

# Supporting Design of Safety-Critical Systems

Dr Mark Nicholson, MATISSE Project (GR/R70590/01), University of York, UK

## Introduction

One of the grand challenges in computing identified in a future oriented review of Computer Science by the U.K. Computing Research Committee is the evolution of dependable computing systems. In this paper, issues that can be addressed using evolution-based algorithms to support a solution to this grand challenge are introduced.

The Grand Challenge states that “Society’s dependence on computing systems is increasing, and the consequences of their failures are at best inconvenient; in certain application areas, they may also lead to large economic losses, and even loss of human life. A computing system is dependable if reliance can *justifiably* be placed on the service that it delivers, characterised in terms such as functionality, availability, safety, and security. Evidence is needed in advance to back up any manufacturer’s promises about a product’s future service, and this evidence must be scientifically rigorous. At the moment it is very expensive and difficult to produce such evidence.”

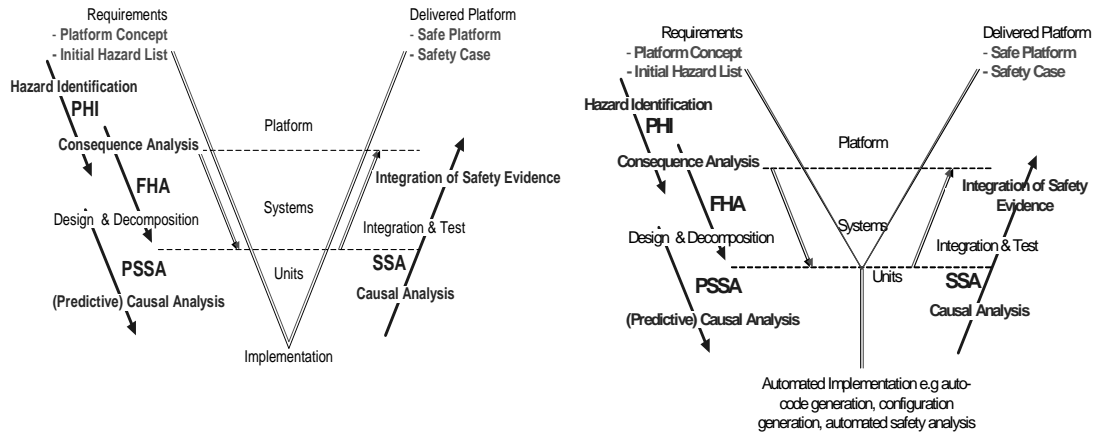
In Section 2 safety critical systems are introduced and the aims of a future development process for these dependable systems is presented. In Section 3 one future avionics architecture is presented as an exemplar of the trends and issues to be addressed by the introduction of a Y-model supported by evolutionary computation. In Section 4 the “allocation” problem is presented and a set of “blueprints” developed. In Section 5 future research issues are proposed. Finally, the conclusions indicate the rewards for successfully employing evolution based search techniques in this context.

## 2. Computer Based Dependable Control Systems

Safety-Critical Control Systems are employed in many products used in everyday life; for example electronic braking systems in cars, avionics systems in aircraft and signaling systems on the railways. The defining feature of these systems is that they not only have the potential to cause financial loss if they fail to act correctly in their operating environment but also can potentially lead to loss of life.

The current generation of safety-critical systems is typically developed using a V model [6] see Figure 1. Experience, with employing this approach has led to the

realisation that the need to build dependability into a design often leads to very time consuming and costly redesigns. Furthermore, the process has led to systems that require a great deal of analysis to be undertaken if a change is subsequently made to the system. Thus there is a need to evolve such systems rapidly, at costs, which reflect the size of change, not the scale of the system. The Y-model, see Figure 1, is likely to be a more appropriate development life cycle.



**Figure 1: V and Y lifecycle for Dependable Systems**

In the Y-model automated support is required in three areas. First, support for implementation via automated code generation, configuration generation and assessment of dependability. Second, support to address a number of “what-if” questions that can be asked during the iterative development phase of the system. Finally, automated support should be provided for implementing and analyzing changes during the lifetime of the system. Evolution based algorithms can be employed to provide the framework and solution engines for a number of aspects of a move to the Y development life cycle.

