

Safety of Data in Real-Time Distributed Systems

Allan S Wake

12th September 2008

Submitted to satisfy the requirements of the project module for:

MSc in Safety Critical Systems Engineering

Department of Computer Science

University of York

SUPERVISED BY: Mark Nicholson, Department of Computer Science

Abstract

This research addresses the safety of data used within real-time distributed systems, using Integrated Modular Systems (IMS) as an example architecture to investigate the safety issues.

Two aspects of the data safety problem are assessed:

- The use of large-scale configuration files to condition generic software
- The use of distributed concurrent data to provide a distributed system state.

Both of these issues are relevant to real-time distributed systems in general, but are specific problems in current IMS architectures. In order to understand these issues relevant research has been evaluated in the areas of:

- IMS
- Safety Standards
- Data Driven Safety Critical Systems
- Distributed Systems
- Safety of IMS
- Data Consistency in Distributed Systems

A safety process for data within IMS is proposed, specifically aimed at the generation and validation of blueprint data. In addition, information models and data models are defined for the two forms of IMS data.

Safety argument modules are then proposed which are designed to be incorporated into existing modular safety case architectures developed for IMS.

For the distributed state data problem a scenario is developed to allow the investigation of the safety concerns. The consistency of the global state, made up of interacting distributed local states, is identified as a major safety concern. The scenario is then used to propose a solution to this data consistency issue. This proposed solution is the extension of a thread-based consistency mechanism Coordinated Atomic Actions (CAA). This concept is then extended to allow its use across a distributed set of interacting processing nodes, resulting in an extended model, Distributed Coordinated Atomic Actions (DCAA). This also includes a distributed synchronisation model used to initialise a DCAA operation.

As part of the evaluation of the DCAA model, advantages of a hierarchical model of system states are identified, and therefore this hierarchical model of states is proposed for use in distributed state data.

Finally the proposed safety arguments are evaluated to ensure that the required evidence to support the arguments can be acceptably produced.

Conclusions are then presented, together with a list of suggested further work on DCAA modelling and other approaches to solving the IMS safety issues.

Table of Contents

Safety of Data in Real-Time Distributed Systems 1

 Abstract 2

 Table of Contents 3

 List of Figures 3

 Introduction 5

 Definitions 7

 Aims of the Research 9

 Literature Review 10

 Integrated Modular Systems (IMS) 10

 Safety Standards 22

 Data Driven Safety Critical Systems 29

 Distributed Systems 35

 Safety of IMS 37

 Data Consistency in Distributed Systems 42

 Proposed IMS Data Safety Process 48

 System of Interest 49

 IMS Safety Process 51

 IMS Data Safety Case 52

 IMS Data Lifecycle 57

 IMS Data Model 58

 Evaluation of IMS Data Safety Process 78

 Evaluation of the data model 78

 Evaluation of the Safety Argument Structure 91

 Conclusions and Further Work 96

 References and Bibliography 99

List of Figures

Figure 1 - Basic IMS Architecture Model 13

Figure 2 - IMS Software Layers 14

Figure 3 - IMS Software Components 14

Figure 4 - IMS System Management Hierarchy 17

Figure 5 - IMS System Management Functions 19

Figure 6 - IMS Management Component Interactions 19

Figure 7 - Typical IMS Management Hierarchy 20

Figure 8 - Nicholson's GSN Model for an IMS Safety Argument 38

Figure 9 - Joliffe's GSN Model for a Blueprint Safety Argument 39

Figure 10 - Kelly's GSN Model for an IMS Safety Argument 40

Figure 11 - Coordinated Atomic Action Model 46

Figure 12 - Spectrum of Data Models in IMS 48

Figure 13 - IMS Software Components on Processing Node 49

Figure 14 - Basic IMS System Management Hierarchy Example	50
Figure 15 - IMS Example System Model Diagram	50
Figure 16 – Blueprint Development Safety Argument Structure	54
Figure 17 - Blueprint Use Safety Argument Structure.....	55
Figure 18 - Distributed System State Safety Argument Structure.....	56
Figure 19 - IMS Health State Example	59
Figure 20 - IMS Mode State Example	60
Figure 21 - Complete IMS state diagram example.....	62
Figure 22 - IMS State Data Model.....	74
Figure 23 - Simple Health State Change Sequence Diagram.....	79
Figure 24 - Health and Configuration State Change Sequence Diagram	80
Figure 25 - Invalid State Change Sequence Diagram.....	82
Figure 26 - System State Transitions Constrained Through System Level	83
Figure 27 - System State Change – No Integration Areas.....	84
Figure 28 - Scope Choices for Distributed Coordinated Atomic Actions.....	86
Figure 29 - Multiple State Changes Using DCAAs.....	88

Introduction

The problem of safety assessment in computer systems has always been a significant part of the system development lifecycle. With the increase in complexity, size and integration of today's new systems this problem is becoming more difficult to solve and this issue is even more significant when we consider the data used within the system.

Distributed systems are becoming more dependent upon the data they use, rather than just the functionality that the software itself provides, and although there are many approaches to constructing a safety argument and gathering safety evidence when considering the functionality of a system, this is not true for the data within the system.

The distributed nature of the systems now being used provides its own challenges. A traditional system can easily access data stored throughout the software as it all resides on a single resource. However, a distributed system contains multiple resources, each with its own data. Managing this data distribution, particularly for such significant information as the overall system state, can lead to increased complexity.

Adding to the challenges of safety assessment is a new reliance on large amounts of data that are used to define the system behaviour. Traditionally the safety of the system has been assessed in terms of its functional effects as part of the system under test:

- Static analysis of the structure of the software
- Dynamic analysis of the software using testing
- Analysis of the processes and methods used to develop the software

These traditional systems manipulated data and used data as part of the overall system functionality, but the safety of such data was assessed as part of the structure of the code. We are now seeing high-integrity systems being deployed that use data in a different manner. Today's new systems use large amounts of data to condition the functionality of the software itself.

In order to show how the use of data has changed we can define two examples, the first showing the traditional use of data within a system, and the second showing why data must now be considered using a new approach.

The traditional use of data within a system can be characterised by the use of data to perform a specific operation within the system, such as a calculation. This form of data use within the system can still require large data sets, but the resultant system behaviour can be determined by the specific data and operation. The changes to system behaviour due to this type of data use are of limited scope within the system. A system that includes this model of data will usually display the following attributes:

- Highly integrated and monolithic system functionality
- Local states independently maintained throughout the system
- Reliance on statically programmed software algorithms to define system behaviour

An aircraft fuel system is an example of such a traditional high-integrity system that requires significant data to perform the fuel system calculations. This data provides a representation of the fuel levels in each of the fuel tanks in the aircraft, and is used to provide information to the pilot. It is obvious that the data within the system has an

impact on the safety of the aircraft, and as such must be assessed as part of the overall safety argument for the system. However, traditional safety approaches can be utilised in order to do this. The functionality of the software itself is not modified by the data it uses it to make decisions. Therefore the safety of the software can be assessed on the basis of the data ranges that are used to perform the computations within the system. Also, there are a defined, and manageable, number of computations that depend upon the data, and the manner in which they interact is statically defined in the structure of the code. Finally, the data within the system is limited by the structure of the software.

The second model of data use within a system is more complex. Here the data forms part of the system behaviour and the operation of the system itself is more directly dependent upon the data, rather than the algorithms contained within the software. This is due to the amount of data they process, and the use of data to condition generic functionality within a system or to configure the system. A system that includes this model of data will usually display the following attributes:

- A requirement for real-time behaviour
- Distribution of system functionality
- Shared state across the distributed system
- Reliance on the shared state to define system behaviour

In a system that displays these attributes we can see that the data does not conform to the constraints that reduced the problem in the traditional system. In these cases the data itself is responsible for some of the safety attributes of the system. An example of such a system is the Integrated Modular Systems (IMS) approach used in avionics systems, utilising a common software infrastructure to host various avionic applications. The infrastructure uses distributed data to condition the operation of the system, defining the applications used by each infrastructure instance and the communications between them.

We can identify two distinct uses of data within the system that differentiate IMS from traditional systems:

- the initial configuration (and subsequent planned configuration) information for the distributed system nodes. This is contained in a set of configuration files for the processing nodes, called the blueprint. This provides a large amount of data used by the IMS system components to “instantiate” the generic IMS software to support the required application configuration.
- an overall dynamic system state that defines current and future system behaviour. This system state can be modified by specific distributed nodes of the system and this modification may be the result of a number of independent decision paths. In the example of IMS the overall system state is determined from both mode change and fault conditions, and these conditions can be generated in any of the nodes within the system. In such a system the determination of the system state is a complex problem and the prediction of the system behaviour is also complex.

The type of system exemplified by IMS can be referred to as a **Distributed Data-Driven System**.

Definitions

The following definitions help understand why the data requirements of these systems need a different approach to safety. In order to categorise the systems that display this new approach to the use of data, we will mainly consider the example of a distributed real-time embedded avionics system that utilises these data-driven approaches. So, developing each of these definitions in turn:

Real-Time – A real-time system definition - “A system is defined as being real-time if it is required to respond to input stimuli within a finite and specified time interval. The stimuli being either an event at the interface to the system or some internal clock tick that is, at least notionally, coordinated with the passage of time in the system’s environment” - [1]. This definition is classically used to define systems such as avionics, mission systems, and control systems. A real-time system increases the difficulties in management of consistency of distributed data, as most real-time systems (but not all) require response times which do not allow extra time to be taken to perform consistency checks across the whole system before the system state is altered. In effect the system cannot “stop” to check consistency before continuing.

Distributed System – The following definitions of a distributed system highlight the difficulties with maintaining an overall system state:

1. “A distributed system is a piece of software that ensures that: a collection of independent computers appears to its users as a single coherent system” – [2]
2. “A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive the system as a single, integrated computing facility.” – [3]

Both of these definitions define an overall system that is made up of a set of distributed independent computing nodes, each able to maintain its own local state, but that to an outside observer the system as a whole displays a single consistent global state. This combining of a set of node states into a global state creates problems with defining safe states and global state consistency.

Data-Driven Architecture – “A data driven architecture/language performs computations in an order dictated by data dependencies. Two kinds of data driven computation are dataflow and demand driven” – [4] – This definition supports the view that a data driven system’s functionality is directly related to the data and not just to the structure of the code.

Data-Driven Systems - “Data-driven systems offer the opportunity to modify the data component and hence influence the behaviour of the system without changing the hardware and software components.” [5]

Embedded System – “Hardware and software which forms a component of some larger system and which is expected to function without human intervention” - [4]

There are a number of safety concerns with this complex model:

- Determining the consistency of the initial configuration data – The initial system state is defined by a complex set of configuration information that must be consistent in order for the system to start in a safe state.
- Determining safe states for the nodes of the system – The set of states that each node of a distributed system can adopt is determined from the combinations of inputs used to determine the state. For any non-trivial set of inputs, the number of combinations can be significant. However, not all of these possible states are necessarily safe states for the node, as not all possible combinations of input may be acceptable. It is necessary to determine which of these input combinations / possible states are safe states for each node.
- Determining possible safe state changes at system nodes – When the set of safe states for each node has been determined it is necessary to understand what paths through this set of states will lead to safe behaviour of the node. Even when the set of safe states is known it is not necessarily safe to move from any one state to any other. The possible node state changes must be understood and the constraints defined that give the possible chains of node states that are considered safe.
- Determining safe global states from node state combinations – The overall system state is defined as the set of node states that the system exhibits at a specified time. As with a node state, not all possible combinations of node states lead to a safe system state, and the set of safe node state combinations must be defined to ensure safe operation.
- Determining safe global state changes due to node state changes – Although the safe system states can be defined from the set of node state inputs, not all system state transitions are necessarily safe. The set of node state transitions that lead to safe system states must be determined. It is important to realise that this is not just the set of independent node state transitions defined above. Some of the node state transitions may lead to safe behaviour at a node, but when combined across all nodes may lead to an unsafe system state transition.
- Consistency of global state during operation – The last problem is also the most complex. Even when we have defined a safe set of global states and transitions, based upon node states and transitions, it is necessary to understand how the consistency of the state is maintained. The consistency of such a global system state refers to the maintenance and transition of the state in a safe manner. This is not just a matter of ensuring we have defined the states and transitions. Due to the distributed nature of the nodes that make up the state, there are temporal and distribution issues to overcome. An example of this would be multiple nodes trying to update the state due to local input variations. This may lead to different parts of the system moving between node states that lead to differing global state transitions. This divergence of areas of the global state is due to delays in passing global state data around the full set of system nodes due to time delays across communication links, where an instantaneous sharing of global state cannot be achieved.

In order to allow these global states to be safely defined and for transitions between states to occur in a safe manner it requires a definition of the global state model based upon these node states and it requires a strategy for ensuring the temporal and distributed consistency can be defined and maintained.

Aims of the Research

These two problems of distributed state data and the use of large configuration files to condition generic software will be the focus of the research. Integrated Modular Systems will be used as the case study, or system of interest, for the assessment of safety techniques for data used in distributed data-driven systems. In the context of this problem statement the aims of this research are to:

- Understand the issues relating to the safety properties of data in such a distributed data-driven system.
- Define a safety process for IMS blueprints and global state data.
- Produce safety argument templates for the use of such data in a distributed system.
- Produce a model of the data in the data-driven distributed system.
- Understand how these arguments relate to the underlying technology and how the required evidence may be collected.
- Evaluate the assessment, including the argument structure and templates, using the IMS distributed system management example.

Much of the knowledge of modular systems, and IMS specifically, that is used in this research has come from the author directly as I have been involved in the field of IMS, and led the BAE Systems IMS development, for over 10 years. This work is therefore used as the primary example system due to the system knowledge already available.

Literature Review

In order to bound the scope of this research it is important to understand both the system of interest that will be the focus of the work, and the current state of research in the area of distributed data safety. There are a number of areas that need to be considered. These areas are:

- Integrated Modular Systems (IMS) – these systems will be used as the system of interest for the research and therefore it is important to understand the main characteristics.
- Safety Standards – Integrated Modular Systems are the subject of rigorous safety standards and we must identify how the applicable standards impact the data within the system.
- Data Driven Safety Critical Systems – It is necessary to understand current research in to data driven systems and their use in a safety critical applications in order to develop a methodology for the creation and use of data within such systems. This will support both the use of blueprints in IMS (a data-driven approach) and the general use of complex data in safety critical systems.
- Distributed Systems – IMS is a distributed system but there has been significant research into distributed systems in the past. The aim of this review will be to understand how the properties of distributed systems impact upon the safety of such systems.
- Safety of IMS – there has been research into the safety of IMS, although none looking directly into the problem of the safety of data. This research needs to be assessed in order to take account of current expected IMS safety processes.
- Data Consistency in Distributed Systems – Various consistency methods will need to be assessed so that a consistency approach can be chosen for use within the methodology to be developed. This will support the global state data model in IMS system management.

These areas are discussed in the following sections and possible components that may be combined to form a solution will be identified.

Integrated Modular Systems (IMS)

The term Integrated Modular Systems (IMS) is now used to describe a wide range of systems across a diverse range of applications. However, for this research, it is used to describe a specific approach to the design and implementation of avionics systems in fast-jet aircraft and rotor-craft. The term IMS has only been used quite recently, as these systems were originally referred to as Integrated Modular Avionics (IMA). During the discussions throughout this report it should be remembered that these two terms refer to the same type of system, as some references to older papers and systems will still use the older term.

Bradley et al described the problem of a conventional avionics system approach –

“A conventional avionics system is made up of many custom embedded computer subsystems. Such a system has disadvantages, which include:

- The software is bound to the hardware: the ability to re-use individual software and hardware elements is limited.

- The maintenance interval is short
- Updates are costly
- The required spares' holding is extensive and costly." [6]

Integrated Modular Avionics were developed as a way of overcoming these problems. Bradley et al went on to define the main advantages of an IMA approach –

“The advantages provided by IMA include:

- Hardware modules used on different type of aircraft
- Software not bound to specific hardware can be re-used
- Can use reconfiguration to extend maintenance interval
- System easier to upgrade as technology advances
- Reduction in range of spares holding” [6]

Integrated Modular Systems are now becoming widespread across a range of different avionics programmes. The approach has been used in both the military and commercial sectors of the industry. However, although the same basic ideas are used across both sectors there are significant differences between the underlying architectures. This research will use the military IMS approach as the primary system of interest. It is necessary to choose one of the architectures as the structure of the system will have a significant impact on the use of data within the system.

The primary European programme that defined an IMS architecture was the Allied Standards Avionics Architecture Council (ASAAC). The purpose of the programme as defined in the document -

“The purpose of the ASAAC Programme is to define and validate a set of open architecture standards, concepts & guidelines for Advanced Avionics Architectures (A3) in order to meet the three main ASAAC drivers. The standards, concepts and guidelines produced by the Programme are to be applicable to both new aircraft and update programmes from 2005. The three main goals for the ASAAC Programme are:

1. Reduced life cycle costs,
2. Improved mission performance,
3. Improved operational performance.” [7]

As can be seen from this description, the programme's goal was to define a set of standards for an avionic architecture. It did not attempt to provide a specific implementation but gave a set of interface definitions and guideline documents that would allow an implementer to create a specific instantiation that met the requirements of the standards. The standards covered a range of different architecture areas such as hardware standards, network standards, packaging standards and guidelines for a range of system level implementation issues. However, it is the software standard that contains the description of the distributed software architecture relevant to this research. The software standard [7] proposed a layered software architecture, common across a set of processor modules the would allow ease of reuse of application modules and technology transparency of the software from the underlying hardware. These primary drives were aimed at the three main ASAAC goals given above.

In order to understand the background to the software architecture we will look at a specific implementation. The specific implementation has been developed by BAE Systems Military Air Solutions for use on its military air platforms. The architecture was described at the 2008 ASSC Open Systems Symposium. The following description is taken from the presentation given at the symposium by the author of this research - [8]

When IMS was developed it was a complete architecture solution, including hardware, software and networks. The intention was to utilise the complete architecture in new-build avionics programmes. However, as the nature of the military procurement landscape changed it became obvious that there was unlikely to be significant new-build programmes in the near-term. This meant that the complete IMS architecture was put on hold and ways were sought to try to use components of the overall set of concepts for upgrades to legacy avionics systems. These upgrade programmes now became the dominant requirement as the MoD attempted to realise its new requirements by modifying existing platforms, rather than developing new aircraft and systems. Even those aircraft still in the development process had traditional federated avionics architectures due to the very long development lifecycles necessary for these systems.

