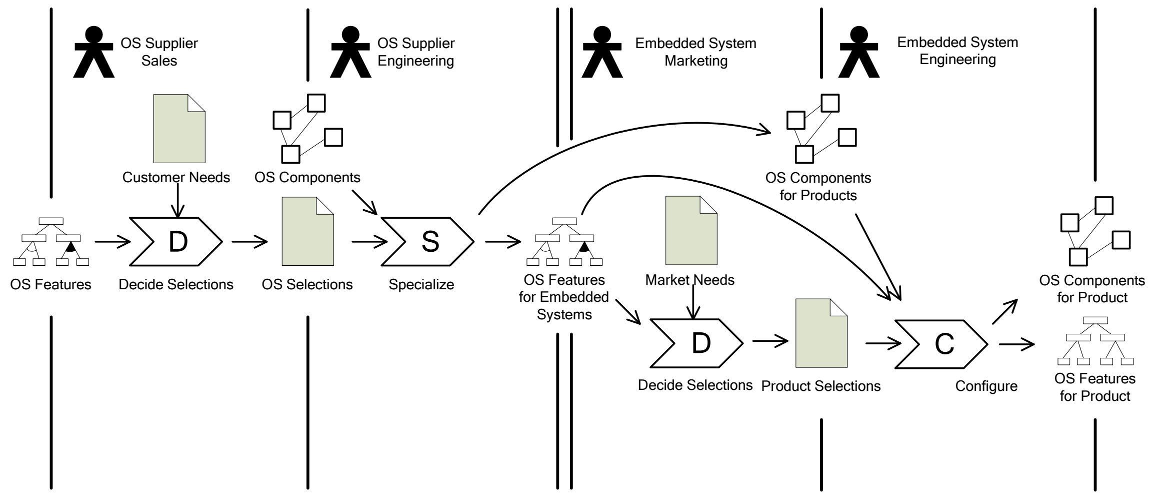

Figure 2 – Staged Product-Line Process

drawn with a double line.

The product-line processes depicted in figure 2 represent interaction between an operating system supplier and an embedded system developer. The embedded system developer supplies some description of needs to the operating System supplier; the product that is decided on and specialized is an operating system product line that contains features relevant to the family of embedded systems being developed. The embedded system developer then decides on the OS features to include with each product, configuring the specialized product line to a set of features for inclusion with the product. The benefit of this level of control over features is that the process model shows where deferred decisions are carried out, and where the responsibility for those deferred decisions lies.

<u>IMS as a Product-Line Process</u>

Our model of an individual IMS platform is relatively abstract, depicted in Figure 3. The computer system functionality is packaged as a number of individual applications. Each application uses Operating System services to access the various sensors and actuators needed to perform its function. Each application also uses the operating system services to gain access to computing resources to determine the appropriate actions given a set of sensor inputs. Processors are expected to form partitions to isolate application execution environments, either by physically allocating the applications to different processing nodes, or by imposing some form of software-based partitioning.

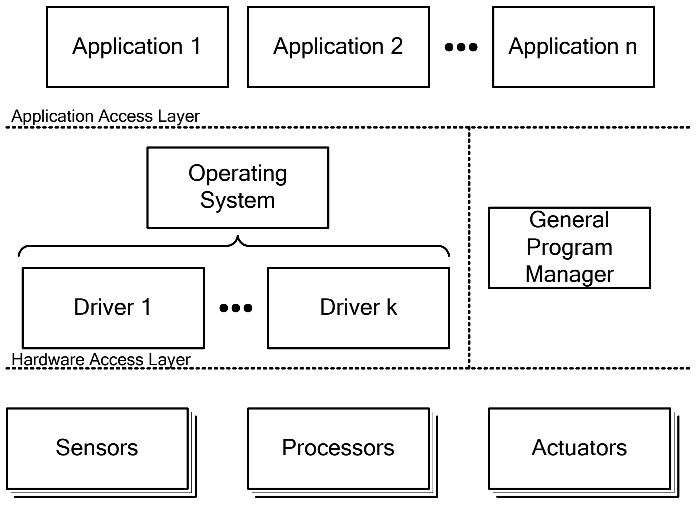

Figure 3 – Generic IMS Structure

The general program manager component is responsible for the overall health of the system. It performs local accommodation for minor problems, such as the loss of one of group of redundant sensors, and communicates with an operator to authorize configuration changes for more serious problems. In future systems it is envisaged that an increasing number of reconfigurations will be undertaken on the authority of the program manager, rather than the operator. These reconfigurations are potentially safety critical.

