Identifying Safety Dependencies in Modular Computer Systems

Philippa Conmy, Department of Computer Science, University of York, UK
Mark Nicholson, Department of Computer Science, University of York, UK
John McDermid, Department of Computer Science, University of York, UK

Keywords: modular computer systems, safety analysis

Abstract

The aviation industry has started to adopt complex distributed computing networks (known Integrated Modular Avionics (IMA)). IMA raises a number of new issues for syst development. Firstly, IMA is designed to use logical partitioning of data, both to provide secu of access to resources and also to support incremental change of one component with limi impact on other components. Secondly, an incremental approach to development is often need for example to purchase an operating system prior to application development. To support t there is a need for a modular approach to safety analysis which not only provides assurance system safety, but also provides a breakdown of responsibilities across logical boundar However, current safety analysis techniques are not well suited to a modular approach, treat software as a monolithic entity. This paper presents a method used to perform safety analysis the module support layer (MSL) of an IMA system as part of a modular approach. The approa takes the system context into account, using system level hazard analysis and compon interaction diagrams to assess the impact and cause of failures occurring in each component. discuss the advantages of this approach, such as potential support for incremental chan However, we also highlight some difficulties associated with the component based approa particularly when considering the re-use of components such as the MSL.

Introduction

Over the past few years the aviation industry (both civil and military) has started to adopt an o distributed system known as Integrated Modular Avionics (IMA). This type of system contain number of networked computer modules. The software within these modules is layered in or to separate applications, operating system and hardware specific code. This type of architect provides many potential benefits such as incremental change or replacement of hardware software components with minimal impact on the other components. It is useful for manufacturers to design and purchase the supporting software, such as an operating system, p to the applications in order to provide an accurate development environment for the applicatio This means safety analysis needs to be performed for the OS prior to the development of applications. However, current safety analysis techniques do not support a staged or modu process for software safety assessment.

This paper presents a process which can be used to perform early safety analysis on the OS Module Support Layers within an IMA system, deriving safety requirements based on the mo and design of the final system including the applications. These safety requirements form a se guarantees which a component in the system must meet in order for the fully integrated syster fulfil its safety requirements. These guarantees also form the basis for a set of “safety contra between the components. We demonstrate how this approach has been applied to the MSL wit a commercial IMA system, showing how analysis of this component provided a set of requi guarantees not only for the MSL but also for other components in the system.

This paper is laid out as follows. The first section contains a general introduction to IMA and IMA system being analysed. The next section presents a general process for analysis of individ IMA components. Then the results obtained when this process was applied to the Module Supp Layer within the IMA system are presented. Finally, conclusions and further work are discusse

Integrated Modular Avionics

IMA systems are often referred to as "open distributed systems", where open implies that a set common standard interfaces are used for components to interact. There are two main standa for IMA, a civil standard based on ARINC 653 [1], and a military standard from the ASA council [2]. The IMA system analysed for this paper was based on the latter standard, theref this is the version now introduced. The software architecture for a single computer process module is shown in Figure 1.