Therefore the current primary need for the IMS architecture is to provide a new capability, or an enhanced existing capability, in a legacy hardware architecture. In addition, due to the extended lifetime of these systems, it is important to ensure the underlying hardware can be easily replaced to overcome obsolescence and allow increasingly capable hardware to be introduced. So, in order that the system is receptive to change it must be:

- Quick to change
- Cost effective to change
- Easy to certify

A number of underlying principles are used to ensure this:

- Modularity of System Functionality
- Adaptive Architecture (at design time)
- Modular and Layered Architecture Components
- Flexible Data Interconnect Architecture
- Modularity of Safety Argument
- Rapid Development Processes

These principles are primarily provided by the IMS software architecture. This brings the flexibility required to the original hardware platform. The main IMS components that support this flexibility are:

- A layered software infrastructure supporting the application software
- Defined interfaces to allow abstraction
- Separation of applications using enforced partitioning
- Standard communications methodology over any underlying communications protocol
- Control of system level activities such as start-up / shutdown / basic error handling

- System definition data file used to define a particular system to the generic IMS software components called the Blueprint
- Toolset to develop and manage Blueprint data files

The basic concept is still as defined by ASAAC – the use of a common software infrastructure which provides the application software with a location independent platform and handles the communication between application software modules. This can be seen in Figure 1 below, where a number of application capabilities are supported on a number of processing modules connected by a data network.

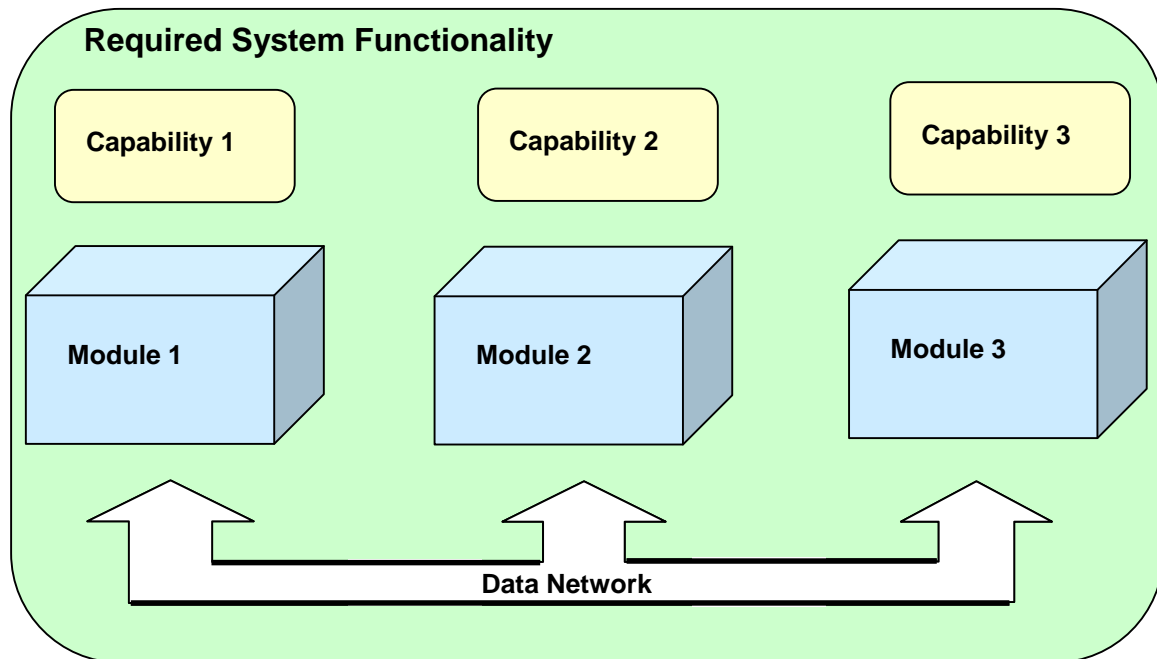


Figure 1 - Basic IMS Architecture Model

If we now consider the structure of the module that supports each application capability we can see the primary building blocks of the software architecture. The IMS software is split into two main sections that interact through two defined interfaces. The module hardware is directly supported by a Module Support Layer (MSL) that contains all hardware specific code and provides the Module to Operating System (MOS) interface to the Operating System Layer (OSL). The OSL in turn provides the Application to Operating System (APOS) interface to the application software run on the module. This is shown in Figure 2 below.

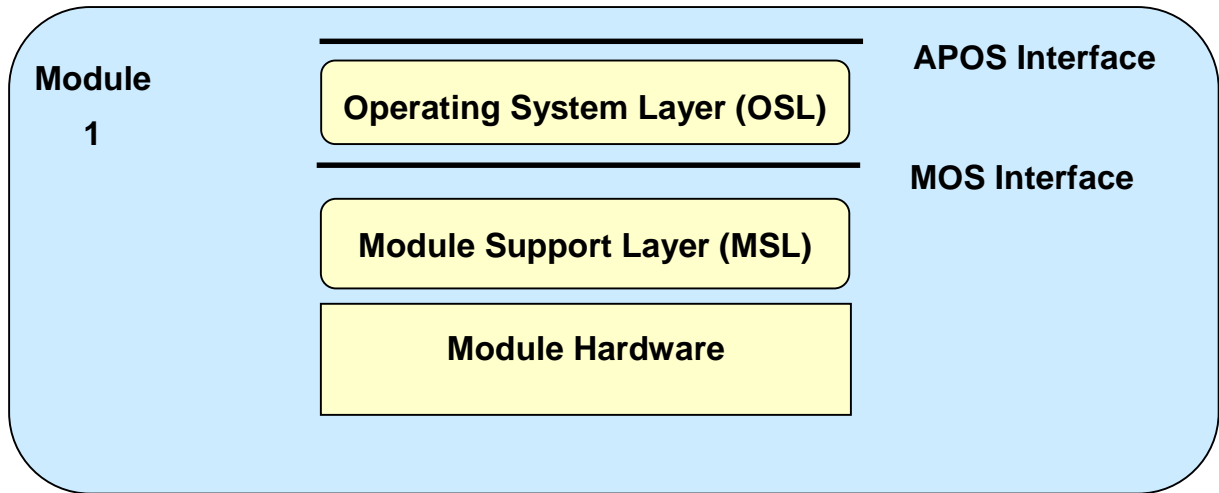


Figure 2 - IMS Software Layers

However, the OSL is not a single software module. If we now represent the different components of the OSL we have Figure 3 shown below.

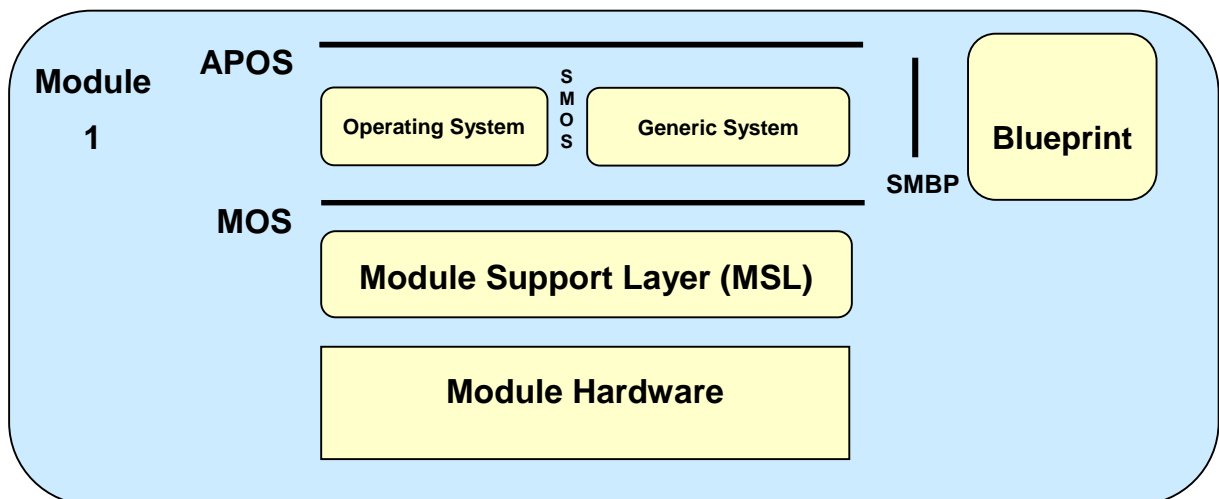


Figure 3 - IMS Software Components

The OSL can be considered as an Infrastructure layer dealing with management and connectivity. However, in this diagram we can see that the OSL actually includes three major components.

Operating System

The Operating System is responsible for control of the local module resources:

- Schedule management
- Communications management
- Protection mechanisms for application partitioning

Blueprint

The Blueprint provides the data the IMS software architecture needs to run the specific applications on this instance of the generic software infrastructure. The

software infrastructure is generic code, so each instance needs to be told what applications it needs to run, when to run them, and who they need to talk to. Each instance also has a set of possible application configurations. The OSL can reconfigure applications at run-time and switches between a known set of configurations that have been pre-determined off-line. Finally the blueprint contains a list of possible fault conditions and what to do if one of them happens. This blueprint data is obviously significant to the data safety / consistency problem.

Generic System Management

However, it is the third component, the Generic System Management (GSM), which is of most concern to the data safety / consistency problem as it raises the issue of distribution. The GSM is responsible for maintaining the correct functioning of the overall system, spanning all of the generic stacks. In order to do this it must maintain a distributed global state across all of the modules. It manages the system resources in three main areas:

- Configuration management
- Health Monitoring
- Fault Management

Module Support Layer (MSL)

Supporting the OSL is the MSL. This contains everything that is hardware dependant:

- Memory protection
- Scheduling (context switching)
- Communications
- Device Drivers for specific implementation requirements

Applications

Obviously to ensure that the overall software architecture is maintained the application software must meet the requirements of the system concept of operation. Therefore the application layer contains:

- Set of application software modules
- Each module is in it's own protected virtual memory area enforced by OSL / MSL
- Application modules can only communicate using virtual channels
- Application modules can not interfere with any other module virtual memory area

Interfaces

The interfaces cover the following areas of functionality:

- APplication to Operating System (**APOS**) interface
 - Send/receive calls
 - Fault handling calls
 - Suspend-self

- Module to Operating System (**MOS**) interface
 - Inter-module data transfer calls
 - Scheduling calls
 - Configuration calls
- System Management to Operation System (**SMOS**) interface
 - Configuration
 - Fault Management
- System Management to BluePrint (**SMBP**) interface
 - Calls to retrieve blueprint data

This architecture allows a number of changes to be made to the system:

- Changing application software modules – Due to the partitioned nature of the application modules it is possible to remove one module and replace it with another without changing the rest of the system. If the new module requires the same input and output data streams all that is required is to change the local blueprint for the hardware module that hosts the application. The same virtual memory partition can be used and the same virtual channel communication paths can be used. The schedule table for the module will need to be updated but the rest of the system will not be affected.
- Adding application software modules – similarly adding a new module can be easily performed as long as there is spare computing resource on the hardware module. In this case the virtual channel map will also need to be updated to connect the new module into the relevant communications.
- Moving application software modules – this is really the same as adding a new software module. As the IMS software architecture provides all communication links it is simply a matter of changing the blueprints on the effected modules to re-map the virtual channels and alter the schedule tables. This means the application modules can be considered location independent – they do not need to know where they sit in the system as long as their processing and communications requirements are met by the infrastructure.
- Changing hardware modules – the hardware module is abstracted from both the application software and the supporting infrastructure by the MOS interface. This means it is possible to replace the underlying hardware and MSL without modifying the majority of the software.

In summary, the main benefits of this IMS software architecture approach are:

- Hardware independence of expensive application software using the APOS and MOS interfaces for
 - Easy technology refresh
 - Overcoming hardware obsolescence
- Portable application code using a defined APOS interface giving
 - Re-use of applications both within and between product lines
- Management of system integration using partitioning, blueprints and virtual channels allowing

- Rapid system upgrade
- Control of system integration
- Maintenance of system integrity

The main issues as far as distributed data safety and consistency are concerned are in the blueprint and the GSM. In order to understand this it is necessary to discuss the operation of the GSM in more detail.

The flexibility offered by an IMS architecture needs to be managed at run-time, particularly where highly integrated systems are concerned. System management makes use of generic system management functionality in the operating system layer and system-specific mechanisms provided by the application management functionality in the application layer. System management is therefore distributed, and, in current IMS implementations, is normally hierarchical.

The objective of system management is to ensure that the correct system state is operational at all times. If functional applications encounter conditions that are beyond their scope the system manager must take action to restore a valid system state. Reconfiguration is the primary system management mechanism that is used to achieve this, based on blueprint information created at design time.

The system management functionality is split into a set of distributed components. These components are present on all processing resources in the system, but are also arranged in a hierarchical structure to provide the overall control of the system. An example of this hierarchical distribution is shown in Figure 4 below.

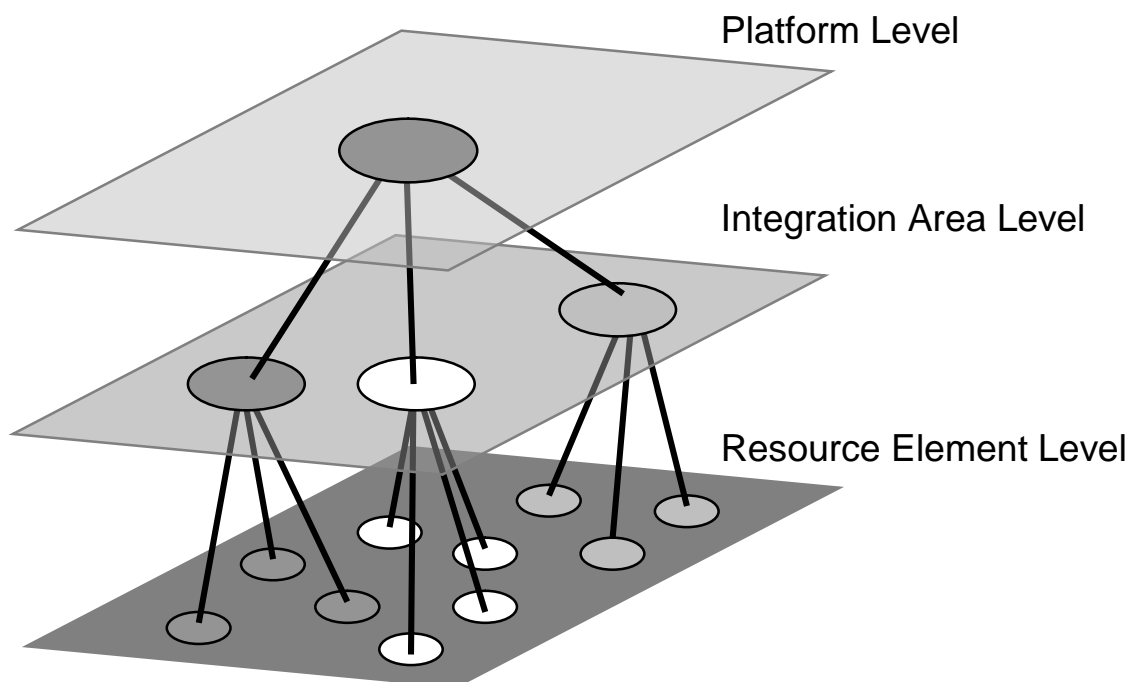


Figure 4 - IMS System Management Hierarchy

This shows system management components at three levels within the system. The lowest level is concerned with a specific resource element such as a data processor. The top level is the platform itself, responsible for all the system resources. Both the platform level and the resource element level of system management are considered to be essential. In some systems, particularly large ones, it may be advantageous to insert additional levels of system management between the platform level and the resource element level. There is no theoretical limit to the number of these

intermediate integration area levels of system management, but most platforms should not need more than three levels of system management.

- **Resource Element Level (GSM-RE)**

At the Resource Element level there is a GSM instance on each processing resource. This element is responsible for management of the local state of the processing resource, in conjunction with the local operating system, and for reporting / handling the local system state as part of the global system state.

- **Platform Level (GSM-PL)**

The platform level system manager is the focal point for the control of the system, using the hierarchy in order to control the individual system elements.

- **Integration Area Level (GSM-IA)**

The role of the integration area manager is to co-ordinate the operation of a set of resource elements and report to/respond to commands from the next higher level of system management. An integration area is a grouping of functional applications that are closely integrated, together with the hardware resources upon which these functional applications are hosted. The selection of the scope and boundaries of an individual integration area are dependent upon the functional requirements of a specific platform, and are at the discretion of the system designer. Integration areas may be hierarchical, i.e. one integration area may consist of a number of smaller integration areas.

This approach allows integration area level system management to perform reconfiguration within a single integration area. This approach to system management also allows reconfiguration at the system level, to control transitions between different mappings of the integration areas themselves. Whether reconfiguration is performed at the integration area level, or only within an integration area, is a project specific decision that depends upon the degree of assurance that is placed on the system management.

Each instance of the GSM that exists within the system to represent these hierarchy levels will be represented as a separate process and is scheduled by operating system. These GSM processes must conform to the same rules as those in the application layer. Therefore there will be a GSM-RE process on every processor hosting the operating system layer but there will also be GSM-IA and GSM-PL processes on those processors specified in the blueprint to hold the hierarchical elements.

If we now look at what each of these GSM elements contains we can see that there are separate functions in each. The ASAAC software standard specified the four functions for system management [7]. These are:

- **Configuration Management**

This function is responsible for set-up of the initial configuration of the system, and all subsequent reconfiguration management.

- **Health Monitoring**

This function is responsible for monitoring of the health status of the system, error collection, filtering of error information, and transmission of subsequent error information to fault management.

- **Fault Management**

This function is responsible for masking, filtering, and localisation of faults including the processing of corrective actions.

- **Security Management**

This function is responsible for implementation of the system security policy, including authentication, decryption, and encryption.

Each of these system management functions is present as a component of each GSM instance at all levels in the hierarchy.