A configuration of an IMS, also known as a blueprint, is a particular assignment of applications to processing partitions. This assignment is used by the Operating System to provide services to the applications. For each existing configuration, there are potentially a very large number of possible next configurations only a small number of which preserve both the safety and the desired functionality of the system. This relationship between one configuration and another can be considered as a graph, with each node a possible configuration and each transition a reconfiguration triggered by some health or mode change. The graph is typically calculated in advance with a heuristic optimization algorithm (refs. 11–12). The IMS for a particular set of equipment and applications is deployed in an initial configuration with a graph data structure that encodes the possible configuration changes that it may undergo.

The staged product-line process in figure 2 can be applied to the IMS technology in figure 3 by considering the deployment of the specialized configuration to be the same as the deployment of the IMS for a particular set of equipment and applications. The result is shown in figure 4. The two lanes on the left of the diagram represent the pre-deployment activities – selecting the right equipment list and application functionality, and creating a set of blueprints for that system according to the necessary safety constraints. The remaining two lanes represent the system in operation, with the operator taking responsibility for authorizing suggested configuration changes, and the general program manager enacting the required change according to some safe reconfiguration scheme.

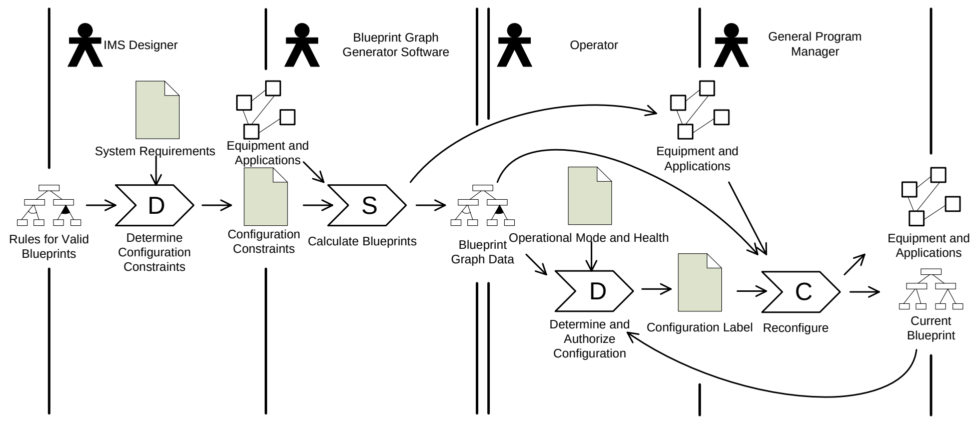
IMS Designer
Blueprint Graph Generator Software
Operator
General Program Manager
System Requirements
Equipment and Applications
Equipment and Applications
Rules for Valid Blueprints
D
Determine Configuration Constraints
Configuration Constraints
S
Calculate Blueprints
Blueprint Graph Data
Operational Mode and Health
Equipment and Applications
D
Determine and Authorize Configuration
Configuration Label
C
Reconfigure
Current Blueprint

Figure 4 – IMS As a Product-Line Process

The process-based representation used here identifies the key dependencies between the IMS configuration concept and the environment needed to support the technology in operation. For example, the product-line approach clearly indicates the different contexts in which offline configuration calculation and post-deployment configuration calculation take place. They both perform calculations based on a given set of equipment and applications, but all post-deployment configuration activities are performed by the operator (e.g. pilot) and the GPM system based on the available health and mode data.

IMS Evolution

There are a number of different ways in which the use of IMS technology can vary. It is expected that the IMS approach will generally move from less flexible configuration schemes to more flexible ones. In the introduction, six different factors were listed that are expected to influence the way IMS technology is used and supported. It would be beneficial to design IMS support environments and implementation components so that the impact of these variations is minimized - hence our interest in product lines in this paper. To derive recommendations to achieve this, each of the different factors is applied to the staged product line IMS process given in Figure 4. Recommendations that limit the impact of changes to those factors on the components of the staged product line IMS process will generally help to reduce the cost and risk associated with IMS development. The rest of this section generates such recommendations for each of the six variation factors.

Functional Integration: The integration strategy for software has a profound effect on the architecture and on the way the different parts of the functionality are allocated to applications. This equates to a different set of applications as input to the configuration process. Hence, changes to the functional integration of the system will be better isolated if the same configuration process can be used regardless of the type of integration used. This includes aspects such as the application description language.

In the fire and smoke detection system, one way in which the integration could change would be to centralize the activity that correlates sensor data and built-in test data to build an accurate model of detected events. This would allow the system to place that functionality at the nodes that are least vulnerable to damage from any detected fires. A flexible mapping between functionality and network nodes would be an important part of the application description language here.