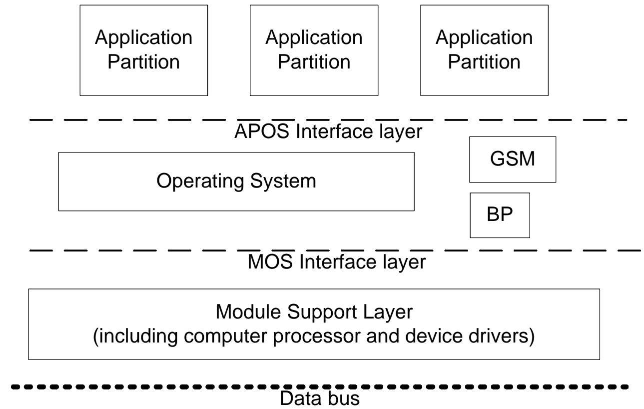

Figure 1 IMA Module

There are basically three layers within each module. At the bottom a layer of hardware spec software is used to control the computer hardware and to provide access to its resources. This known as the Module Support Layer (MSL). The MSL also contains device drivers to connect the data bus. Above this is an operating system layer. This consists of three components, the m Operating System (OS), the General System Manager (GSM), and BluePrint manager (BP). T GSM provides health monitoring information. The BP controls the system layout informati known as a blueprint. This includes each applications required resources, permit communications between applications and also scheduling requirements. The OS servi requests from the applications and interacts with the MSL, using data from the GSM and BP order to validate requests and also initialize the system. The OS communicates with the M using a fixed software interface. The advantage of this design is that new processing hardw can be inserted into the system, and as long as the new MSL has the same interface, the ot layers can be moved to the new hardware without a need to alter the code.
Above the OS layers are one or more application partitions. These contain software wh performs avionics functions, such as navigation, or flight control. Again, a standard interface used to communicate with the OS in order to permit change or replacement of components w minimal impact.

The IMA system analysed for this paper consisted of a small network of modules runnin navigational display system. The precise details are not given here for reasons of commerc confidentiality, however enough detail is given to illustrate some key results from the analy Two key failure modes were identified at the system level, display of incorrect data and loss display. These are associated with hazards such as controlled flight into terrain. These failures partially mitigated by an independent display system and also pilot observation. The te developing the system wished to purchase the MSL software and hardware prior to development of the application software. This meant that some safety analysis of the MSL required in order to ensure system safety once all the components were developed and integrate

Analysis Process

This section describes a generic process which can be used to assess individual components in IMA system. The process is in three parts. Firstly system level analysis and decompositio undertaken. Secondly analysis of individual components is undertaken. Finally, the Deri Safety Requirements (DSRs) found from the analysis are collated for each component.

The overall analysis process is summarised in Figure 2 and detailed below:

- ??Stage one:
  - Assemble system analysis team. It is useful to have representatives from e supplier at the meeting to avoid misunderstandings and misrepresentations of components in the system.
  - Examine the system as a whole and undertake traditional analysis to establish syst level hazards
  - Agree system design, producing a system decomposition model for each compone Note the role of each component at a high level of abstraction.
  - Assess the sequence of events and produce representation of event paths through components in order to assess which components have a direct impact on another and assure system safety.
  - Assess the services each component provides in more detail as a pre-cursor analysis
  - State any assumptions made about components (e.g. if not designed yet)
- ??Stage two:
  - This stage applies to each component. Using any Failure Modes and Effe Analysis (FMEA) technique derive possible failure modes which could occur in t component. Assess effects of the failure modes at a system level as far as possible should be noted that causes of failures could be other components. When deriv safety requirements consider which component needs to address a particular failu Any requirements for other components should be noted.
- ??Stage three:
  - Collate and consolidate derived safety requirements for each component (as sho in Figure 2). There will be three types of requirement:

    1. Requirements from this components analysis which address internal failures.
    2. Requirements on this component which address external failures (in ot components)
    3. Requirements on this component which are the results of another compone analysis.

    Additionally, there may be “if – then” requirements where the results depend behaviour of another component.
  - Using unique identifiers for requirements, failures and components ensure those t those failures which cross component boundaries (categories 2 and 3 above)

listed along side the relevant derived requirements. This captures interdependencies between components. In addition these types of requireme provide the constraints for upgrading a component.