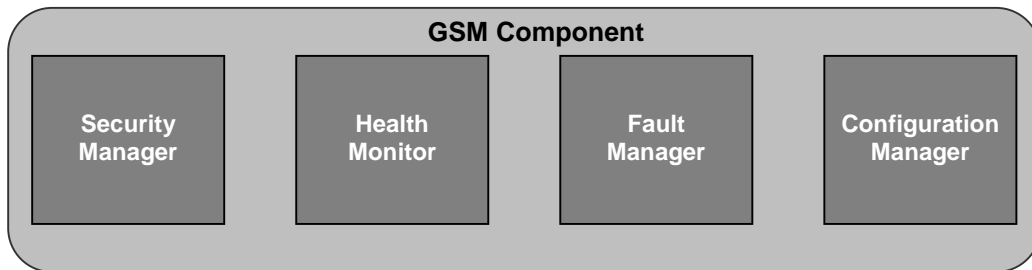


Figure 5 - IMS System Management Functions

As part of the distributed data problem under discussion the Security Manager functionality is independent of the other components and does not significantly contribute to the system of interest for developing a solution. So, for the rest of the discussions we will limit the scope of the GSM to the three functions that interact, the Health Monitor, the Fault Manager and the Configuration Manager.

There are a large number of interactions between the various functional blocks within a GSM component. In order to understand how these influence our problem we need to describe the communications paths and their purpose. The ASAAC software standard describes the interfaces between GSM components and the flow of fault and configuration data across them [7]. From these descriptions the following model of interactions can be developed.

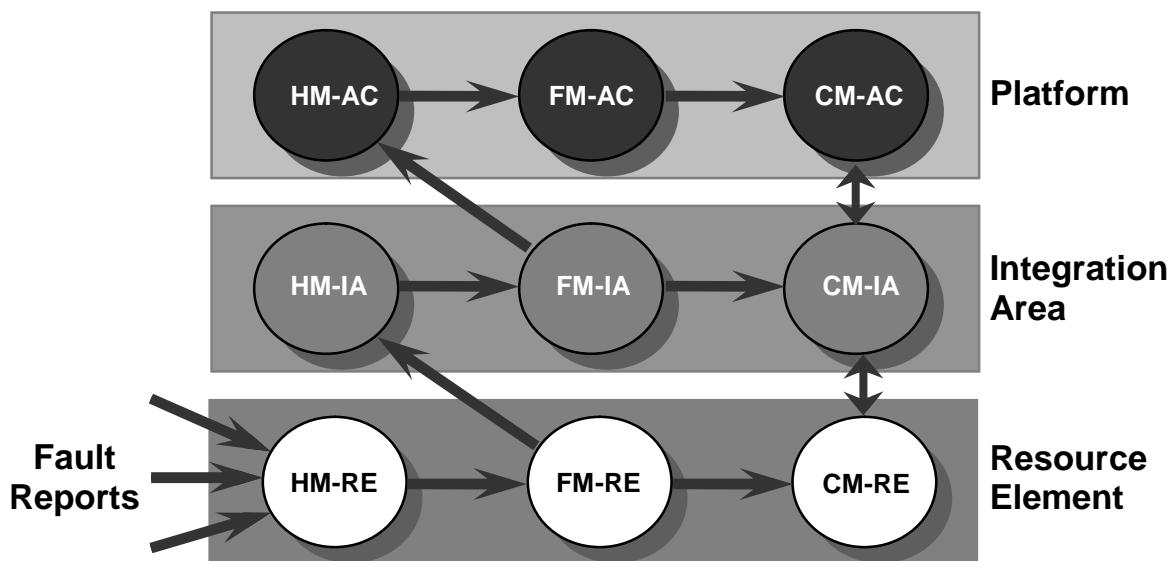


Figure 6 - IMS Management Component Interactions

At the resource element level the health monitor collects fault reports from the resource, application, operating system or MSL derived. It then filters these to overcome transitory effects and generates fault reports. These are then passed to the fault manager. The resource level fault manager maintains a health state for the resource based upon these inputs and makes decisions based upon this state. This can then lead to a request for a local state change (using the reconfiguration capability of the configuration manager), or can initiate a resource health status report that is sent to the next highest management layer in the hierarchy.

At the integration area level the components provide the same function, but they do not respond to information directly from the resource. Instead they provide this function for the collection of lower level resource element GSM instances for which they are responsible. This means the health monitor utilises the health state messages from the resource element fault managers to generate fault reports and in turn the fault manager maintains an integration area health state. Again, the fault manager can request a state (configuration) change of its configuration manager, or can initiate a resource health status report to the next highest management layer in the hierarchy (a further integration area level or the platform level manager).

Finally the platform manager health monitor utilises the health state messages from the integration area (or directly from the resource element) fault managers to generate fault reports and in turn the fault manager maintains an overall system health state.

In addition to the fault based states and reports, the configuration managers are also required to maintain the normal operation state of the system. This state can also be locally varied at each layer in the hierarchy but can also be fed down the hierarchy from the platform or integration area level. These normal operation states are changed based upon the operation of the application software and pilot inputs.

The following Figure 7 shows a typical hierarchy

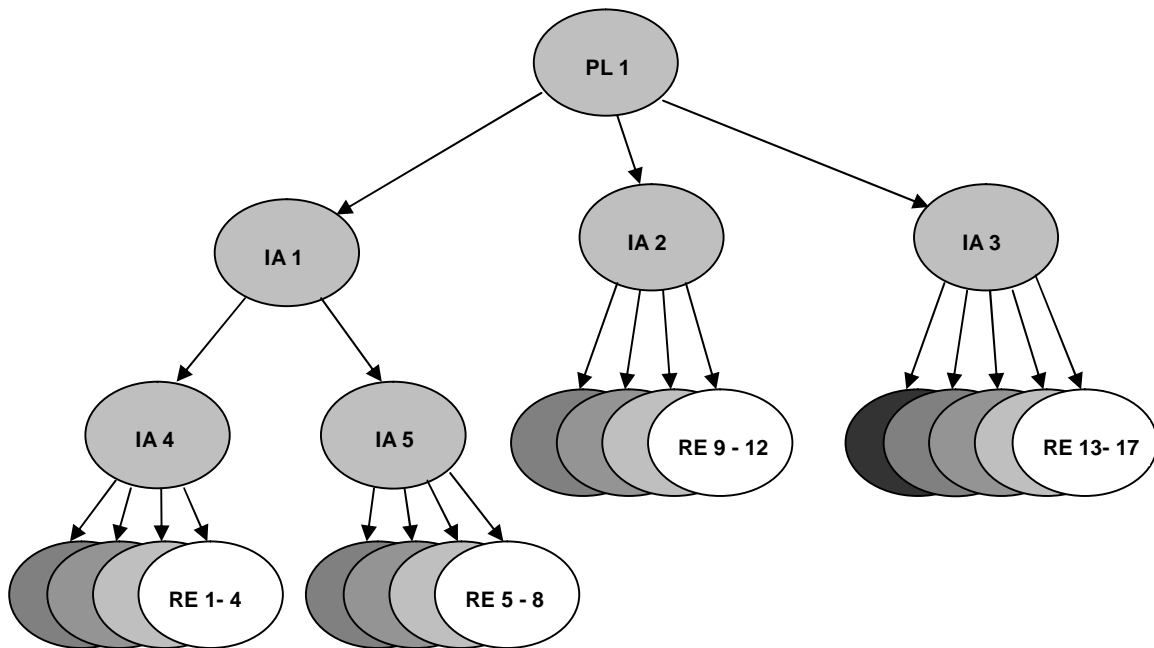


Figure 7 - Typical IMS Management Hierarchy

Each of these system nodes contains a state generated from both normal operational modes and fault conditions, and these states can be altered independently due to the

operation of the local system management decision process. The overall combination of data represented by this set of states and transitions can be seen to be a distributed global state and the elements of this global state can be updated in a distributed independent manner. Adding to the complexity is the issue of safety, as this could lead to replication of significant single point failures in the system. This may mean that the platform level GSM is actually replicated across three instances and use a voting algorithm to provide control. This means the overall platform state is held independently in each replicated instance. This IMS global health / fault / configuration state will form the basis of the system of interest for the development of our solution.

The term Blueprint refers in general to an abstract description of a real component. In the development of an IMS system, Blueprints will formally capture the behaviour characteristics and interfaces of the system and its components. These will then enable the software to be allocated to the available hardware in order to provide the required system functionality under a variety of operational conditions.

Blueprints allow the generic operating system layer to be adapted for a particular implementation of the system; standardised descriptions of resources, applications and system specific configuration data generated during system design. By utilising such a technique, a common operating system layer may be used across many platforms with any changes in operation being prescribed by the data in the blueprint.

The blueprints must encapsulate the development of a system and present the run-time operation with the correct information in a structured and efficient manner. The information required to be resident in the blueprints is as follows:

- Identification of all missions to be run on the system and any rules for their invocation, which would be used by system management.
- Allocation information to determine the actual allocation of software processes to hardware resources and the communication network. This information will take the form of allocation maps and hardware architecture (re)configuration rules.
- Fault / Error detection and response mechanisms.
- Resource Information - Information regarding the complete resources allowed and available on the system, the interconnections and status / capability of each module.
- Scheduling strategy and temporal information for all applications / processes.
- Security and Criticality information for all applications, processes and data.

The necessary information to support all configurations within the system can be classified as follows:

- The complete set of applications required to be executed on the platform for a given mission.
- Each main configuration that represents the set of applications which execute on the system at the same time, during a defined mission phase.
- Alternative subsets of each main configuration to provide safe but degraded performance, for occurrences of failures in the hardware or battle damage.

At run-time the blueprint will be required to enable management of the resources to meet the application requirements.

The blueprint is an essential element in the implementation of reusability of software and hardware within an IMS system. It is implemented as a set of large data files, one for each processing node. This large data file is used to define the initial configuration of the system, and all subsequent configuration information used during any required reconfiguration operation. However, although this information can be used at many points throughout the execution of the system the data contained within the blueprint is effectively a set of multiple configurations and should therefore remain static throughout system execution.

From this IMS description it can be seen that there are two distinct data problems that need to be assessed, the dynamic distributed state data in the system manager and the large distributed static configuration data sets. Both of these issues will be examined in this research.

Safety Standards

There are a number of safety standards that are used in the avionics industry and it is important to understand what impact these have on the problem of safety / consistency of distributed data. Of primary importance for IMS are the UK Defence Standard 00-56 Issue 4 and RTCA DO-178B, as these two standards are used to derive the certification requirements for IMS systems in the UK. There are other standards that cover safety of software and electronic systems and it is appropriate to also determine if these other documents cover the area of interest. The standards to be assessed are

- Defence Standard 00-56 Issue 4 (Def Stan 00-56)
- RTCA DO-178B
- Defence Standard 00-55 Issue 2 (Def Stan 00-55)
- IEC 61508
- RTCA DO-200A
- MIL-STD 882E
- ARP 4754

Defence Standard 00-56 Issue 4

This is the main standard imposed by the Ministry of defence (MoD) on all suppliers of defence systems. It sets out the requirements for safety management of all such projects. Def Stan 00-56 issue 4 is primarily an evidence based approach. It requires that the provider of a system sets out the reasons why the system is safe, in a structured manner such as a safety case. These reasons can be based upon the processes employed to develop the system, analysis of the system design and implementation, or on testing of the system implementation. However, in all cases the evidence must be presented to explain why this argument shows that the system is safe.

Part 1 of the standard states the requirements that must be met to demonstrate the safety of the system and Part 2 gives guidance on how these requirements can be fulfilled. Issue 4 of Def Stan 00-56 encapsulates other Defence Standards; significantly it now replaces Defence Standard 00-55 Issue 2, Requirements for Safety Related Software in Defence Systems. The guidance in Part 2 of Def Stan 00-56 attempts to include some of the original intent of Def Stan 00-55. However, as Def Stan 00-55 was directly related to software it has been assessed separately below.

If we now try to understand how data is handled within the standard we will find no specific mention of the safety aspects of data in Part 1 [9], the requirements section. It is important to understand that the definition of software used in this, and other standards, includes both code and data. This means that the standard does in general apply to data within systems, but it does not necessarily address the issue of interest in this research. So, the requirement to provide a safety argument and demonstrate evidence that the argument has been fulfilled does apply to data within the system, but there is no specific requirement to treat data in any specific manner that is different to software.

If we now consider Part 2 [10], the guidance section, we can see that it explicitly refers to the applicability of the guidance to data within the system and we must determine if this places a significant requirement on the IMS data problem. The following is an extract from Def Stan 00-56 Part 2 [10] with the significant references highlighted.

“15.3 Guidance within this section can apply to all safety-related complex electronic elements in defence systems. The guidance provided in this section has intentionally been detailed in its nature, as equivalent guidance was previously contained within Defence Standards 00-54 and 00-55.

15.4 The complex electronic elements include, but are not limited to:

- **All forms of electronically executed algorithm(s) and associated data (such as configuration data, digital maps, lookup tables).**
- Bespoke software including both embedded and computer platform type elements.
- **Databases, spreadsheets and other data”**

Therefore all aspects of data do fall within the scope of Part 2. However, this statement is a requirement to ensure whatever solution is developed for data within a system, it can be subject to the safety requirements of Def Stan 00-56 Issue 4. It does not place a specific data safety requirement on the user of the standard.

RTCA DO-178B

DO-178B is a software safety standard. It is intended to apply to software in airborne platforms and the purpose is to define how a supplier must comply with the certification requirements. According to the standard itself:

“DO-178B is primarily a process-oriented document. For each process, objectives are defined and a means of satisfying these objectives are described. A description of the software life cycle data which shows that the objectives have been satisfied is provided. The objectives for each process are summarized in tables and the effect of software level on the applicability and independence of these objectives is specified in the tables. The variation by software level in configuration management rigor for the software life cycle data is also in the tables.” [11]

DO-178B is therefore a process based standard based around the software lifecycle. This means the assessment of software relies on the process used to develop it and on testing the resultant system. It has gained significant support in the commercial aerospace sector, including the commercial aerospace version of IMS, and it is also the primary standard in use for U.S. aerospace programmes.

The main aspects of the standard are:

- Reviews and Analyses of the Software Architecture and Source Code
- Requirements-Based Testing Methods
- Rigorous Software Design Standards

In terms of specific advice or requirements for the safety assessment of data, there are a number of areas where data is discussed.

Section 5.5.2 Software Design Process Activities discusses the concept of the Design Description. The standard gives a definition in a later section:

“11.10 Design Description

The Design Description is a definition of the software architecture and the low-level requirements that will satisfy the software high-level requirements. This data should include:

- a. A detailed description of how the software satisfies the specified software high-level requirements, including algorithms, data structures, and how software requirements are allocated to processors and tasks.
- b. The description of the software architecture defining the software structure to implement the requirements.
- c. The input/output description, for example, a data dictionary, both internally and externally throughout the software architecture.
- d. The data flow and control flow of the design.” [11]

This Design Description includes both the software architecture and the low-level requirements. The standard states:

“Control flow and data flow should be monitored when safety-related requirements dictate, for example, watchdog timers, reasonableness-checks and cross-channel comparisons.” [11]

This statement does not cover distributed data specifically, but it does cover the use of complex data structures as can be seen from the Design Description. However, the requirements are related to a comprehensive design process, including a detailed design, a data dictionary, data and control flow analysis. This is a typical software design requirement and therefore no more stringent than Def Stan 00-56.

Defence Standard 00-55 Issue 2

Def Stan 00-55 is now redundant, although the main drivers behind the standard now form part of Def Stan 00-56. However, it is still useful to understand the standard as it is still acceptable to use Def Stan 00-55 as part of response to a Def Stan 00-56 issue 4 requirement.

Again the definition of software used in this standard includes both code and data. This means that the standard does in general apply to data within systems, but it does not necessarily address the issue of interest in this research. Where references to data are made they relate to the use of data within the system, not to specific requirements for extra safety analysis of complex distributed data structures.

When discussing the role of software specification and design the standard states:

“32.2.2 The Software Specification and Software Design shall conform to the following:

g) If safety functions or safety properties rely on the characteristics or properties of data which form part of the SRS, the characteristics and properties of the data shall be formally specified and designed.” [12]

It also states:

“36.5 Static analysis and formal verification

36.5.4 If any data used by the SRS are required to have particular characteristics or properties, and the correct behaviour of the SRS relies on these characteristics or properties, the actual data to be used by the SRS shall be shown to have these characteristics or properties.” [12]

As seen in the other standards above, these statements mandate that the specification and design phases of the software development lifecycle also include data used within the overall design and that any specific requirement on the data is shown to hold in the same manner as a software requirement would be handled. However, it does not specify additional requirements for data safety of complex distributed data structures.

Def Stan 00-55 Issue 2 Part 2 gives guidance on how to apply the requirements in Part 1. This guidance does cover data in more detail.

On the subject of static analysis the guidance states:

“26.2.4.1 Static analysis is the process of evaluating software without executing the software. For the purposes of this Standard, static analysis is considered to comprise a range of analyses which includes all of the following:

d) Data use analysis: analysis of the sequence in which variables are read from and written to, in order to detect any anomalous usage.” [13]

This does suggest that an analysis is required of the sequencing of data structures. This is relatively straightforward for most data structures, but for a complex distributed structure, such as IMS management contains, this is a more difficult problem. Due to the interaction of multiple individual elements combining to create the structure the number of possible sequences of events is significant, and therefore requires a more complex solution. This safety requirement will need to be addressed when considering the distributed system state safety issue.

The guidance section also goes further with respect to the software specification and design, stating:

“32.2 Production of the Software Specification, Software Design and code

Where the software relies on input of a particular form, such as configuration data, look-up tables etc, the characteristics and properties of this data should be formally specified and designed. This might include specifying type, range, accuracy, permissible values, relationships between two or more data items, permissible operations etc. It may not be possible to generate proofs of the safety properties without such specification.” [13]

This explicitly covers problems with data, including the relationship between data items. This suggests that a specification of the data interactions will be required in order to certify a data-driven system component.

Section 34.2 again takes this further:

“34.2 Verification of the Software Specification

34.2.1 The general method of constructing proof obligations and discharging them using formal arguments should be used to verify the Software Specification. Since the Software Requirement is informal, the proof obligations will be those required for internal consistency of the Software Specification. These should include proof that constraints demanded by the Software Requirement are satisfied by the Software Specification.