### 3. Integrated Modular Avionics (IMA)

The current generation of avionics platforms has been developed using a *federated* approach. In this approach the functionality required is split into a number of systems. Each system is developed independently and placed onto dedicated hardware. Quite large systems have been built this way. Unfortunately, complexity, resource usage, flexibility and change management issues have made continued development of systems this way infeasible.

Integrated Modular Avionics (IMA) [2,7] aims to bring the flexibility of distributed architectures, such as networks of PCs to aircraft applications. With IMA a number of functions are run on a processor, communicating via services provided by an operating

system (O/S). The overarching aim is to reduce the cost of developing and maintaining a system through its lifetime.

One standard form of IMA is the ASAAC [3] computing architecture that consists of three layers divided by interface layers see Figure 2. The O/S polices access by functions to the computing resources. To provide this service the O/S has access to configuration data comprising of which application process runs on which processor, what priority each process has, assuming a priority based schedule scheme [1], and which message uses which bus. This configuration information is stored in a run-time “*blueprint*” that is accessed via the Generic Systems Manager (GSM).

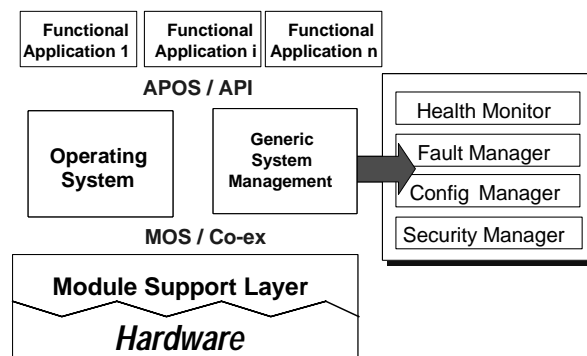


Figure 2: ASAAC Architecture

If a failure is detected via the health monitor, the fault manager determines the appropriate course of action. For instance, the processes normally resident on a failed processor may be moved to other processors, and the accompanying messages moved to different buses. If more than one failure occurs then some less essential functionality may have to be dropped, as well as a reconfiguration taking place.

#### 4. Solving the Blueprints Issue for IMA – First Steps

Consider the configuration and reconfiguration of a system via blueprints for a given IMA platform when all components are working and when any one hardware component has failed. This problem essentially is an allocation problem that can be characterized as:

*“given a hardware platform map the a given set of software components to the processing hardware and messages to the bus hardware such that a given set of requirements are met. Furthermore, given a hardware component fails re-allocate the software components and messages as required.”*

Each allocation becomes a blueprint in the run-time system. To address the allocation and re-allocation under failure (and change) issue the following process is followed:

1. Model the system and allocate the software components to produce a “baseline” system that meets all performance and safety requirements.
2. Produce a list of hardware platforms that result from any single hardware failure. Identify those systems that no longer meet their requirements.
3. For those systems that no longer meet their requirements, reallocate the software components to produce a new hardware-software mapping which does meet the requirements.
4. Produce a “reconfiguration blueprint” of allocations for each blueprint.
5. If there remain failed systems for which there are no allocations that meet all requirements determine which software components need to be dropped and revisit step 4 to produce a new configuration.
6. Indicate any failures for which an acceptable reconfiguration cannot be found. Is there a minimum safe-state that the system can be put into?
7. Investigate the *mode change* problem for each possible blueprint
8. Revisit 1 to 7 for multiple failed components?

So how is a baseline allocation determined? There are four steps: produce a representation of a solution, determine the quality of that solution using a fitness function, determine a way of changing any solution into another solution and automate the process of finding the best option amongst the many possible solutions. Space limitations have led to only one aspect of this approach being considered here.

What is the quality of a proposed allocation? The fitness function is split into two parts: primary and secondary objectives. For the allocation problem the primary requirements are for no worst case response time (WCRT) requirements to be missed, no resource constraints to be violated and no fault tolerant software components to be placed on the same processor as other replicas of the same component. The secondary requirements include minimising the number of processors / buses used and minimising WCRT of each component (given all WCRT have been met).

The penalty for a primary requirement ( $k_1(G_x)^2$ ) is determined by adding up the degree to which the proposed solution misses the requirement and squaring the resulting number. The penalty for a secondary requirement is a simple linear function ( $k_2F_x$ ). The value  $k_x$  is a weighting factor that indicates the acceptable *trade-offs* with other elements of the fitness function. Thus, the fitness function for a problem embodies the set of good versus bad trade-offs between different design solutions, since all possible solutions will be scored via this function.