Segregation: A logical partition segregation scheme will require support from a global initialization system, to initialize the virtual partitions and allow for partition restarts. The change to the initialization process could impact on the interface used in bootstrapping equipment and drivers used inside partitions.

To reduce this change impact, the initialization interface for all software components should support initialization within any planned partitioning scheme. For the fire detector, there is little need to segregate any of the functionality. Some built-in test maintenance management functions could be partitioned off separately, but the requirement for a consistent initialization interface would remain regardless of the type of functionality being partitioned.

Cardinality: The expectation is that IMS hardware will be used with a single configuration to demonstrate its viability, then multiple-configuration systems will be made available as the technology gains more widespread use. This will have an impact by eliminating the need to store any configuration graph for reconfiguration. However to ensure that configuration infrastructure is also viable, a configuration graph should be included from the beginning.

The interface between the program manager and the configuration graph should be constructed so 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. This advice is applicable to the fire and smoke detector example in general terms; some advanced configurations may be possible if there is redundancy in the CAN bus connections for the sensors, for example – this in addition to the ability to locate the monitoring functionality in a safe location.

Reconfiguration: There are a number of factors that are expected to influence the reconfiguration process rather than the generated configuration. There may be an opportunity to perform some of the heuristic calculations during online reconfiguration, rather than pre-calculating all of the possible configurations. However, it may only be appropriate to attempt this calculation while the aircraft is on the ground. Similarly, the system may only be trusted to a certain degree, such that reconfiguration calculation is restricted to a set of predefined known safe configurations during take-off and landing maneuvers.

This influences both the configuration graph and the process of determining and authorizing a new 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. In the fire and smoke detector system, there is no indication that any of the configuration changes would be based on operational modes; nevertheless, it would be wise to allow for the possibility in the software structure.

Recovery: Partition recovery may be required unless the application software is assessed sufficiently to show that it is unnecessary. The partition recovery process must be included as part of the reconfiguration process; that is, the program manager must be made aware of the ability to restart partitions. A configuration graph component must be able to represent failed partitions and partition-restarting configuration changes. It should also take into account the history of partition restarts, to avoid continually restarting the same partition or restarting multiple partitions simultaneously.

To account for these characteristics, 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. One particular reconfiguration scenario is especially important to consider, in which the system configures to avoid a problem, then later configures back to the original configuration and the problem recurs. If this were to happen in rapid succession, the opportunity to carry out regular functionality would be compromised.

Discovery: The configuration discovery process is expected to begin with a simple static graph structure, a known start state, and an emergency safe state. The emergency safe state will be independent of the main graph structure. As the system becomes more complex, there may only be a static graph for the first few reconfigurations, and for a number of stages before the final system configurations in the graph, with the intermediate transitions calculated online. The final stage will involve only a known start and end point, with the full graph calculation algorithm deployed to the GPM.

The interface to the graph structure should cater for each of these situations; in particular, it should be possible to specify a time-limit for any calculation. This relates back to the need to identify the maximum time for any given reconfiguration.

Conclusions and Further Work

In this paper, the concept of staged configuration in product lines has been applied to the domain of integrated modular systems. An interface between two stages was proposed, and mapped to the deployment of a dynamically re-configurable IMS. The model was used as a focus for the discussion of change impact minimization strategies for six particular types of change expected in the evolution of IMS technology. A simple example was used to help explain the discussion of the following recommendations:

- 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 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.

A particular concern, aside from technological stability, is the need to perform safety analysis that covers all of these various configurations. This is potentially the biggest impediment to the adoption of dynamic configuration strategies in safety-critical applications. The staged configuration model gives some insight into the problem; the binding time for decisions will move to later parts of the lifecycle, and so the safety case for the system as deployed must cover more configuration possibilities.

There is clearly much work to be done in this area; the following points identify the key research themes:

Incremental Analysis: Analyze the nodes and transitions of the configuration graph to identify those nodes for which a full safety analysis is needed, and those nodes that are similar enough that an incremental approach (ref. 20) is justified. Some related work in product-line safety (refs. 21–22) may help to target such an incremental technique. There may also be some value in incremental analysis of changes to the configuration graph itself.

Authority: Analyze the configuration graph to determine the authority needed for each reconfiguration, and demonstrate that the implementation upholds that authority, and that that authority is correct for the configuration of the system.

Strategy: The software is structured as a number of applications, and these applications may contain threads that negotiate over shared resources. How does the system safely negotiate reconfigurations of such applications? AADL (ref. 23) provides one way of addressing this problem, but there may be specific concerns for IMS that are not directly addressed in that work.