34.2.2 All safety functions and all safety properties should be formally specified. Particular care should be taken to ensure that, for example, all timing and capacity constraints have been identified, and that constraints and relationships between data items are recorded.” [13]

This states that formal specification is required for relationships between data items that have a direct influence on a safety function. The IMS system state could be said to be in this category. The question of formal proof is an ongoing debate within the safety community, but it may be necessary to assess this requirement for the highest integrity level systems that use IMS.

IEC 61508

“IEC 61508 applies to electrical/electronic/programmable electronic safety-related systems. This encompasses avionic systems and therefore the standard forms a useful part of the safety domain for IMS. IEC 61508 is based around the concept of functional safety. The standard defines both safety and functional safety:

“We begin with a definition of *safety*. This is freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment.

Functional safety is part of the overall safety that depends on a system or equipment operating correctly in response to its inputs.” [14]

Functional safety therefore encompasses not just the system, but also the system during operation as it also includes correct responses to system inputs. This is a useful distinction, but other safety standards, although not as explicit, also include testing and analysis of the system and its inputs.

IEC 61508 has the following characteristics:

- It uses a risk based approach to determine the safety integrity requirements
- It requires an overall safety lifecycle model
- It covers all safety lifecycle activities from initial concept, through to disposal
- It covers all normal system safety functions and failure
- It covers requirements for preventing failures and for controlling failures
- It specifies the techniques and measures necessary to achieve the required safety integrity [14]

Although the standard is comprehensive when discussing the role of software in systems, it does not make a distinction for data structures and therefore offers no further guidance than the previous standards.

RTCA DO-200A

The scope of DO-200A is aeronautical data used for navigation, flight planning terrain awareness, flight simulators and other applications [15]. As such, it is the only safety standard specifically aimed at data. However, it is specifically aimed at the use of large data sets in applications; it is not designed to cover the real-time distributed data model in our IMS system of interest. The standard covers the process for developing and using data. It refers to a “data chain” that links the supplier chain through to the end-user of a data set. It then requires that both the Interfaces along the chain and the Processes used within the chain are specified as per the standard [16]. Although most of the focus of the standard is on the development process, the interface requirements include a specification of the required data quality of the data set, covering:

- Accuracy
- Resolution
- Assurance Level
- Format
- Timeliness
- Completeness
- Traceability

This set of quality attributes can usefully be applied to the distributed data problem, and can form the basis of a data model for distributed data that will be developed as part of the solution. This set of properties will be used to propose our system data model.

MIL-STD 882E

882E sets out a standard practice for system safety. The standard is based around a process for assessing risk and maintaining the process to record that the risks have been mitigated. The standard applies across all of the system development activities, including software, but does not make any specific statements about data. However, the standard does include a software hazard risk assessment process. This sets out the level of assessment and testing required due to level of risk associated with a software function. It defines a number of software control categories (i.e. how much control a software function exerts over a system safety function) and then maps these, together with a categorisation of the possible outcome of failure, using a Software Mishap Criticality Matrix to define the requirement for system analysis and testing for the function.

The example given in the standard:

- a. Software Control Categories.
 - I Software exercises autonomous control over potentially hazardous hardware systems, subsystems or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazard's occurrence.
 - Ila Software exercises control over potentially hazardous hardware systems, subsystems, or components allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.

- IIb Software item displays information requiring immediate operator action to mitigate a hazard. Software failures will allow or fail to prevent the hazard's occurrence.
- IIIa Software item issues commands over potentially hazardous hardware systems, subsystems or components requiring human action to complete the control function. There are several, redundant, independent safety measures for each hazardous event.
- IIIb Software generates information of a safety critical nature used to make safety critical decisions. There are several, redundant, independent safety measures for each hazardous event.
- IV Software does not control safety critical hardware systems, subsystems or components and does not provide safety critical information.

EXAMPLE SOFTWARE MISHAP CRITICALITY MATRIX.

HAZARD CATEGORY CONTROL CATEGORY	CATASTROPHIC	CRITICAL	MARGINAL	NEGLIGIBLE
I	1	1	3	5
II	1	2	4	5
III	2	3	5	5
IV	3	4	5	5

Mishap Risk Index Suggested Criteria

- 1 High risk - significant analysis and testing resources
- 2 Medium risk - requirements and design analysis and in-depth testing required
- 3-4 Moderate risk - high level analysis and testing acceptable with Managing Activity approval
- 5 Low Risk – Acceptable” [17]

This method of deciding the risk factor suggests that IMS system management could fall into the high risk category, depending upon the required safety level of application software run on the IMS architecture.

ARP 4754

ARP4754 is another process based standard. The standard’s main focus is the process used to ensure certification, the evidence required to demonstrate that the certification has been performed appropriately, and that the correct information is retained to prove the adequacy of the certification. This standard offers very little specific information on software aspects of the system, and does not mention data explicitly.

Summary of Safety Standards

From these various safety standards we can define a set of issues that must be addressed to ensure the safety of the data in an IMS system:

- Are the requirements for the data complete and unambiguous?
- Has the appropriate process been used to create the data?
- Have the hazards associated with the data been identified?
- Have all the risks associated with the data been identified?
- Have all hazards been mitigated to an acceptable level?
- Is the flow of the data through the system understood and defined?
- Does a specification exist for all safety related data structures?
- Is the implementation of each data structure traceable to the specification and requirements?
- Is there a safety argument for the data in the system?
- Has all evidence been supplied to satisfy the safety argument?

The IMS data safety solution must ensure these issues are addressed. We must also take account of the Defence Standard 00-55 Issue 2 requirement to perform data use analysis. This must be understood when proposing a solution for the distributed system state problem.

This list of issues will form the basis of the proposed data safety lifecycle described later.

Data Driven Safety Critical Systems

The problem of data safety and consistency in the type of distributed systems of interest has not been the subject of a large amount of research. There has been work in the area of safety in data intensive systems, notably by Professor Neil Storey at Warwick University and Alastair Faulkner, but the topic could be said to be in its early stages. However, it is important to ensure that any relevant work is taken into account when trying to understand the scope of the problem to be assessed during this research.

Firstly we will assess the reasons why data is important in data-driven systems, and why the safety assessment of this data can not be performed using the same techniques used for the software in a system.

Role of Data

Storey and Faulkner describe the role of data within a system [18]. They make it clear that all software uses and requires data but that in most cases this is limited to initial constants and calculations during execution. However, some software makes use of a more significant data set, and Storey and Faulkner give examples of both large initial configuration files, such as plant configuration data, and databases used during flight, such as terrain databases. The distinction made for these forms of data is that they are generated outside the development lifecycle of the software that uses them, and that this means they are not part of the verification process for the software. This leads them to suggest that these forms of data should be subject to their own separate development lifecycle, including their own verification process.

Storey and Faulkner also propose that data is subject to the same form of faults as software [18]. They suggest that just as software is subject to systematic faults and faults that appear to be random (but in reality are systematic in nature), so data can also be subject to these forms of faults. This leads to both the suggestion that data should be subjected to standard software safety practice of hazard and risk analysis, and the recognition that this analysis for data is not usual. Storey and Faulkner identify three outcomes:

- “Data is often not subjected to any systematic hazard or risk analysis.
- Data is often poorly structured, making errors more likely to be produced and harder to detect.
- Data is often not subjected to any form of verification.” [18]

In order to overcome these data faults Storey and Faulkner make a set of recommendations:

- “The architectural design of a safety-related system should clearly identify the data elements within the system, distinct from the hardware and software components.
- The data within the system should have its own development lifecycle, with appropriate stages of hazard analysis, design and verification.
- The data should be described by an appropriate data model that is self-sufficient, clear, analysable and unambiguous.
- The data model, and its populated data set, should be developed, verified, documented and maintained, using techniques appropriate to the Safety Integrity Level (SIL) of the system concerned.
- The tools and techniques used in the creation of data for safety-related systems should be appropriate to the SIL of the system concerned. This might require tool manufacturers to develop new tools, specifically for this purpose.
- Data that is derived from external sources, or that is developed separately from the safety related system itself, should be subject to the same requirements of verification and documentation, as data produced as an integral part of the project.” [18]

Some of these recommendations have already formed part of the modular architecture approach of IMS. Separation of the blueprint design and development from the software is already commonplace. The tools and techniques have been developed with the appropriate safety level in mind. However, definition of an appropriate process and lifecycle to manage the data in the system is not clearly understood and will be the focus of our proposed solution later.

One of the problems with this lack of safety analysis and the lack of focus on the data safety problem is identified by Storey and Faulkner as the assumption that the data used by any software module is part of the software, when in reality this is not this case. This assumption leads to the required analysis of the data being neglected, and not subject to the same degree of care. The suggestion is that this is exacerbated by most safety standards, which provide a large amount of guidance and requirements for analysis of software, but do not provide the same explicitly for data. [19]

In order to ensure that data is subject to the required level of safety analysis Storey and Faulkner suggest that it should be subject to its own lifecycle for development [19]. They argue that the standard software development ‘V’ lifecycle used within IEC 61508 is a good starting point. This is in fact the same lifecycle used to develop the code, but it needs to remain separate as the data is not developed at the same time as the software.

Faulkner and Storey justify this separate data safety lifecycle from his definition of a data-driven system (given in the introduction). As the data in a data-driven system is not only developed separately from the software but is used to define the actual functionality of generic code, they argue that a data-driven system should have two safety arguments, one for the software and one for the data. [5] This separation of safety arguments will lead to a proposed argument module for the data in our solution.

Finally, Faulkner and Storey identify one last aspect of this separation of data and software development. As the data is generated after the system has been developed the “design-intent” of the data component may not be understood. [20] This can lead to use of the system with this new data in an unsafe manner. Separate safety assessment of the data can be used to overcome this problem.

There is one safety standard that discusses the problems of data: RTCA DO-200A. This standard is applicable to large data sets used in aircraft, such as terrain data bases, and is mainly process based. However, Faulkner and Storey [5] highlight the importance of the data quality criteria set out in DO-200A: [16]

- The *accuracy* of the data;
- The *resolution* of the data;
- The confidence that the data is not corrupted while stored or in transit (*assurance level*);
- The ability to determine the origin of the data (*traceability*);
- The level of confidence that the data is applicable to the period of (its) intended use (*timeliness*);
- All of the data needed to support the function is provided (*completeness*); and
- The *format* of the data meets the users requirements.

These data quality criteria will be used later in definition of a data model for the distributed data problem.

Building upon Storey and Faulkner’s work Knight et al discuss the issue of data specification [21]. Recognising the need for a data lifecycle Knight et al suggest an approach for data specification that allows the data to be instantiated correctly and validated. This approach includes three elements:

- “*Data Syntax*: formal specification of the structure of the data.
- *Data Context*: rigorous specification of the real-world entities that the data encode and the relationship between the real-world entities and their encoding.
- *Data Semantics*: formal specification of the rules that limit data specifications to those which are acceptable to the application.” [21]

Knight et al state that whilst a data syntax can be large and time-consuming to complete, the basic structure is simple, consisting of defined fields for each data

element. This structure can then be used to provide a complete and unambiguous definition of the data model. This then allows all users of the model have the correct understanding of the structure and validation of the data to be performed. Although the significant size of the structure may still make the validation process long and complex, it can be approached in a systematic manner due to the rigid syntax.

Knight et al also state that a defined syntax alone is not enough to ensure that all users of the data model understand how to use / manipulate the data in the intended correct manner. In order for this to be achieved it is also necessary to define and understand the context of the data, i.e. a more complete description of the intent of the data element than can be provided by a pure structural decomposition.

Finally, Knight et al state that a data model must include semantics as well as syntax. The semantics of the data elements, such as valid range limitations, allowed interactions between data elements, and contextual limitations between elements, are also required to ensure that the data is used in the correct manner. A user can meet the syntax of each element of a data model but still not use the overall data model correctly without understanding the semantics of the model.

This approach to specification will be considered as part of the IMS solution as the use of syntax, context and semantics will allow the specification of the data requirements for the IMS structures. These syntax and semantics based approaches will be reflected in the data model produces for our solution.

Data Lifecycle

Faulkner and Storey also describe the data supply chain defined in DO-200A and uses this concept to develop a set of main elements required to ensure the supply chain provides data of the correct integrity level are set out. These elements are:

1. "Identify data origins
2. Identify boundaries (organisational, legal, political):
3. Identify process and adaptation phases
4. Apportion the integrity requirements
5. Identify evidence requirements
6. Specify corrective action process" [20]

This set of elements is used to describe the design of a supply chain but it is not well suited to the definition of a set of requirements for the development of blueprints in the IMS design process. This is due to the specific nature of blueprint development, which is performed as an integrators task. The data chain set out above is related to the production of data through a chain of different suppliers. For the IMS blueprint this chain does not exist. It is the final integration team that uses design information to produce the blueprint files for the integrated system.

Part of this data supply chain problem concerns the origin of the data. Faulkner and Storey suggest that the origin of the data can influence the possible integrity of the data. [20] As systems grow in size and become more complex the process used to create the data becomes more important. It is suggested that for small scale systems manual data entry with some form of tool support may be able to deliver the required integrity. However, as the size of the data structures increase a more complex toll-driven approach will be required in order to ensure that an adequate integrity level will be maintained. Indeed, the potential for errors to be introduced using manual data entry without toolset enforcement of safety and integrity rules is directly related

to the size of the data structure. Faulkner and Storey state that some form of abstraction, both horizontal across the structure and vertically using hierarchies, may be required in order to control these integrity requirements [20]. This is analogous to the IMS blueprint design tool, where automation is used to ensure the structure of the blueprint output, and a set of IMS design rules, are enforced by the blueprint to remove some of these manual entry problems. The ideal in this situation would be to automatically generate blueprint files from the design data stored in the design tool, thus removing the entire data entry problem. However, this is a non-trivial task, and would need to be repeated for each combination of design and blueprint tool.

Faulkner et al also identified the need to treat different forms of data within the system in a different manner. [22] Configuration data represents the initial static model of a system and should not be capable of modification at run-time. However, dynamic data derived from system operation will alter at run-time. Faulkner et al suggested that these two forms of data must be kept independent to ensure the initial static model is not corrupted. If we relate this to IMS we can see that the initial configuration is the blueprint, and the dynamic data covers the distributed system state. It is important therefore to ensure that the dynamic system state structure, whilst being initially derived from the blueprint, should be maintained as a separate data structure.

The distributed system state is of interest in our research problem, and Storey and Faulkner discussed some of the particular problems associated with dynamic data [23]. Whereas a static initial data structure can be subject to off-line validation, dynamic data structures are subject to constant change and are therefore can not be validated in the same manner. Storey and Faulkner suggest that it may be possible to apply some level of real-time validation to dynamic data, but for any significantly complex data structure this will be extremely difficult.

As an alternative approach to run-time validation Storey also suggests that a possible alternative approach would be to pre-validate all possible data combinations prior to system operation to ensure all possible state combinations are safe. [24] Storey does point out that this may be infeasible for any complex systems. He suggests an alternative would be to encapsulate the data in a separate module and then try to define safety requirements for this module that can be proved off-line. However, the methods to encapsulate a distributed data structure are not obvious, and we are also faced with the problem of definition of safety requirements.

For our IMS system state structure it is unlikely that the real-time nature will allow comprehensive validation of the whole structure as a single operation. Therefore one of the key aspects of the research to be performed will be to define a realistic validation scheme and the definition of safety requirements for the dynamic system state data. This will form part of both the process definition and the safety argument structure.

Storey also discusses the relative immaturity of current approaches to validation of data driven systems [24]. He suggests that current software approaches to ease validation, such as isolation and partitioning, could also be applied to data, but this is not a common practice. Even the most fundamental concept for validation, a complete set of requirements, does not always exist for the data used within a system. Storey argues that this is a basic pre-requisite for data driven safety related systems [24].

Storey and Faulkner also suggest that re-validation due to changes to the data in the system can be handled using similar techniques to software modifications by the use of the correct architecture employing modularisation and partitioning [25]. A data

model is required that defines a data architecture for all data within a system. This approach allows the data in the system to be structured into modules with defined interfaces, where data that is subject to high change can be isolated into specific modules in the model. Initial definition of the data is therefore simpler as each module can be subject to its own requirements, with the entire model being integrated as the system is developed. Changes can be handled in a modular manner, where single modules of the data model can be altered and any impact on other modules being constrained by the defined interfaces. This approach works well for initial static data, but a similar approach could be used to make validation of a dynamic data structure simpler. This will be investigated as part of our problem definition. However, it must be noted that the use of such a data model is dependent upon the nature of the data used by the system. Storey and Faulkner note that a system employing a large monolithic data structure is not suitable for this modularisation approach [25]. Therefore the structure of the data must be considered from the initial definition of the system architecture.

Storey and Faulkner also note a statement from RTCA DO-200A concerning the responsibility for data integrity [25]. The ability to achieve the correct data integrity depends upon the origin of the data. Where a system is built of modules that pass data it is not always possible to validate the integrity of the data at interfaces between modules due to the difficulty of defining requirements at this level. Therefore the responsibility for data integrity flows back through the chain of modules and often it is the originator of the data that must provide the validation requirements that can be placed upon the data. If we consider IMS data, this means the blueprint validation must occur as the data is defined in the blueprint file – the validation can not be performed by the IMS software using the data at run-time. This suggests some form of off-line validation is required. The dynamic system state data is also subject to the same problem but this can not be performed pre-run-time as the state is built as the system runs. This requires the validation of the system state to be performed at run-time by the module creating the state. Again, this must be taken into account in our final solution.

Overall Storey and Faulkner's work gives a useful introduction to some of the issues with data driven systems and their use in safety related applications. However, Storey and Faulkner are only addressing the use of large data sets to control system behaviour. An aircraft navigation system is an example of such a traditional high-integrity system that requires a large amount of data to operate, in the form of a large terrain data base. This terrain data base provides a digital representation of the ground ahead of the aircraft, and can be used to provide cues to the pilot for terrain following and avoidance. It is obvious that the data within the system has an impact on the safety of the aircraft, and as such must be assessed as part of the overall safety argument for the system. Storey and Faulkner's suggested techniques can be used to manage the integrity requirements of such a system and a more traditional safety approach can be utilised in order to do this. The functionality of the software itself is not modified by the data it uses it to make decisions. Therefore the safety of the software can be assessed on the basis of the data ranges that are used to perform the computations within the system. Also, there are a defined, and manageable, number of computations that depend upon the data, and the manner in which they interact is statically defined in the structure of the code. Finally, the data within the terrain database is limited by the structure of the database, and is not dynamic at run-time. Once a digital map of a landscape is created it only changes due to a known update process outside of the operation of the system in flight. These

constraints on the data mean that the data itself can be statically analysed before use to ensure its validity and consistency.