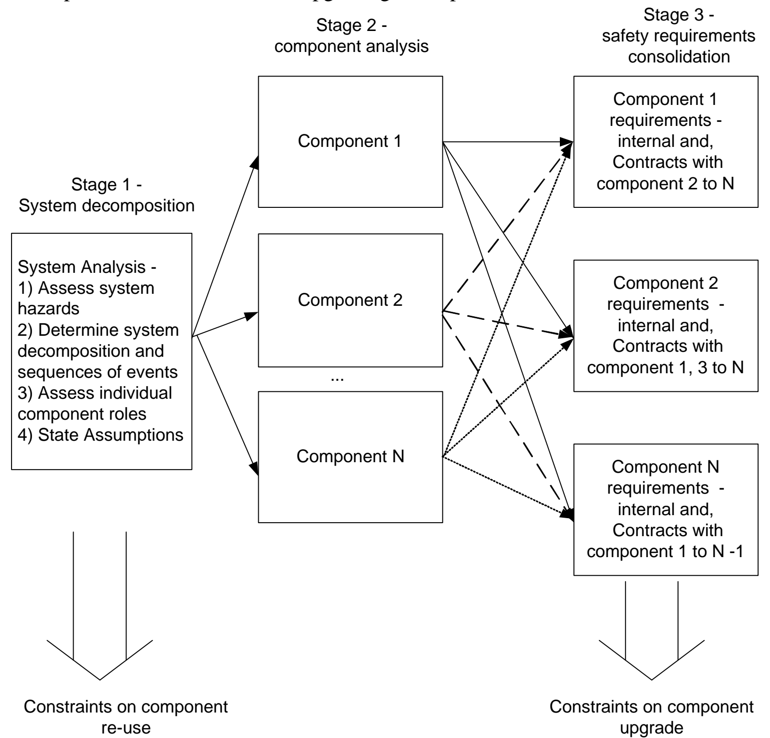

Figure 2 IMA Analysis process

<u>Comments</u>

Stage one of the process is designed to generate some context for the analysis of individ components. Then when stage two is undertaken the effects of component failure modes can tl be assessed at a system level. This is particularly important when performing analysis of a gene component such as the MSL as without any context it is impossible to determine whethe predicted failure mode is either credible or what its consequences may be. Further, with understanding its consequences it is impossible to determine whether any action is required deal with that failure. It could be assumed that any failure is credible, with severe consequenc and hence must be dealt with. However, this would lead to very inefficient and bloated softwa In addition, it may not be appropriate for the component actually being analysed to deal wit failure (e.g. for an application to deal with a hardware failure).

There is no easy guidance on the level of detail required as it will differ from system to syste However, at the highest level a one line sentence describing the component could be used, a

when examining the components in a little more detail a list of essential services, and perh peculiarities could be used.

Undertaking the process for just the MSL was made easier by assembling an analysis team wh included suppliers of other components. Failure modes of these other components could assessed as to their credibility, and appropriate roles and responsibilities of the other compone understood. Additionally, it is important to reassess the system decomposition when the ot components are analysed to ensure its validity. This reassessment may mean components wh have already been analysed, such as the MSL, may need to be re-analysed and even engineered! In this particular instance because representatives from each component supp were present during the initial phases of analysis we have a high degree of confidence that analysis is valid. It is highly recommended that all suppliers are involved when analysing system.

Capturing inter-dependencies between components in the requirements consolidation phase is key to both ensuring that the initial system will be acceptably safe, and to ensuring any update a component will be acceptably safe. In the latter case, if a component still meets requirements and constraints placed on it by other components then the impact of its alterat should be limited. Where one component relies on another to address a particular failur "contract" is formed between the two. This contract should describe the interdependenc between the components, but at a high enough level to allow the specific details to be hidden one component from another.

On the left hand side of Figure 2 is a comment noting that the initial system analysis provi constraints on component re-use. Basically, this indicates that as the safety analysis is undertal within this system context, a component may not necessarily be safe outside of this context. the right hand side is a comment noting that the results provide constraints on compon upgrade. This means that when a component is altered in some way it is constrained by the saf requirements of the system.

Analysing the Module Support Layer

This section describes and comments upon the analysis undertaken for the MSL component, st by stage.

Stage 1: System analysis and decomposition

A team consisting of the application designers, system designers and MSL supplier assembled for the first stage of analysis. A system architecture diagram was produced, overl with arrows to indicate data flow into the system, between the layers, and out to the display. identifying the top level system hazards, and determining the sequence of events through components in the system it was possible to examine the effects that a failure in an individ component could have at a system level. For example, the MSL could corrupt calculated disp data after it had left the main application and then pass that data on, and hence incorrect d would be displayed to the pilot (potentially causing a system level hazard). Thus a set of deri requirements needed to be developed for the MSL.