Automated searches have been undertaken on the allocation problem with some success. For instance, Nicholson employs Simulated Annealing hybridised with elements of Tabu Search [4,5].

Assuming that a baseline can be produced how can the process be extended to look at reconfiguration on failure, or a change to the system? Suppose that in a three-processor example onto which three components that are fault tolerant copies have been allocated, one of the processors failed. Since there are only two processors left one of these components can be dropped. Thus the problem becomes one of placing four components onto two processors. Once the first failure has occurred, and a redundant has been dropped, the reliability of the system in the next period of time has decreased over that in the original configuration. Any decision to allow the aircraft to fly on it's next flight with the failure still on board will need to be take this failure into consideration.

The reconfiguration algorithm is employed for every new architecture resulting from a single failure of a hardware component. Each resultant allocation can be placed in a look-up table that forms part of the set of runtime blueprints available to the GSM. It would be possible to extend this to look at reconfigurations for two failures, etc. However, the number of scenarios to be explored becomes an issue. It may be worth defining a default safe-state configuration to go to if necessary.

In this Section it has been shown that a Y-model approach can in part be supported by automated search for an acceptable allocation of software components to a hardware platform, such as an IMA platform for a fully working system, one subject to hardware failures and for one subject to change.

## **5. Extensions to the Approach**

The issues considered in Section 4 have been applied to a number of different architectural styles, including IMA, and control problems both of an academic and industrial nature. However, these techniques potentially can provide even greater benefit by allowing the implications of different design decisions on the eventual size, and form, of the hardware and software architectures, as well as predicting whether appropriate allocations exist, to be addressed.

For any given proposed set of software components resource usage and worst case execution time (WCET) budgets can be set during the design process. A hardware architecture can then be proposed and a prediction made as to the ability to produce an appropriate set of blueprints for this proposed system. Thus the impact of a design change, such as the introduction of extra software components to provide extra fault tolerance and reliability, on the likely blueprints can be investigated. This gives the designer useful information on the implications of such a design change. This is very much an area of future research, but some first steps have been taken [4,5].

## 6. Conclusions

In this paper it has been stated that the Grand Challenge of producing a process by which evolvable dependable computer based control systems can be developed and maintained requires a range of problems to be addressed. It requires automated tool support for aspects of system (especially architectural) design, automated implementation of the design and support for changes to the design during the lifetime of the system. In this paper it has been proposed that evolution based algorithms can provide the basis of solutions to a number of problems in this area. Specifically, attention has been drawn to work on the evolution of a fault tolerant architecture for reliability and the allocation of software items to this architecture in such a way that the resource constraints and timing characteristics of the required dependable system can be met.

The implication in the “real-world” of employing such techniques is that products will be introduced into service more quickly and will be quickly upgradeable to keep pace with the demands being placed on the system during its lifetime. Furthermore, the designs being produced should be more efficient and analysable for their dependability characteristics, especially safety, giving more confidence that the level of risk to the consumer, and general public, from the system is as Low As Reasonably Practicable. A number of issues especially related to design support and the set of characteristics covered; remain to be addressed to achieve this.

## References

- [1] Burns, A. Wellings, A. “Real-time Systems and Programming Languages (3<sup>rd</sup> Ed), Addison Wesley Longmain, March 2001
- [2] Conmy, P., et al. Safety Analysis and Certification of Open Distributed Systems. in International System Safety Conference. 2002. Denver, USA
- [3] Multedo, G., D. Jibb, and G. Angel, ASAAC Phase II programme progress on the definition of standards for the core processing architecture of future military aircraft. *Microprocessors and Microsystems*, 1999. **23**: p. 393-407
- [4] M. Nicholson, “Selecting a Topology for Safety-critical real-time Control Systems” YCST 98/08 Dphil Thesis, Department of Computer Science, University of York (1998)
- [5] M. Nicholson, P. Hollow and J. A. McDermid, "Approaches to Certification of Reconfigurable IMA Systems", INCOSE 2000, Minneapolis, USA, 17-20 July 2000
- [6] SAE-ARP4754 Certification Considerations for Highly Integrated or Complex Aircraft Systems, Issue 1996-11
- [7] VICTORIA, Validation platform for Integration of standardised Components, technologies and Tools in an Open, Modular and Improved Aircraft electronic systems, EU grant GRD1-2000-25209, 2001 - 2004