Storey and Faulkner do not consider, or make any provision for, a distributed computing model and the added complications that this brings to our problem. IMS contains large data structures of the form Storey and Faulkner define, the blueprints are an example of this, but it also requires distributed data sets that also control system behaviour, such as the distributed system state. The concepts that Storey and Faulkner develop, such as the use of data lifecycles, data requirements, validation techniques, data models, and data chains, give a sound basis for a solution to the IMS problem, but require validation, modification or extension to cover the distributed and dynamic nature of IMS data.

Distributed Systems

Due to the distributed nature of the IMS data problem we must also assess what impact distributed systems research has on possible solutions.

Distributed System Characteristics

Emmerich sets out a set of characteristics that occur due to the nature of distributed systems [3]:

- “Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple Points of control
- Multiple Points of failure” [3]

He also sets out a set of desirable characteristics that we can use to assess distributed systems [3]:

- **Resource Sharing** – the ability to share or use data and software anywhere within the system.
- **Openness** – the ability to easily add to or improve a distributed system, where the interfaces to components need to be defined and published.
- **Concurrency** – the ability to run multiple software modules simultaneously in a controlled manner preserving the intent of the design.
- **Scalability** – the ability to expand the system to allow more processing power to be added without changing the function of the original system.
- **Fault Tolerance** – the ability to maintain the required functionality in the presence of faults.
- **Transparency** – the ability of the system to appear as a single computing resource, rather than a collection of separate computing nodes. Emmerich lists a set of transparency properties that combine to achieve overall transparency [3]:

In assessing the IMS data problem it is important to understand how well the IMS model provides these “ideal” distributed system properties. However, these properties did form the basis of the IMS initial concept, so in terms of the Emmerich

definition IMS should be an ideal distributed system model. In fact, IMS meets all of these ideal distributed properties and can therefore be considered as an ideal distributed system for the purpose of this research.

Van Streen states that there are many pitfalls in the design of distributed systems and sets out a series of false assumptions that can lead to problems with system design and analysis [2]:

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator” [2]

In assessing the IMS data problem it will be important to ensure that these false assumptions have not been made in the IMS software model, and we must ensure they are taken into account in the solution for the IMS distributed state data solution.

Transactions

One essential component of a distributed system is the concept of transactions. A distributed system is made up of a set of computing modules and their associated software components. However, due to the distributed nature of the system the overall system functionality is provided by the set of interacting modules. This requires information to be passed between modules. These modules therefore have individual states, but these states combine to form the overall system state. Due to this complex set of interacting states a model is required for state modifications due to these interactions. Van Streen defines a transaction model [2] that describes how the state of an object can be updated:

“A transaction is a collection of operations on the state of an object (database, object composition, etc.) that satisfies the following properties (**ACID**):

- **Atomicity:** All operations either succeed, or all of them fail. When the transaction fails, the state of the object will remain unaffected by the transaction.
- **Consistency:** A transaction establishes a valid state transition. This does not exclude the possibility of invalid, intermediate states during the transaction’s execution.
- **Isolation:** Concurrent transactions do not interfere with each other. It appears to each transaction T that other transactions occur either before T , or after T , but never both.
- **Durability:** After the execution of a transaction, its effects are made permanent: changes to the state survive failures.” [2]

This set of properties is important to ensure that each object’s (or software module’s) state remains consistent, and by the combinations of multiple object states and correct transactions the overall system state can be maintained. The IMS distributed system state solution must take account of this ACID transaction model and this

subject will be investigate further during the review of consistency models in distributed systems

Safety of IMS

Although not dealing directly with the problem of data within IMS there are a number of research papers on the subject of IMS safety in general. An understanding of this research is necessary to define the context within which our data safety solution must operate.

Hollow et al describe an approach to the certification of IMA systems that includes reconfiguration; a key element that drives the need for complex blueprints and distributed system states[26]. The main issue with a reconfigurable IMA system is the number of possible configurations that may be supported in one blueprint. A traditional certification approach would assess a system for its particular configuration of software and hardware. The assumption is made that the system to be analysed can be seen as a single entity and that the system is deterministic (or at least predictable).

In a reconfigurable IMA system this can not be done in a single operation as the mapping between software and hardware depends upon configuration mappings in the blueprints. This large number of possible configurations would require each mapping to be certified in isolation if a traditional approach were used. Hollow et al suggest that this would lead to an unjustifiable cost to certify a complex IMA system.

The suggested solution is to try to define sets of “equivalent” configurations that can then be analysed as a group to reduce the number of possible separate certification steps. This would lead to the definition of a set of safe configuration changes. The approach requires the use of a baseline model of the IMA system to be certified. This baseline model is analysed to ensure its suitability against the requirements. A set of new configurations can then be defined to take account of failures and possible mode changes and these can then be assessed against the requirements. If any of the new configurations does not meet the requirements it may be necessary to iterate the process from a new baseline. By limiting the system changes to a set of single likely failures and mode changes the number of new configurations is limited. The “equivalence” of configurations is based upon a “fitness” algorithm using weighted parameters, such as resource usage and timing properties. This is an abstract view of the equivalence of two configurations and is used to determine which new configurations will produce the most equivalence for any needed change. This can then be used to reduce the certification effort required.

This is a novel approach to generating configurations that will reduce the certification burden, but it does not deal with the problem of data or distributed computation. Due to the need to assess overall configurations the method is necessarily abstracted away from implementation detail. A further solution is still required to show the safety of the data within IMA systems, although this may well be complimentary to the Hollow approach.

Nicholson et al describe a baseline safety argument for IMS that covers a set of main components of the IMS model [27]. This Goal Structured Notation (GSN) argument structure gives a top-level breakdown for arguing the safety of:

- Memory partitioning
- Communications
- Time partitioning and scheduling

- Initialisation
- Configuration

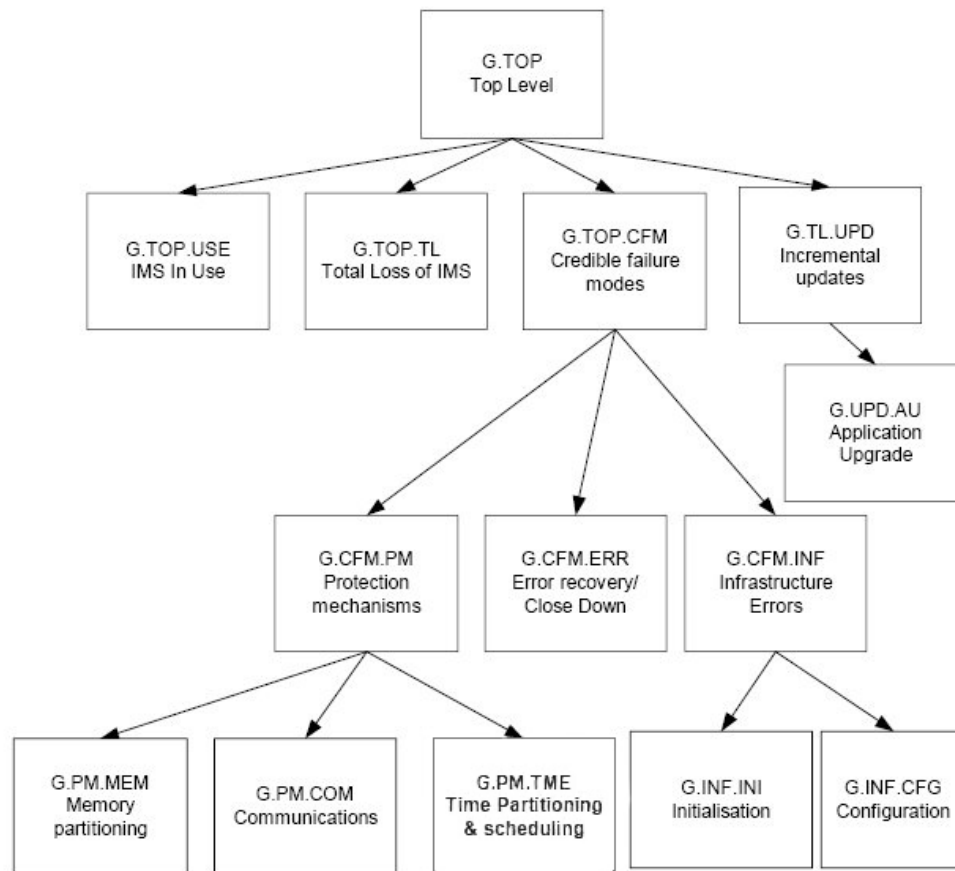


Figure 8 - Nicholson's GSN Model for an IMS Safety Argument

However, they do not break down all of these elements into their own GSN argument. They argue that a structure safety argument is required in order to partition the certification problem and to allow the use of a modular safety argument approach. This modular certification approach allows incremental updates of parts of the system, related safety argument and evidence, whilst maintaining the rest of the system argument and evidence. This requires careful partitioning of not only the safety argument structure, but also the software itself.

The safety argument structure developed, whilst providing a top-level structure, does not cover the problem of data safety, either the use of blueprints or the distributed dynamic data structures used within the IMS GSM. The model is only one of a number of IMS GSN safety arguments and therefore this research will need to assess each model and understand how to combine and extend these GSN models to cover the IMS data problem.

Joliffe also defines a safety case structure for IMS, this time relating to the blueprints used for IMS [28]. This argument structure is based upon the idea of static initial blueprints and run-time blueprints. The argument relies on the safe instantiation of the static blueprint (including the safety of each configuration definition), the maintained safety of the run-time blueprint, and a safe mechanism to move between the safe configurations. These three elements are not developed further as GSN arguments.

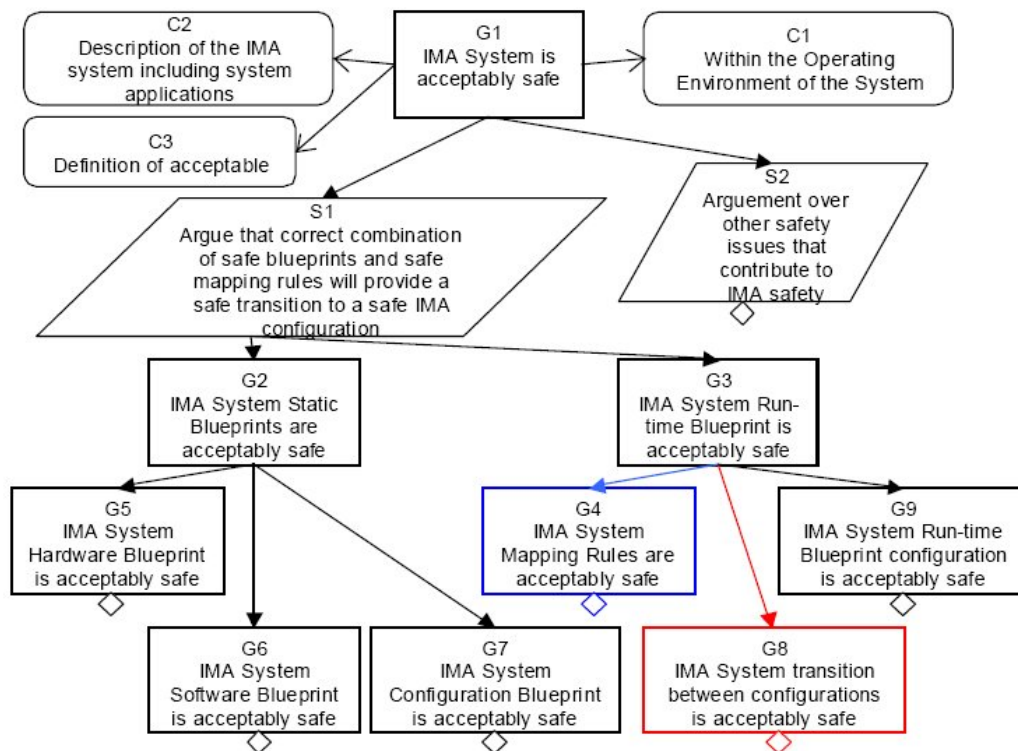


Figure 9 - Joliffe's GSN Model for a Blueprint Safety Argument

Joliffe also gives guidance on how some of these required safety elements may be assessed, such as the use of HAZOP techniques to assess each of the blueprint types and the use of a safety margin approach to ensure the safety of each component can be maintained. However, the paper covers many of the issues of the certification of blueprints but does not provide developed solutions. Although the overall GSN structure is useful as an input to the IMS data problem, the research does not consider directly the problem of data safety, and will need to be combined and extended with other GSN components.

In a traditional certification approach the safety case is developed for the whole systems as a monolithic argument. If the system is modified (no matter how large the change) the whole safety case must be re-evaluated. This does not support the IMA concept of modularisation where changes can be made to individual system modules without impact on the rest of the system. The reduced costs and timescales offered by the IMA approach would offer even greater benefits if they were also carried through into the certification approach. Conmy et al describe a technique which allows this modularisation to be also applied to the safety argument [29]. This modular certification approach depends upon a modular safety argument. Just as the system is decomposed into modules fulfilling specific functions, so the safety argument is decomposed into modules arguing the safety of a specific module (or set of modules). The key to this approach is the combination of safety argument modules into the overall safety case for the system. The key to this is the safety assurance contract [29]. This provides a rely / guarantee contract between two modules. The safety requirements of each module are assessed and then a definition of the relationship between modules is used to develop a set of rely / guarantee relationships that describe how the module interact. As long as these rely / guarantee relationships can be shown to hold the overall safety can be demonstrated. If a change to a module is required the safety assessment for that module can be repeated. Then any guarantees the module makes and any relies it requires must be validated. In this way the rest of the module safety arguments do not need to be

altered and the scope of the change can be constrained. Although this approach can bring significant cost savings, it is likely that some of the safety requirements for the overall system will require a minimal overall safety argument to exist in addition to the modular approach.

It is therefore beneficial to our data safety solution to try to ensure that a modular safety approach can be supported, even though it may not have a direct impact upon the safety structure defined for the data safety problem.

Kelly also describes a modular approach to the certification of safety-critical systems [30]. He describes how the GSN notation can be extended and used to develop a modular safety case architecture for a system. He provides a series of steps by which a modular safety case can be constructed. As part of this assessment he develops a GSN structure for an IMA safety case based upon earlier traditional safety cases for modular systems.

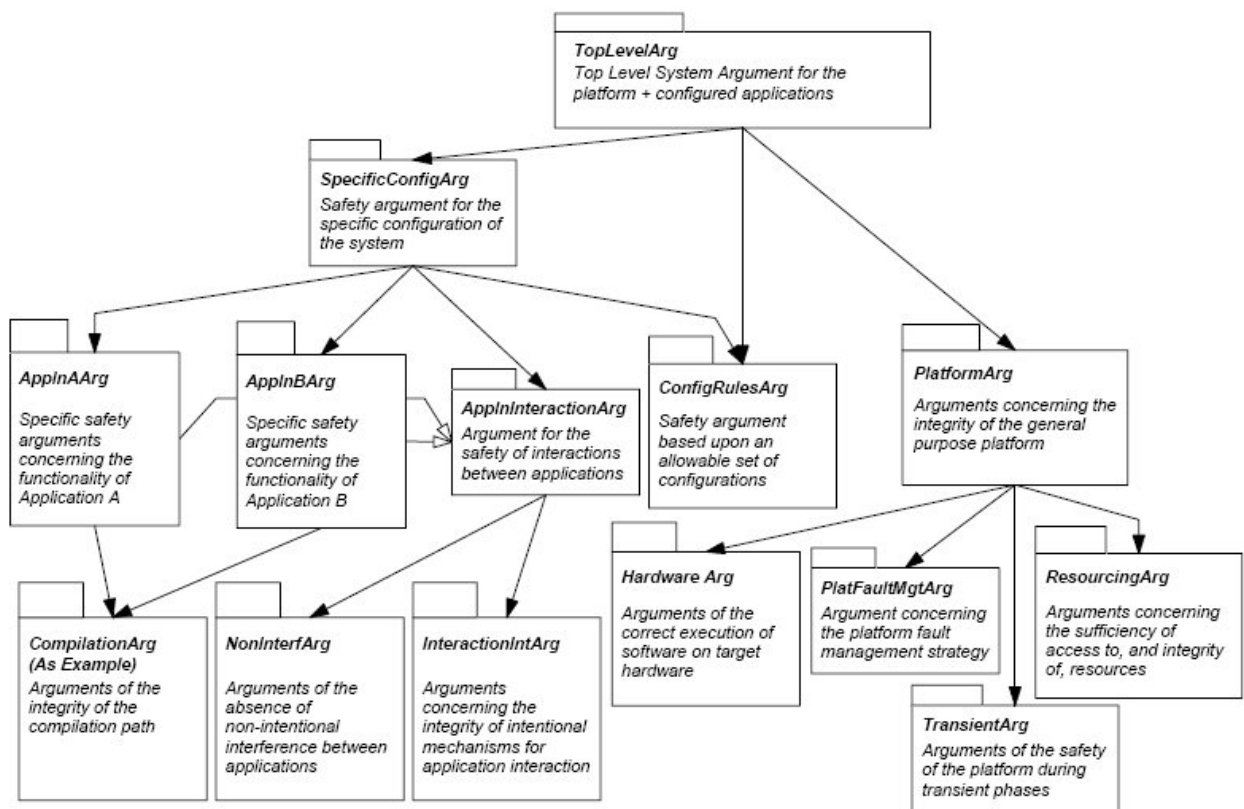


Figure 10 - Kelly's GSN Model for an IMS Safety Argument

This provides us with yet another basis for a top-level IMS safety case that could be modified or extended to include the requirements of data safety. This model also fits with the concept of safety assurance contracts developed by Conmy et al [29]. Kelly uses the concept of contracts to link the modules within the GSN structure and discusses various change scenarios that may cause the system to be altered. As this model supports the modular approach to safety it will form part of our assessment of the safety approach for the IMS data safety problem.

If we now look at some of the IMS techniques that are used within the IMS system management we see that they can add to the safety problem.