A number of basic assumptions were made about the system and its components. As this anal concentrated on the MSL, the assumptions made concerned the functionality and responsibili of components with direct interactions to the MSL such as OS and applications. The assumpti made were as follows. Firstly, it was assumed that the OS and applications worked broadly specified. That is, they may still fail in some way, but are designed and will perform th

individual tasks in the manner specified. For example, when a communications request is ma the OS will check the validity of the request against the blueprint but may return an incorr value due to some kind of software bug. However, it is assumed it is the responsibility of the to perform this task and any fault management associated with it.

Once the system decomposition was agreed, an assessment of the essential services provided the MSL was examined, using its current specification. The important features included not o services provided via direct interface calls, but also indirect calls. For example, mem management and context switches are not explicitly called but they are essential services for application and OS.

The overall process of system decomposition took some time for the analysis group. In particu specifying the MSL services and agreeing a place to start the analysis was difficult. Agreeing the correct level of abstraction was also difficult. For example, should individual MOS calls used, or grouped together? Should each scenario be examined? In the end it was decided that services should be grouped, and the general effect on each of the other components considered. For example, the context switch was chosen as the first service to analyse, sim because it takes place so often, and the effects of failure to provide this service were conside for the OS and Applications.

Stage 2: Component Analysis

The next stage in the process is to analyse each individual component. The process is intended allow any kind of Failure Modes and Effects Analysis (FMEA) to be undertaken for a compone The analysis process used for the MSL was based on the SHARD approach [3]. This techni uses five guidewords to suggest possible failure modes for an interaction between t components. These guidewords are shown in **Table 1**. The cause and effects of the failure then assessed, and DSR determined to address the failure.

Guide word	Meaning
Omission	A service is not provided
Commission	A service is provided when it is not required (a perfectly functioning syste would have done nothing)
Early	A service is provided early in relation to an expected time frame
Late	A service is provided late in relation to an expected time frame
Value	There are two types of value error to be considered – Detectable/Coarse ar Undetectable/Subtle

Table 1 SHARD guidewords

One of the most frequently used functions of the MSL is the context switch. A context swi occurs when the MSL is passing access to the processor from itself to either the applications OS so they can execute. A further context switch occurs when control is returned to the MSL the end of the processing time slot. A selection of the analysis results for the context switch shown in **Table 2**. The complexity of some of the results reflects the dependencies between different layers and components. For example, the MSL is obliged to copy data from communications bus without interruption to avoid data corruption. However, if data is l arriving and hence the MSL is late completing a data copy, this could have a knock on effect preventing the context switch to an application from occurring on time. This may in turn m the application does not complete its execution. The effects of this are difficult to predict, depend on whether the application will restart from scratch during each processing slot

whether it is re-entrant (i.e. will restart from the position it stopped at after a suspension at the e
of its time slot).

The results also show why it is essential to have knowledge of the system context for the M
analysis. An example is the value failure where incorrect values are returned to the application
the context stack, potentially meaning incorrect display data is generated, and thus shown to
pilot.

The analysis has thrown up instances where a derived requirement is made on anot
component. For example, if a context switch is made early then some OS processing requested
the application may be incomplete. The only method of detecting this by the application is in
OS results passed back. Therefore it is essential that the OS set all return values to "failed" u
the last commands in the sequence. Figure 2 shows the links between the sets of deri
requirements for each layer. It is important to capture the requirements on other components a
pass them on to the component supplier. This helps ensure interoperability.