Dormancy: Equipment that is intended for use as a redundant fallback should not be left dormant until required; instead, the overall dormancy of the configurations across a number of missions should be minimized. The configuration calculation system must respect this type of requirement.

Repair: Equipment may be repaired, allowing the system to return to an earlier configuration. However, there are safety implications in trusting that the repair really has occurred, and in avoiding an oscillation between configurations because a device appears faulty in one configuration and working in the configuration reached because of that fault.

References

1. Parnas, D. On the Design and Development of Program Families. IEEE Transactions on Software Engineering volume 2(1), March 1976. pages 1–9.

2. Weiss, D. M.; Lai, C. T. R. Software Product-Line Engineering: A Family-Based Development Process. Addison-Wesley, 1999.

3. Griss, M. L. Implementing Product-Line Features with Component Reuse. Software Reuse: Advances in Software Reusability – Lecture Notes in Computer Science volume 1844. Springer, June 2000. pages 137–152.

4. Czarnecki, K.; Eisenecker, U. Generative Programming. Addison-Wesley, 2000.

5. Buhrdorf, R.; Churchett, D.; Krueger, C. W. Salion's Experience with a Reactive Software Product Line Approach. Fifth International Workshop on Product Family Engineering. Siena, Italy. November 2003.

6. Ardis, M.; Duley, N.; Hoffman, D.; Siy, H.; Weiss, D. Software Product Lines: A Case Study. Software Practice and Experience volume 30(7). June 2002. pages 825–847.

7. Sinnema, M.; de Graaf, O.; Bosch, J. Tool Support for COVAMOF. International Workshop on Software Variability Management for Product Derivation – Towards Tool Support. Boston, USA. August 2004.

8. Gomaa, H.; Shin, M. E. Tool Support for Software Variability Management and Product Derivation in Software Product Lines. International Workshop on Software Variability Management for Product Derivation – Towards Tool Support. Boston, USA. August 2004.

9. 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.

10. Bontemps, Y.; Heymans, P.; Schobbens, P.-Y.; Trigaux, J.-C. Semantics of FODA Feature Diagrams. International Workshop on Software Variability Management for Product Derivation – Towards Tool Support. Boston, USA. August 2004.

11. Nicholson, M. Selecting a Topology for Safety-Critical Real-Time Control Systems. Department of Computer Science University of York U.K., 1998.

12. Strunk, E.A.; Knight, J. C. Distributed Reconfigurable Avionics Architectures. 23rd Digital Avionics Systems Conference, Salt Lake City, UT, USA. 2004

13. Rosam, A. The Impact of IME-Based Design for an Equipment Supplier. VICTORIA Forum, Athens. May 2004.

14. ARINC. ARINC 653-1 Avionics Application Software Standard Interface. October 2003.

15. ASAAC. ASAAC Phase II Stage 2, Second draft of proposed guidelines for system issues – Volume 2: Fault Management. REF-WP: 32350, 2002

16. Perry, D. E. A Product Line Architecture for a Network Product. Proceedings of the Third International Workshop on Software Architecture for Product Families. March 2000. pages 41–54.

17. Deelstra, S.; Sinnema, M.; Bosch, J. Experiences in Software Product Families: Problems and Issues During Product Derivation. Proceedings of the Third International Conference on Software Product Lines – Lecture Notes in Computer Science volume 3154. Springer, September 2004. pages 165–182.

18. Maccari, A.; Tuovinen, A.-P. System Family Architectures: Current Challenges at Nokia. Proceedings of the Third International Workshop on Software Architecture for Product Families. March 2000. pages 106–115.

19. Object Management Group. Software Process Engineering Metamodel. January 2005.

20. Conmy, P.; Nicholson, M.; McDermid, J. A. Safety Assurance Contracts for Integrated Modular Avionics. 8th Australian Workshop on Safety Critical Systems and Software. Canberra, CRPIT volume 33. October 2003. pages 69–78.

21. Dehlinger, J.; Lutz, R. R. Software Fault Tree Analysis for Product Lines. IEEE International Symposium on High Assurance Systems Engineering. 2004.

22. Stephenson, Z.; de Souza, S.; McDermid, J. A.; Ward, A. G. Product Line Analysis and the System Safety Process. Proceedings of the International System Safety Conference. System Safety Society. 2004. pages 790–799.

23. Feiler, P.; Lewis, B.; Vestal, S. The SAE Avionics Architecture Description Language (AADL) Standard. Society of Automotive Engineers, 2002.

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 on flexible product-line 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. This work has led onto the work introduced in this paper.

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.ukJohn 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.