Blackwell et al discussed the problem of reconfiguration of modular systems [31]. They define the two approaches to reconfiguration, reconfiguration before and after system operation and operational reconfiguration (including flight). Reconfiguration

during operation is the most difficult to manage. This form of reconfiguration can cause further safety problems as the system must remain in a safe state whilst it changes configuration. Blackwell et al suggest the two main issues that must be handled are timing and consistency. Reconfiguration takes a finite time to occur and during this period the system state must be maintained. This is a difficult issue as communication links may not meet their usual timing requirements whilst portions of the system are between two operation states. Consistency is also a problem as the changes in a sub-set of the system could cause inconsistency in the overall system state and may lead to further fault propagation. Given time this state can be recovered but this may be impossible to manage for a safety related system. These two issues directly impact the IMS distributed data problem and must be taken into account in our solution.

There are two important concepts that are used to allow fault recovery:

- Checkpointing and Rollback – the system component saves periodic snapshots of the internal data of the component as it executes. If a fault is detected the system will be returned to a default state and then the last valid set of checkpoint data will be used to “roll back” the system to the state recorded at that checkpoint. The problem with this technique is that a significant amount of system resources, in terms of time, bandwidth and storage, are required in order to provide the checkpointing facility. Also, the rollback process inevitably requires a system’s normal operation to be halted until the rollback is complete. If the system is required to provide continuous operation to some external system or user this delay can not be tolerated.
- Replication – An alternative to checkpointing and rollback is the use of multiple copies of the system functionality, either used as hot standbys or used in multi-lane voting systems. Here multiple copies of the same system functionality are actively using the same inputs to produce the same outputs.
 1. In a hot standby approach the output of the primary system must be validated by some external means. If the output is seen to fail the system switches over to use the hot standby copy of the functionality. The disadvantage here is the use of extra system resources to provide the standby functionality, and also the problem of providing the output validation. If this validation component itself is subject to failures the standby decision may not be correctly made. This validation component must not just be able to detect output failure conditions (which may be very difficult for a complex data set) but it must also be of a high integrity (both in terms of availability and trust).
 2. A replicated voting system uses the outputs of all replicated components continuously and then votes on the outputs. Output consensus denotes no failure conditions. If all outputs are not the same then the majority rule applies and the module that does not agree is deemed to be faulty. This technique suffers from many of the same problems as hot standby; extra resources are required to implement the functional copies and the voting component needs to be trusted and available. Obviously in order to vote on an output three or more functional lanes are required.

These techniques form the basis of the approaches described below.

Huang and Kintala describe another technique that is employed in IMS systems; that of fault tolerance [32]. They describe five levels of fault tolerance that a system can exhibit:

- **Level 0** – No tolerance to faults in the application software – the system must be restarted from an initial state manually.
- **Level 1** – Automatic detection and restart – the system is capable of detecting a failure and restarting itself from an initial state, possibly on a new hardware resource.
- **Level 2** – Level 1 plus periodic checkpointing, logging and recovery of internal state – additionally to level 1 the system maintains a periodic copy of internal state (a checkpoint) and therefore can restart from a known position during operation.
- **Level 3** – Level 2 plus persistent data recovery – additionally to level 2 the application state is continuously maintained on an external resource and after a failure the system can continue as close as possible to the point of failure.
- **Level 4** - Continuous operation without any interruption – the system maintains the required operation continuously by the use of replicated versions of the system, either by handing over to a backup system or using replicated systems with voting.

In an IMS architecture fault tolerance techniques are defined by the system management component and the data in the blueprint. However, to ensure the IMS system can maintain functionality after data faults our data safety solution will need to utilise an appropriate level of fault tolerance in order to maintain a safe system state. This may mean that the data model will need to allow both checkpointing and rollback, and replicated instance of data to ensure that the system state is maintained or recovered. It is not yet clear if these techniques will be required in order to resolve the state consistency issue.

Data Consistency in Distributed Systems

For our distributed system state problem we need to understand how research into data consistency can help with our solution. Most data consistency research has been performed in the area of databases, both traditional and real-time. Consistency across databases is obviously a significant issue as databases require distributed versions of the data to be maintained. Therefore the research in this area may be able to provide some solutions to the issue of real-time consistency in an IMS architecture.

Babaoglu and Marzullo define a global state of a distributed system as “the union of the states of the individual processes” [33]. This would be a relatively easy concept to manage if the distributed nodes shared a common memory area where the state was recorded, but distributed systems, and IMS in particular, do not use shared memory techniques. Instead each processing node has its own local memory containing its own local state. This means that communication between processes in the system is by the use of messages, represented as virtual channels in IMS. The global state is therefore the union of a set of independently stored local states. However, due to the distributed nature of the processing nodes there are finite time delays when sending messages around the system. This means that it is difficult to ensure that a single meaningful representation of these distributed local states can be obtained at any single point in the system. It is important to take into account these message delays

and ensure a consistency protocol is employed in order to ensure that both the delays and any ordering problems are effectively handled. Babaoglu and Marzullo suggest a snapshot based approach.

The first method based around snapshots requires the use of a global clock that all processes can access [33]. The snapshot method allows a single process to request all process to take a snapshot of their local state at some given future point. This point is chosen to ensure enough time is available to broadcast the snapshot request to all processes in the system. Each process then records incoming messages when the snapshot time has passed. When a message is received on any channel that contains a time stamp later then the snapshot time the recording is stopped. The set of messages recorded by the processor nodes and their internal states at the snapshot time provide a consistent global state. Babaoglu and Marzullo also propose modifications to this protocol to remove the need for a global clock as the snapshot is initiated by the receipt of a message to begin the snapshot from a single lead process. A very similar snapshot based protocol is known as the global state recording algorithm [34] which similarly makes up the state at all nodes by recording an initial state at one node and then the sequences of intermediate messages across the system until all other state nodes are recorded. However, the global state from all of these snapshot protocols, whilst consistent, is still distributed across the processor nodes and the data is not shared at all nodes. It is in fact, as Chandy and Lamport state [34], a recording algorithm. This is still a problem when the global state needs to be shared knowledge across the system in order to make decisions on future operations. The state data could be distributed to all nodes after the snapshot has been completed, but this then means the global state is out of date.

So, if a snapshot protocol cannot provide a global state across the entire system that is visible by all processing nodes we need to look at alternative solutions. An alternative to a global state mechanism such as snapshots is to look at consistency models that are used to ensure replicated data instances are consistent across distributed systems.

Belalem and Slimani propose a consistency model that is used to provide data replication consistency in a Data Grid computing system [35]. They define consistency as “the degree of similarity between copies of a distributed entity” [35]. In order to ensure consistency some form of synchronisation mechanism is required in order to ensure the multiple entities are updated to take into account changes in local copies. This type of mechanism directly relates to the problem of ensuring consistent global states across a distributed IMS architecture. Belalem and Slimani describe two extremes of such a model. A pessimistic approach ensures consistency at the expense of availability, and is therefore less suited to real-time behaviour, and an optimistic approach ensures availability at the expense of continuous consistency. A pessimistic approach uses a strong synchronisation mechanism to ensure all copies of distributed data are consistent and that updates across the copies are performed in a deterministic manner. Access to this global data state is blocked unless the consistency has been checked. The disadvantage is that this mechanism will take significant time and communications bandwidth, impacting on the real-time nature of the system. If this pessimistic approach can be handled within the real-time constraints of the architecture then it provides the ideal consistency mechanism. However, as the real-time constraints may not be able to allow this rigid mechanism an optimistic approach allows individual processor nodes to access local copies of the data even though the consistency mechanism may not have updated the local states. There will be periods when individual processing nodes are not using

consistent copies of the global data, and it may take time before all processor nodes are brought back to a consistent state.

So, these two approaches are the ends of a spectrum of availability and consistency. Neither extreme is likely to give us an acceptable solution for IMS and therefore a trade-off is required between an acceptable level of availability and consistency. Belalem and Slimani propose a multi-layer model where the system is broken into a hierarchy of groups [35]. Consistency within groups can be handled more easily and is therefore more pessimistic, whilst consistency between groups is more difficult and is therefore handled in an optimistic manner. This approach also reflects the hierarchical model of system management proposed within IMS systems.

As we can see from the above discussion consistency in real-time systems is not based only upon the value of the multiple instances of data, but also on the timeliness of the multiple instances of data. The value of a data item may be represented correctly at all processing nodes, but if the data is not available within a timeframe where the data is still relevant then consistency is not maintained. This means a consistency argument needs to consider both the value of all data items and the time at which they are delivered to each processing node. Audsley et al define three forms of consistency for real-time databases [36]:

- Absolute Temporal Consistency – is the data value sufficiently recent compared to the real-world information it represents? If a data value lags behind the real-world state beyond some defined limit (set for each data element) the value is not considered correct.
- Relative Temporal Consistency – do the points in time a set of data values were captured fall within an acceptable time bound? If a set of data values used for a function represent a set of real-world events at significantly different times the function will not be using a consistent set of input data.
- Functional Consistency – if a data element depends upon another data element in the data set there is a functional dependency between the elements. This can also be expressed as a predicate relationship where in order for the state of element 2 to be updated, the state of element 1 must be updated first.

These three forms of consistency allow a better definition of an overall consistency model. Audsley et al use these three consistency definitions to develop a model for transactions in real-time databases [36]. This model is useful as it is based upon transactions. These database transactions are analogous to the updates to local and global state information in an IMS architecture. In order to ensure the three consistency definitions hold two transaction requirements are generated:

- Each transaction must satisfy any corresponding predicate.
- Transactions must occur in such a way that the temporal consistency requirements are met (both in terms of the timeliness of data elements and in the relative temporal spread of a set of functionally linked data elements).

We can use these transaction requirements and consistency definitions in order to assess our solution for the IMS distributed data problem.

Another consideration when considering distributed data models is discussed by Barbara-Milla and Garcia-Mollina. They suggest that the distributed data falls into two categories, core and cached, both of which can be held on modules in the distributed system [37]. A core copy of the data can be updated by transactions on the module as it is considered a primary copy of the data, whereas a cached copy of the data item is a read only copy that is only used as a reference value on a module. This

gives the ability to maintain a system state as a core copy on one processor, and then use multiple cached copies of this data to distribute the state around the system. This concept may allow more control over the updating of a global state condition, as the data is only updated at a single point. The penalty for this more deterministic behaviour is the communications overhead required for a remote module to request a global state transaction on the core module. Even given this disadvantage, this concept will be considered as part of the solution to the IMS distributed data problem. Barbara-Milla and Garcia-Mollina also discuss how concurrency control can be maintained over such a distributed model, particularly the core copies of data [37]. They suggest a number of methods for allowing control of data updates. These all use the concurrency mechanism of read and write locks for a particular data item. Using a lock ensures that only one user / processing node can either read or write to that data item at a time and avoids errors due to multiple simultaneous accesses. A user must wait for a lock to be available before it can set the lock, access the data item, and then release the lock. Using such locks gives the following four possibilities:

- Primary copy – with this method only the site holding the core copy of the data can lock or unlock a data item.
- Read-one-write-all – to read a data item we only need to lock it at our local site. To write an item we must lock the data item at all sites.
- General quorums – sites are allocated a number of votes, and those votes are used to allocate read and write locks. Those sites with more votes have more control over the allocation of read and write locks (these will generally hold the core copies)
- Majority quorums – a special case of general quorums where each site has a single vote and a majority of votes is required in order to read or write to a data item.

These four options provide different strategies for the management and update of data, but they must be matched to the system of interest. For example, the read-one-write-all mechanism requires all sites to be active and functioning correctly in order that the write lock can be set. In a system that allows reconfiguration and fault tolerance it is possible to prevent the update of a data item if this method is not actively managed to keep the list of active sites up to date. An assessment of these mechanisms for concurrency control will be performed as part of evaluating our IMS solution.

A similar concept to the voting quorums noted above is proposed for more general use in overcoming conflicts between processing nodes when handling data in distributed systems by Frohofer et al [38]. They suggest the assigning of weights to partitions that grant a higher level of access to a particular data set. In their solution for networked partitions updating distributed data, a run-time management component assigns weights to partitions based upon the integrity requirements and the relative importance of server nodes within the system. The management component can calculate the weight of a partition at run-time and decide upon the access to the data items in each node. Thus a server node directly responsible for a data item and that needs to be able to provide high availability access to the data will be given a higher weight than other nodes. For another data item another node may be weighted higher, and so on. It seems likely that some form of access prioritisation to the shared data in an IMS system will be required in order to produce a safe data model.

There are other database derived mechanisms that may be used to avoid conflicts in transactions, such as token-based conflict resolution [39], but these mechanisms are likely to add too much complexity for the IMS system management real-time behaviour. This will be assessed further during the data-model assessment.

Whatever protocol is used to ensure that updates are coordinated and consistent, we must ensure that the transactions meet the four ACID properties defined by Van Steen [2]. The four properties are

- **“Atomicity:** All operations either succeed, or all of them fail. When the transaction fails, the state of the object will remain unaffected by the transaction.
- **Consistency:** A transaction establishes a valid state transition. This does not exclude the possibility of invalid, intermediate states during the transaction’s execution.
- **Isolation:** Concurrent transactions do not interfere with each other. It appears to each transaction T that other transactions occur either before T , or after T , but never both.
- **Durability:** After the execution of a transaction, its effects are made permanent: changes to the state survive failures.” [2]

We have already discussed consistency with regard to multiple processing nodes, but we now need to consider how we can ensure that the ACID properties are maintained for each individual transaction.

One mechanism that can allow us to ensure this is Coordinated Atomic Actions (CAA). In order to assess this mechanism we first need to understand some definitions. Schaefer defines the following [40]:

- **Action** – an abstraction that allows the systems application programmer to group a set of operations into a single logical execution unit.
- **Multiparty Action** - an action where several processes cooperate to produce an intermediate combined state, exercise some activity, leave this interaction, and then continue execution independently.
- **Atomic Action** – a multiparty action that allows for exception handling for cooperative error recovery
- **Coordinated Atomic Action** – multiparty atomic action that allows controlled access to shared external resources

Xu defines the Coordinated Atomic Action as “an enclosure of recoverable activities of multiple cooperating threads” [41]. This is a more manageable definition of the concept but needs Schaefer’s definitions to explain the relations between the components. Figure 11 shows the CAA concept.

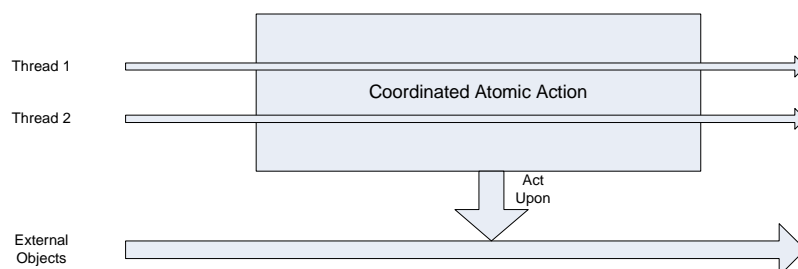


Figure 11 - Coordinated Atomic Action Model

If we utilise the CAA approach to implement transactions on our distributed components we can ensure the ACID properties hold.

- **Atomicity:** the transaction is atomic as the transaction process is contained within the CAA. No other thread or process can interfere and the CAA concept ensures that fault recovery is included in the action. This will ensure either the action completes successfully or that the state is not altered.
- **Consistency:** the CAA must ensure that the transaction only results in valid state outputs. Again, the fault recovery part of the CAA must ensure that the state is always set to a valid value.
- **Isolation:** the CAA concept only allows contributing processes to affect the CAA output and no implicit interactions occur as the atomicity properties ensures the CAA output is either all or nothing.
- **Durability:** the CAA ensures that the output is always set to a stable valid state, so the output will be durable until the next CAA occurs.

Throughout the execution of the CAA the participating processes are cooperative. They utilise shared data in order to coordinate their actions, both in terms of normal functionality and failure handling. The internal processing must ensure that the participating processes come to a collective decision regarding the output of the CAA and in operations over external objects. Note that any operations on external objects may need to be undone by the CAA exception handling if the CAA fails to complete successfully as the output state must remain consistent with any external object operations.

Schaefer also defines a set of steps that can be used to implement the CAA concept [40]:

1. Synchronisation – All participating processes must be synchronised at the start of the CAA
2. Input Validation – All input parameters to the CAA must be validated upon entry
3. Normal Operation - Perform the normal operational functions in order
4. Exception Handling – Whilst performing normal operation handle any failure conditions locally within the CAA
5. Complete Normal Operation – Wait for the operation to complete or timeout
6. Output – set the calculated output parameters
7. Synchronisation – ensure all participating processes agree the output conditions, including success or failure
8. External Objects – complete any transactions on external objects, ensuring no intermediate results are visible externally, and ensuring any fault conditions are passed on as required.

Use of this CAA approach to update IMS global state data will be investigated as part of our IMS data solution.

Proposed IMS Data Safety Process

From our assessment of the research domains that may be able to offer a solution to the IMS data problem it is clear that the two separate forms of IMS data structures that we have identified will present different safety problems.

The most obvious IMS data structure is that of the blueprint. This set of large data files is an obvious component of the IMS architecture and is key to the modular and generic approach. The safety problem here can be summarised as the validation of this large static data file before use and the ability to ensure that it remains uncorrupted at run-time.

The more complex, and less well defined, data structure is the distributed dynamic system state information contained in the system management components of the processing nodes. This data structure gives us a more significant safety issue to solve. Not only must we ensure that the structure used here is valid at the start point of execution, but we must also ensure that the dynamic behaviour provides a trusted and consistent global state representation throughout the system execution.

In fact, these two different data items provide the two extremes of a spectrum of data structures used within real-time distributed systems, of which IMS is an example. In ensuring we have a method to allow safe use of these two extremes we can assume that we will be able to provide a safety argument for other data structures in these systems. This spectrum can be shown in Figure 12.