Deviation	Causes	Effects	DSR
Omission Switch to correct app not made	Entry point in saved table is corrupted - OS or App has corrupted area, random failure. Context switch is interrupted by something else Processor stops / failed. Deadline for process been reached (process overrun, timeout in OS)	Switch to wrong application Control is not returned to application -> timeout Unintended process is executed -> Very difficult to predict effects of omission when application is re-entrant. Therefore must stop causes.	MSL s/ware should not switch to invalid entry context/wrong app MSL must detect writes outside memory area MSL must implement watchdog for failed/stopp processor (e.g. in initialisation)
Commission	See omission	Application run when not required	As for omission
Early	Processor failure Systematic software failure	Switches back to application too soon: Data could be corrupted/data not copied / partially copied. Application unaware of failure	Alternatives: Shall not happen or/(and?) O/S set status to failed as first thi on data copy so app can detect (no MSL requirement)
Late	Data held up on bus Omission of data copy MSL is in critical section so cannot be interrupted	Data has arrived but too late for applications to complete Stale data at output Incomplete processing of app when end of slot occurs (strategy for restart required)	If we start at beginning o app each time then lose context data and entry po If start where stopped the must save context data ar save entry point – re-entr (MSL must keep itself in consistent state must be a to pass control back to O this is to do with critical section requirements)
Value wrong app	See omission re-entrant	Wrong application run	As for omission
Value	Restore incorrect parameters /values in application stack Restore incorrect parameters /values in OS stack		For this system not too ba from app point of view– there is pilot perception a range limiting

Table 2 Analysis Results looking at context switch between MSL and generic application

Stage 3: Consolidation of Analysis results

The final stage in the process is to consolidate results for each component in the system. At time of writing this stage had not been fully completed, so the initial results only are shown he The DSRs for many of the failures shown in **Table 2** appear more than once. For examp omission, commission and value failures all have the DSR that the "MSL must detect wri outside memory area". This requirement also appears in other sections of the results (not sho in this paper) such as data transmission where an incorrect data address may have been sent to OS, and hence data may be written at an invalid address. All the DSRs need to be consolida into a single table for each component in order to ensure that they are consistent. They can also numbered within the consolidated table, and the analyses tables to capture traceability fr failure to solution.

As discussed above, the analysis of the MSL also produced safety requirements for ot components which needed to be captured. As these are linked back to failures in the MSL traceability across components is also important. As discussed earlier, when one component rel on another to address a particular failure a "contract" is formed between the two. This contr contains the dependencies between components. The importance of contracts is not just to ens that the initial system is safe, but also to allow upgrades of components to be undertaken w limited impact on other components. When a component is altered (e.g. the MSL is upgraded must adhere to its contracts. If the DSRs within the contract can still be met then the integra system should still be acceptably safe, and no alteration should be required to the ot components.

An example of a contract here is "MSL must detect writes outside of memory area". This D protects against failures in both the OS and applications, preventing data corruption of mem space. This requirement is therefore a contract with the other two types of component. If processor, and hence the MSL, is upgraded it must still meet this requirement as the failure still occur. However the means by which the MSL meets the requirement can change.

Table 3 shows a method for capturing the interdependencies across components, using the f results row of **Table 2** as an example. The first derived requirement, MSL_DSR_1, addres failures in two external components – applications and the OS. Therefore when upgrading MSL this derived requirement must still be met. The second requirement, MSL_DSR addresses an internal failure. In this case if the failure still exists then it must be addressed by MSL, but if it doesn't then the derived requirement is no longer required. Clearly the failures a need to be numbered or summarised within the analysis results tables in order to use this metho

Derived Requirement	Derived Requirement No	Failure + cause	Failure No
MSL must detect writes outside of memory area	MSL_DSR_1	OS or App has corrupted memory area	App_F1, OS_ F1
Set maximum numbers of interrupts and hence re-entrance entries in tables.	MSL_DSR_2	No more space in entry table	MSL_F1

Table 3 MSL Consolidated requirements

Further discussion

One of the potential benefits of IMA is the ability to re-use components, such as the OS i
different context. However, this process uses a fixed system context to analyse gene
components such as the MSL and this may mean that there are limitations on how the finisl
component can be re-used. For example, the MSL design was analysed for a display system or
Most hazardous failures for this system involve data corruption. If the MSL was to be used in
IMA system which supported an application such as flight control, timing failures may be mu
more critical. One way around this problem would be to list other likely scenarios for re-
during the first stage of the process. This approach has the advantage that the MSL is develop
and built for more scenarios. However, it may also mean that the MSL is over-engineered fc
particular project and hence may be more difficult certify for that project.