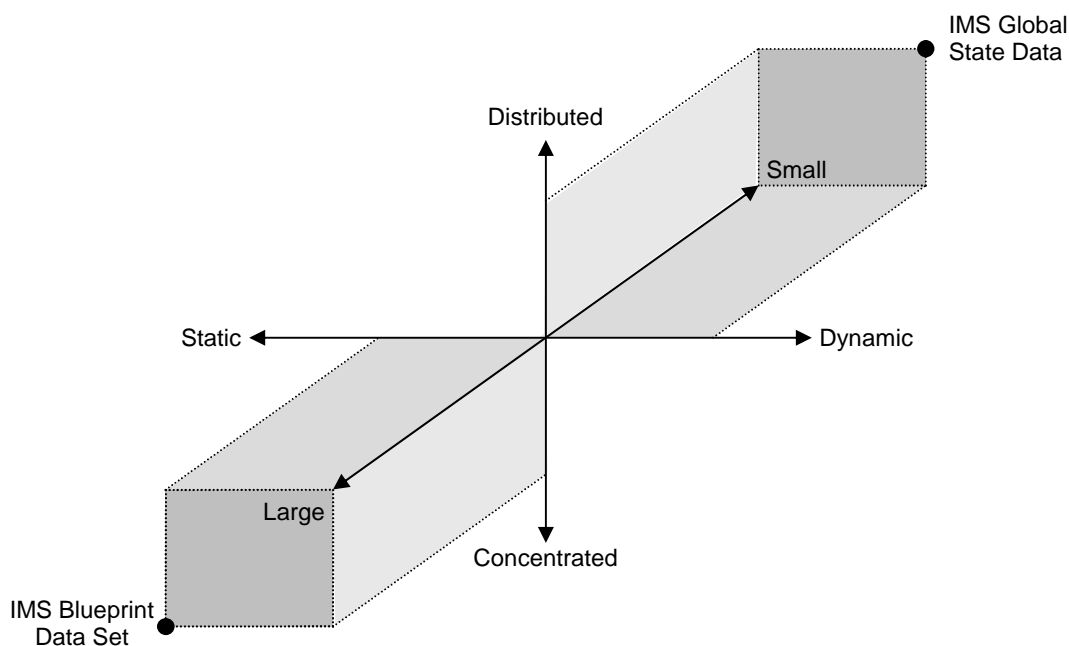


Figure 12 - Spectrum of Data Models in IMS

So, rather than assess one of these IMS data problems we will propose a solution for both and then evaluate what this means for an overall data safety process for IMS.

System of Interest

In order to assess and evaluate our proposed safety process for IMS we must first define an example IMS system that will be used to validate the proposed solution. For the areas of IMS we are discussing, global state data and blueprints, it is the system management and OSL components that are the main focus, as they both contain the distributed state data and are responsible for access to the blueprint data for configuration actions. So, in order to understand the use of this data we need to firstly understand how the data items relate to the IMS software modules, and then provide an example of a distributed IMS system. The IMS software components of interest to our problem can be seen in Figure 13 below.

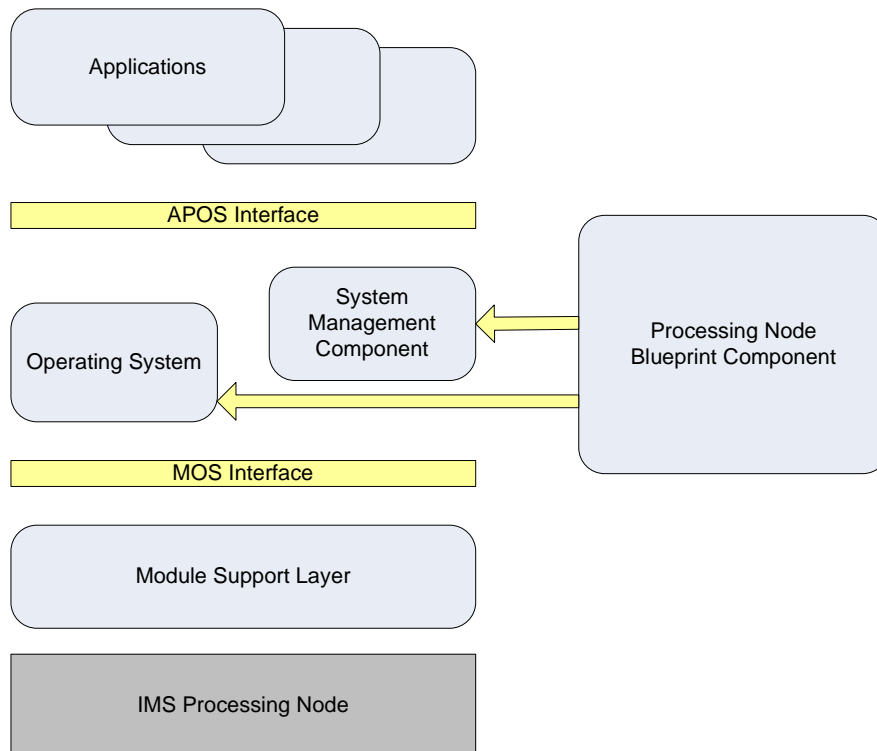


Figure 13 - IMS Software Components on Processing Node

This diagram shows how the blueprint configuration data is used by both the system management component and the operating system layer. The blueprint for each processing node is specific to that node as it only contains the valid configuration data for that specific node and the application modules to be hosted. The data is then used to define the set of applications for the site, the set of communications channels used within the site and externally to other sites, the scheduling of applications, and the fault and configuration decisions to be made by system management. This set of components is present on all processing nodes (or sites) within the system. However, the system management components are hierarchical in nature and therefore separate management components for the other hierarchy layers are also required.

For our system of interest a simple management hierarchy will be used, shown in Figure 14.

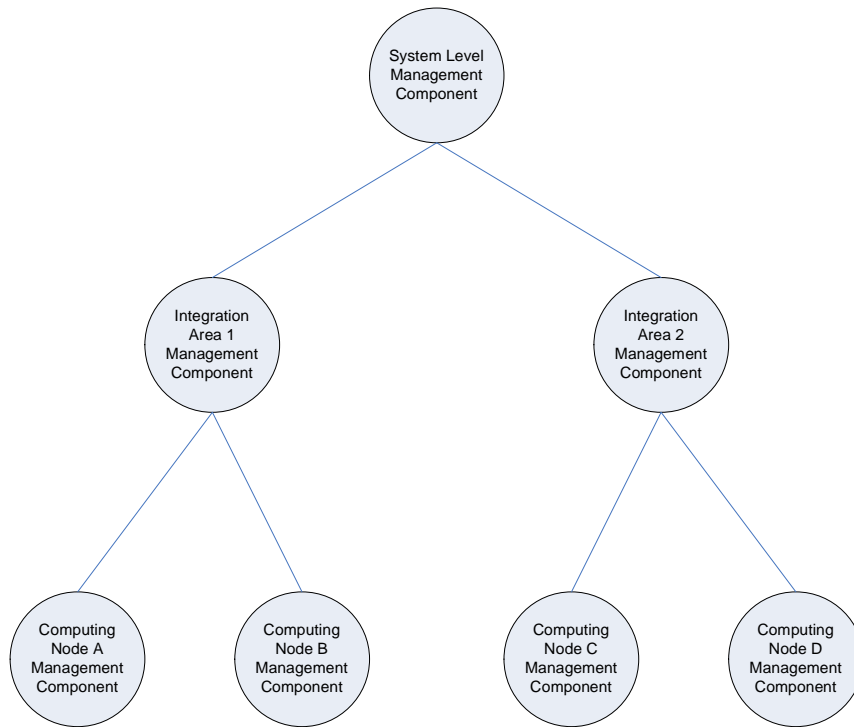


Figure 14 - Basic IMS System Management Hierarchy Example

This contains four computing nodes, each requiring its own management component. We then have two integration areas, each containing two of the computing nodes, and finally a single system level management component.

Finally the following Figure 15 shows how these components of the management hierarchy are mapped onto a representative set of IMS processing nodes, together with the other IMS software components.

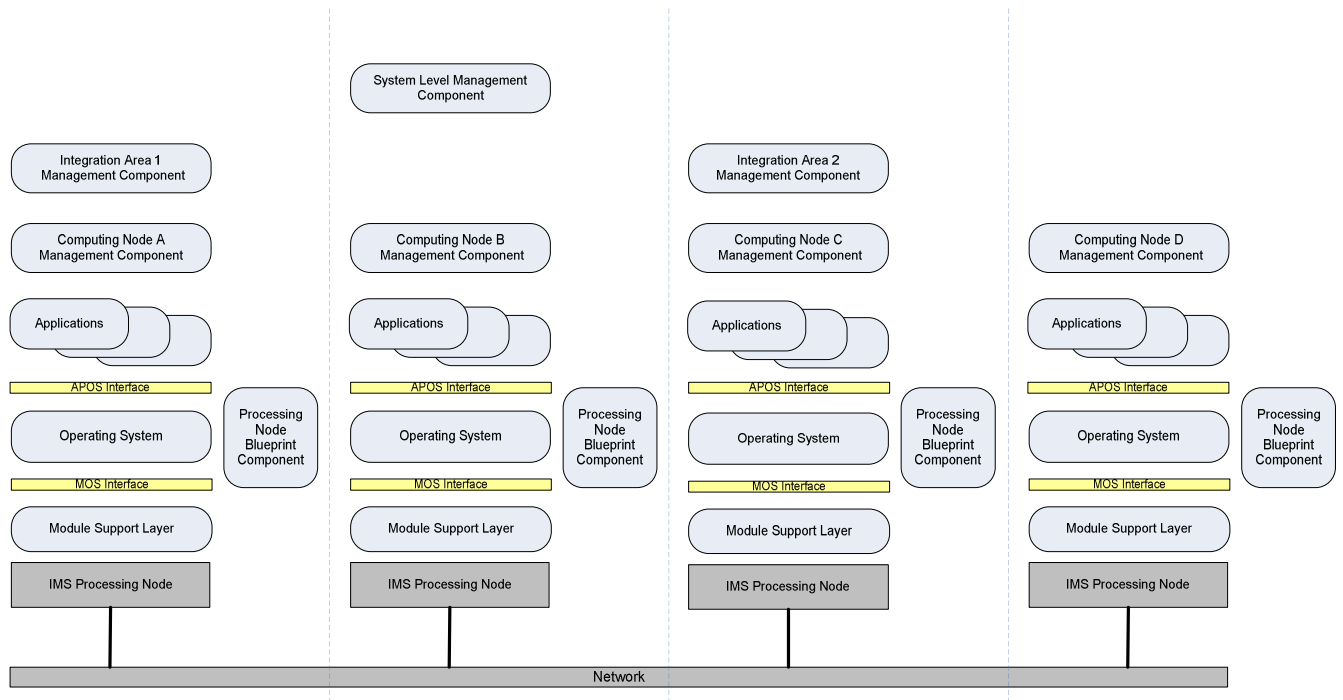


Figure 15 - IMS Example System Model Diagram

In this diagram the components have been repositioned in order to show the hierarchical elements of the system management components. It can be clearly seen that each processing node may be required to host more than one management component as it may need to host an integration area or system level component.

This example IMS architecture will be used to exercise the various IMS transactions and processes in order to understand the safety implications of the architectural approach.

IMS Safety Process

In the assessment of current safety standards we proposed a set of questions to address in order to ensure data safety. These questions are:

- Are the requirements for the data complete and unambiguous?
- Has the appropriate process been used to create the data?
- Have the hazards associated with the data been identified?
- Have all the risks associated with the data been identified?
- Have all hazards been mitigated to an acceptable level?
- Is the flow of the data through the system understood and defined?
- Does a specification exist for all safety related data structures?
- Is the implementation of each data structure traceable to the specification and requirements?
- Is there a safety argument for the data in the system?
- Has all evidence been supplied to satisfy the safety argument?

Whilst these issues are generic and can be used to assess both ends of our data structure spectrum the solution required for each will be different in some areas.

So in order to evaluate the safety of each of these extremes we can define a series of steps. Some of these steps are generic to all IMS system data and therefore need only be performed once.

Data Safety Process for IMS

- Define safety argument structure for IMS data safety
- Define data lifecycle model for IMS data
- Identify data model for IMS data structures
- Justify how this model fits properties of interest
- Identify safety issues with this data model (hazards and risks)
- Identify possible mitigation strategies for these safety issues
- Identify data requirements
- Populate data models to produce data specification
- Validate data against system design and ensure traceability of data to requirements
- Check list of hazards against populated model

- Have the hazards associated with the data been identified?
- Have all the risks associated with the data been identified?
- Have all hazards associated with the data been mitigated?
- Generate safety evidence for each element in safety argument (some of the evidence may be generic for all IMS systems)

Once these generic steps have been defined we will have a framework that can be used to assess the safety of data for each implementation of an IMS system.

In order to use this process we need to derive some of the processes and components. In order to do this IMS data safety argument modules and IMS data models are proposed. We can then follow some of the other steps in our proposed process in order to evaluate these concepts.

IMS Data Safety Case

In the review of IMS safety research three different safety argument architectures were discussed. However, none of these argument structures directly addressed the problem of data safety. We have seen during the review of safety standards that the presentation of an argument that any system component is safe is one of the basic mechanisms in a safety process. It provides the focus to enable us to demonstrate, via evidence supporting the argument made, that the component is acceptably safe. In order to provide such an argument for data safety in IMS we need to either extend the GSN arguments developed by Nicholson, Joliffe and Kelly or develop a new safety argument model. One technique that could be used is the concept of safety argument GSN patterns. A GSN pattern is a part of a GSN argument developed in order to provide a generic template to cover a particular safety concern [42]. A safety case pattern could be established for IMS data safety, or two patterns, one for blueprint safety and one for distributed state data.

In Figure 10 Kelly proposed a safety argument that supported modular certification approaches. Given the modular nature of an IMS architecture, and the cost and timescale advantages to be gained in allowing modular approaches to be used in the system development and certification processes as well as the software architecture, it would be beneficial to ensure any IMS data safety GSN patterns are compatible with this modular approach. However, in order to do this we need to understand what this means in terms of IMS data use.

In Figure 10 Kelly proposes a number of argument components that cover aspects of the overall system[30]. One of these, the PlatformArg, covers those aspects of the safety argument that are related to the general purpose platform. This is then split down into a number of other modules. It is in this argument module that we need to consider the safe and consistent management of the system state data. In Kelly's model this PlatformArg is split into four modules:

- HardwareArg – this covers the correct execution of the software on the target hardware
- PlatFaultMgtArg – this covers the platform fault management strategy
- TransientArg – this covers safe operation during transient phases such as initialisation, reconfiguration and shut-down
- ResourcingArg – this covers the sufficiency of resources and the ability to access them in a safe manner

Although not contained within Kelly's suggested safety case diagram he suggests that there are two other modules that also have an impact on our data safety problem [30]:

- ModeChangeArg – this covers the ability of the platform to handle the process of reconfiguration, including state preservation and recovery.
- ConfigurationRulesArg – this covers the safety of the rules used to configure and reconfigure the platform.

In order to allow the development of our three new data safety argument modules we need to understand if the scope of the PlatformArg module is correct and if the new data safety modules can be included. In the current breakdown of the PlatformArg a number of the concepts that are contained within either the blueprint configuration data or in the combined system state data are split across the modular argument boundaries. Also, the SpecificConfigArg (concerning the safety of a specific configuration of application modules) module is outside the scope of the PlatformArg module, and is part of the application argument branch of the safety case.

So, in order to allow specific data safety modules to be established it is necessary to define what goals these new modules will state. We can then define a structure, based upon Kelly's, which allows these new modules to be contained within a modular IMS safety argument.

For the blueprint, there are two different aspects to consider. Firstly there is an argument to be made about the development of the blueprint data. The blueprints for IMS are generated by a blueprint tool. This tool accepts design data for the IMS applications and configuration information for the generic IMS components and produces a set of run-time files for each processing node. According to our data safety process we need to consider the complete data lifecycle and this will obviously include the development of the blueprint data using the design information and the tool. So, part of our blueprint safety argument needs to cover this development aspect. However, this activity is outside the scope of the modular system as it is an off-line process. This then needs to be considered as a separate pattern structure from the safe usage of each run-time blueprint file. These run-time components are contained within the generic OSL layer of each processing node and therefore the pattern can be linked into the modular safety argument structure for the OSL, subject to the same interfaces to the other IMS safety case components.

For the distributed system state data we need to consider only the run-time aspects, all of which are contained within the generic IMS system management components. The initial configuration of this state data is a key safety issue, and in fact this is directly linked to the correct instantiation and use of the run-time blueprint component.

Blueprint Safety Case Module

We have identified that the blueprint argument needs to cover the development of the blueprint off-line and the use of the blueprint on-line. If we first consider the off-line argument it must ensure that we cover not just the development of a single blueprint, but also the process by which we develop all blueprints. A proposed argument structure is given in Figure 16.