The other potential benefit of IMA is the ability to upgrade components with minimal impact
other components. This process supports this feature via the generation of DSRs which h
specific implementation details from the other component suppliers. For example, say the MSL
an IMA system is required to ensure data integrity to a certain probability and it ensures t
using hardware supported methods. If the processor, and hence the MSL were upgraded then
long as the same level of data integrity is assured then the assumptions made by ot
components are still valid, even if the MSL has to use a software based method to ensure d
integrity.

One potential problem with the staged development process in general (rather than the analy
process used) is that when other components are analysed later in development further deri
requirements may be found for the components which may already have been complet
Potentially the completed items may need to be altered. There is no obvious solution to t
problem, however, it is hoped that the use of a guideword based technique in this particu
instance has provided enough coverage of those OS and application failures which effect
MSL to avoid any major requirements being missed.

The system examined was relatively simple, in that only one application was running on the IM
system. If more applications were to be used the analysis would be lengthier and it wo
potentially be more difficult to assess derived requirements due to conflicts in applicat
requirements. The ability to consolidate these competing requirements is a major feature of a
future "design by contract" approach. Additionally, it should be noted that scenarios for the re
of a component might need to be examined to avoid potential future conflicts. This should
born in mind when performing the analysis in order to allocate adequate time and resources to
team.

Conclusions

This paper has presented a process for analysis of individual IMA components. The proc
supports the incremental development of IMA systems often required by airframe manufactur
This process has been applied to the MSL for a simple IMA based display system, generatin
set of derived requirements on all system components. This methodology also suppo
incremental change as it captures safety requirement dependencies between components. Wh
one component relies on the service of another to address a particular failure a "contract"
formed between the two. Further work is being undertaken at York to develop these contra
fully by examining exactly how a component meets its safety requirements. This will generate
full set of constraints upon a component relying on that service which will not only impro
assurance of interoperability, and provide better support for incremental change.

References

1. ARINC, *Avionics Application Software Standard Interface ARINC 653*. 1997: Aeronautical Radio Inc.
2. G Multedo D Jibb, G.A., *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.
3. Pumfrey, D.J., *The Principled Design of Computer System Safety Analyses*. 2000, Department of Computer Science, University of York.

Biography

Philippa Conmy is a research associate within the Defence and Aerospace Research Partners at the University of York. This is funded jointly by the EPSRC, UK DTI, BAE SYSTEMS a Rolls Royce. She attained an MSc in Computer Science from the University of Leeds in 19 Prior to her current position she was a research associate within the Dependable Compu Systems Centre at York. She is currently studying for a PhD, examining techniques for saf analysis of component-based systems.

Dr. Mark Nicholson is a teaching and research fellow in the High Integrity Systems Engineer group at York. He attained his PhD in 1998 examining process allocation in real-time distribu systems. He has over 7 years of experience examining safety assessment of real-time systems. I current research interests include IMA certification processes and the evidence required for use off-the-shelf operating systems in safety critical applications. He is a member of the MATIS project.

Prof. John McDermid has been Professor of Software Engineering at the University of York si 1987 where he runs the High Integrity Systems Engineering (HISE) research group. HI studies a broad range of issues in systems, software and safety engineering, and works clos with the UK aerospace industry. Professor McDermid is the Director of the Rolls-Royce func University Technology Centre (UTC) in Systems and Software Engineering and the B SYSTEMS-funded Dependable Computing System Centre (DCSC). He is author or editor o books, and has published about 250 papers.