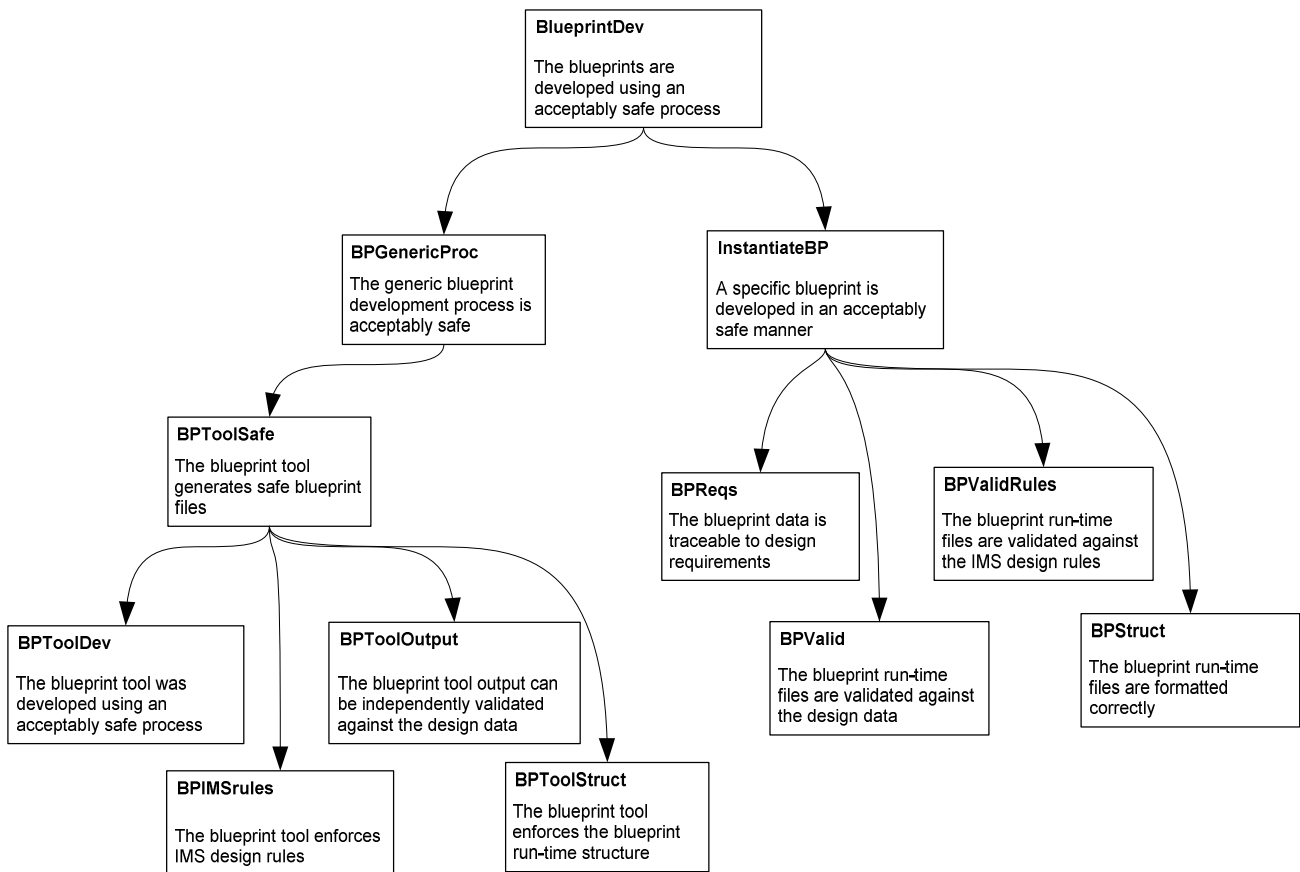


Figure 16 – Blueprint Development Safety Argument Structure

This safety case module for blueprint development uses two main arguments. The first argues that the tool used to develop the blueprints is in itself safe. Due to the complex nature of such a tool it is not possible to formally prove that the transformations the tool performs on the design data to create the blueprints are safe. Therefore, in order to argue that our development process is safe we need to argue that the tool itself was developed using an acceptably safe process. This form of argument, over the process used for development rather than formal proof of the operation, is one of the basic techniques demanded by the safety standards reviewed earlier. However, as we cannot be entirely sure the tool transforms the information correctly we can also test the output of the tool to ensure it meets the expected results. This is also a technique described by the safety standards and is analogous to the testing of software before its use. So, the argument also requires that:

- The output of the tool can be independently validated against the design data – i.e. that some independent path can be shown to exist to ensure traceability of output to input of the tool.
- The blueprint tool enforces the structure of the run-time blueprint files – this must either be done by formal proof of a small subset of the tool output software or by rigorous testing
- The blueprint tool enforces the IMS design rules – this must either be done by formal proof of a small subset of the tool output software or by rigorous testing

The second argument used covers the individual blueprints generated for each IMS system. We are again using an argument based upon correct process and output validation. The argument over each set of created blueprint data is an argument over

output validation. In this case we are again checking that the major blueprint file components have been created correctly by using validation against inputs and generic IMS design rules / structures. In this case we will have real blueprint files to validate, although as these files can be large the validation process may be a significant task. However, the entire operation of the IMS system relies on the correct instantiation of these blueprint files as they control every aspect of the execution of the application software. They have to be correct in order for the system to operate. Therefore the validation task must be performed, whatever the cost, to ensure the system is safe.

The second part of the blueprint safety argument concerns the use of the blueprint data at run-time. We have satisfied ourselves that the blueprints have been created correctly, but we must now make sure the IMS can make use of them correctly.

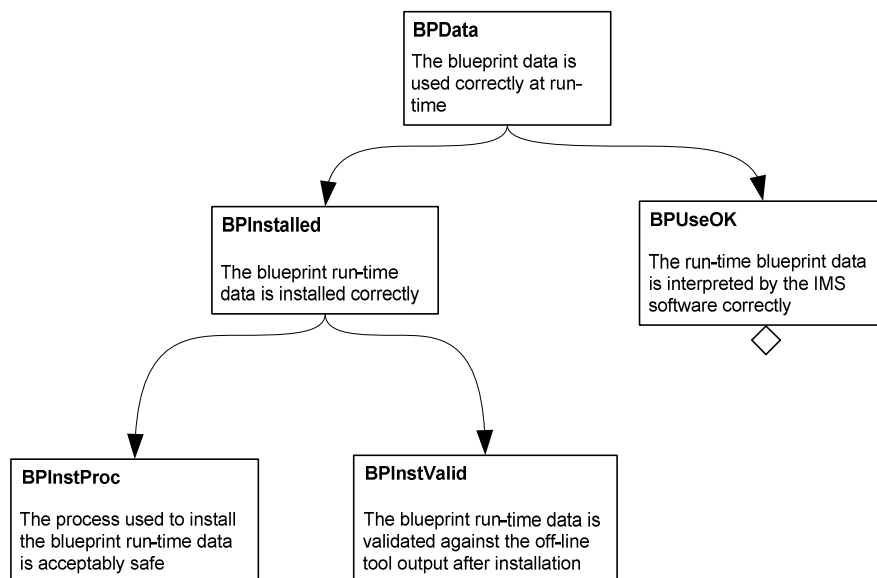


Figure 17 - Blueprint Use Safety Argument Structure

The argument over the safe use of blueprint data is more difficult. We can develop the argument for correct download and installation of the blueprint run-time data, as this can be based upon an argument over the process used and then a validation of the installed information against the off-line original. However, on order to show safe use we must also be able to argue over the safe use of the data by the IMS software architecture. This section of the safety argument falls outside the scope of the data safety argument. This needs to be assessed in terms of the overall IMS safety argument structure.

Distributed System State Safety Module

The argument for the distributed global state is based around three lower level arguments. The first of these is directly related to the initial data provided by the blueprints to set the initial state. This data must be based upon blueprint data as it requires the mode and configuration for each processing node to be established. This argument establishes that this initial distributed state is a valid combination of the lower level state values that comprise it. This is necessary as not all combinations of processing configurations and modes will result in valid system states. We will also assume for this simple IMS system that all processing nodes are fault free at start up of the system, although the IMS concept does not necessarily assume this. In a real IMS implementation it may be allowed to continue to use the system after some resources have failed. Alternative configurations would be required in the blueprint to support this fault tolerance approach.

The second sub-argument concerns the validity of the global state at run-time. As the global state is made up of the distributed local states this argument is then based upon these distributed local states. We need to ensure that all the local states remain valid throughout operation, and that the collection of these global states remains consistent throughout operation. It is this second local state argument that will be most difficult to prove as it requires us to solve the distributed consistency problem. This will be investigated as a separate discussion.

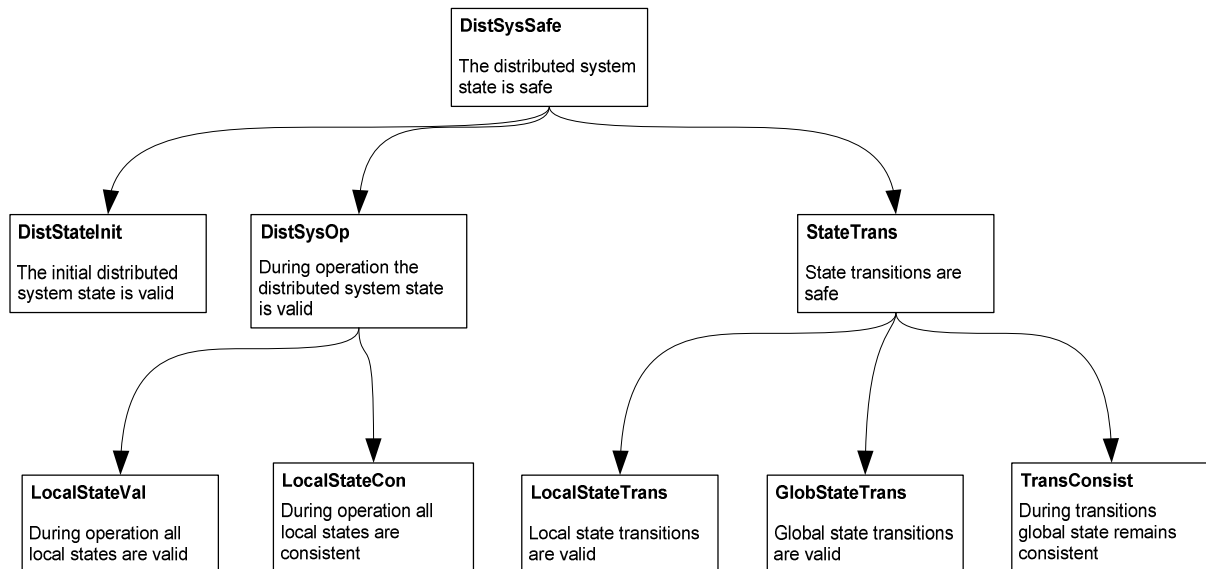


Figure 18 - Distributed System State Safety Argument Structure

The third element to the argument covers the transitions between states. This argument requires that local state transitions are valid. This means that the new state is a valid change of state from the old state, as not all states can be reached from all other states. The global state transitions must also be valid. Not all combinations of local state conditions result in a global state that is valid, so even though any single local state change may be a valid transition, if it then results in an invalid global state it cannot be allowed. Also, in order to cover the transient periods of the state change process we must ensure that consistency is maintained throughout the transition between states. Much of this argument relies on the use of distributed systems theory and consistency mechanisms to ensure that the argument can be shown to hold. As before, this will be investigated as a separate discussion.

We have now created three IMS data safety argument modules. It is important to understand how these modules can be incorporated into an overall argument such as the argument presented by Kelly [30]. In fact the two run-time argument modules, the blueprint data is used correctly at run-time and the distributed system state is safe can be used as part of the PlatformArg argument module defined in Figure 10 - Kelly's GSN Model for an IMS Safety Argument.

The previous discussion of the scope of PlatformArg suggested that some of the blueprint concepts were split across module boundaries, but whilst some of the information provided in the blueprint is used across various argument modules our two new blueprint argument modules do not try to argue about the use of specific elements of the blueprint data but rather argue about the validation of the blueprint as a whole. In this way we are arguing about the generic structure of the blueprint and are not causing contention between the modules defined by Kelly. In the same way the distributed system state argument is focused upon the specific state data. The

initial state is provided by the blueprint, but it does not cause contention with the blueprint argument. Therefore, both the run-time blueprint argument module and the distributed system state argument module can be added as new modules within the PlatformArg module in Figure 10. The off-line blueprint argument does not form part of this platform argument. In fact it lies outside the scope of any of the current argument modules below the top level argument in Figure 10, and will form a new independent argument module directly below the top level argument.

The methods required to provide evidence for these new argument modules will be discussed in the following sections.

IMS Data Lifecycle

The lifecycle for IMS data is based around the blueprint development process. In this instance we do not have to take account of the distributed state data model separately as the initial configuration of the global state is based upon blueprint data, and subsequent updates are managed by the IMS software components. The lifecycle can then assume that the safety of the blueprint will determine the safety of the initial global state of the system.

The blueprint lifecycle can be defined as:

- Design IMS applications using IMS design and structural rules
- Define IMS architecture components required to support specific system implementation (formulate system layout based upon library of IMS components)
- Populate blueprint generation tool using source data from:
 - Application design data (for application software process details)
 - Application integration mapping (for communications links)
 - Application scheduling information (from allocation process – may be a targeted allocation tool)
 - IMS system management hierarchy information (the hierarchy structure used for this specific system implementation)
 - System resource information (for numbers of processing modules, etc.)
- Create run-time blueprint instantiations for each processing node in IMS implementation (using tool)
- Validate tool output against system and application requirements
- Validate tool output against IMS structural and design rules
- Download run-time files to IMS system
- Validate installed run-time files against off-line tool outputs

This process is iterative as system development is an evolutionary process. As the system is developed and tested the blueprints require constant change to define the required system instantiations. This means that the blueprint tool and supporting lifecycle = is in constant use until the system implementation is frozen for delivery. It will then be used to support system modification and “bug fixes”. However, it can be seen from the lifecycle for blueprint data that each stage of the defined process has formed part of our data safety argument and therefore we are ensuring that our defined lifecycle is consistent with our approach to arguing the safety of the data.

IMS Data Model

The next step in our generic process is to define the data model for the IMS data structures under assessment. This must be done for each of the data structures, but the model will then be generic for all instantiations of this data.

Distributed Dynamic Global State Data

We will begin with the analysis of the more difficult of the two IMS data structure concepts; the distributed dynamic global state data. This data structure will require not only evaluation of the structures to be used but also of the real-time dynamic processes used to update the global state and ensure consistency. However, before we can assess the safety of the processes used to manipulate the data we must have a clear and concise specification of the representation of the data.

Information and Data Model

In order to assess the global data structures against our suggested list of safety questions we need to define a more formal model. As was seen in our evaluation of data-driven systems, use of an appropriate data model is a basic step in ensuring an acceptable data lifecycle can be defined. Note that there is a difference between an information model and a data model. An information model is an abstract description of information and its relationship to other information. A data model is a more formal representation of how the data will be represented and provides an instantiation of the information model for a particular implementation.

The information model for the global state data can be defined by understanding the properties of interest within the IMS system management component, and understanding how these relate to each other, any influencing components / data structures, and any components / data structures that are influenced by the global state.

The information model for IMS system state is actually dependent upon three separate sets of decisions; health status information derived from fault reports, mode information based upon application derived mode change requests and the current configuration of a particular system component. In addition to these three state components there are three types of communication that cause these state components to change: fault / health reporting, mode change requests, configuration change requests. The last of these, configuration change requests, form a link between the mode and health state data, as a fault report may lead to a configuration change request that then causes a mode change.

The system management hierarchy, the required state data and the communications that control the states are shown in the following diagrams.

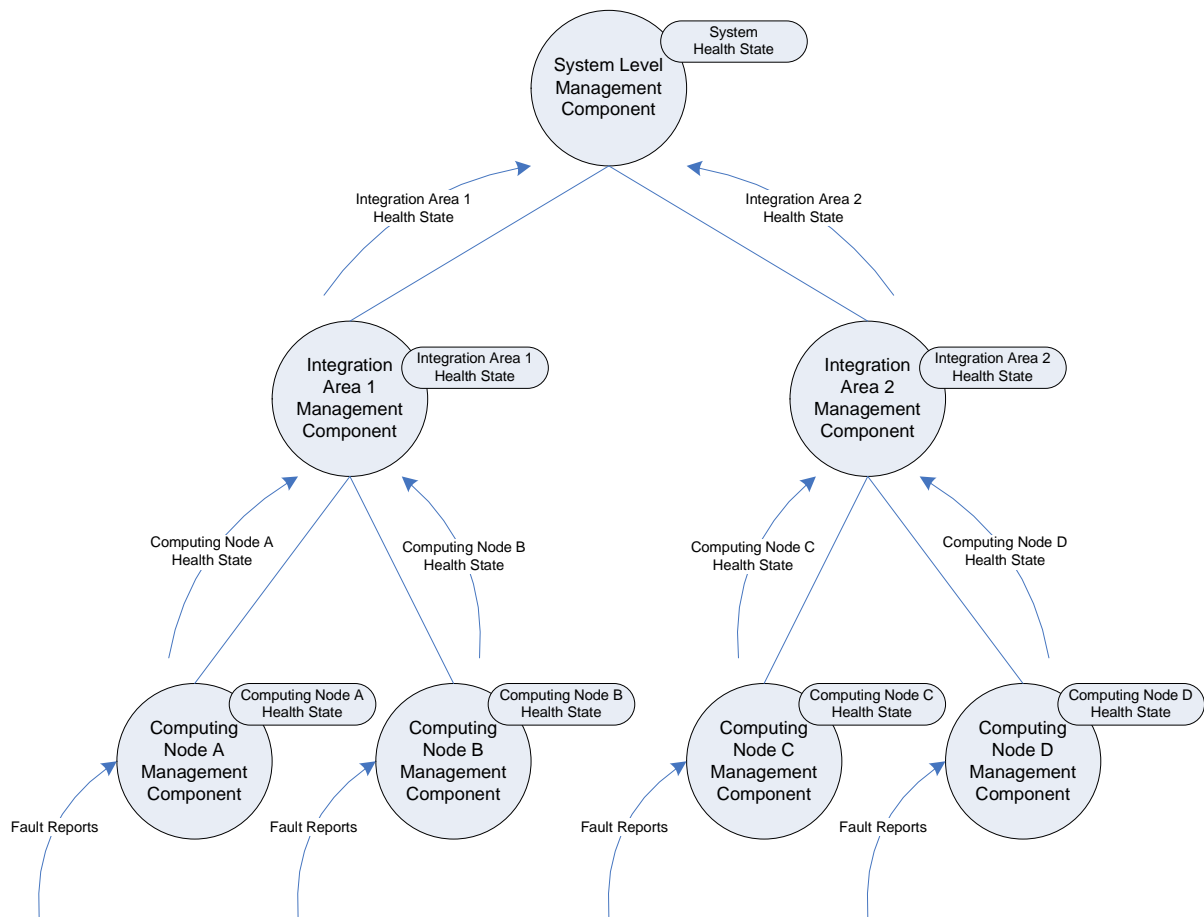


Figure 19 - IMS Health State Example

We have now added a health state for each of the management components, the fault reports and health state reports that are used to control these states. The fault reports are derived from health monitoring routines on each computing node and are therefore handled by each computing element management component. These directly affect the health state of the computing node. The health state of an integration area or the system cannot be directly affected by these fault reports as they are abstracted by the hierarchical structure. The health state of an integration area is based upon the health state reports from the computing nodes. So, for example, Integration Area 1 Health State is derived from Computing Node A Health State and Computing Node B Health State. Similarly the system state is dependent upon the health states of the integration areas. On its own this health state model is not complex. However, the ability of IMS to reconfigure and change mode adds significant complexity to the model. When we examine the configuration state example we will see that the above Figure 19 needs to be linked into the configuration model as a fault report or health status message can lead to the need to reconfigure the system to overcome a fault condition. These links will be shown in the complete structural diagram.

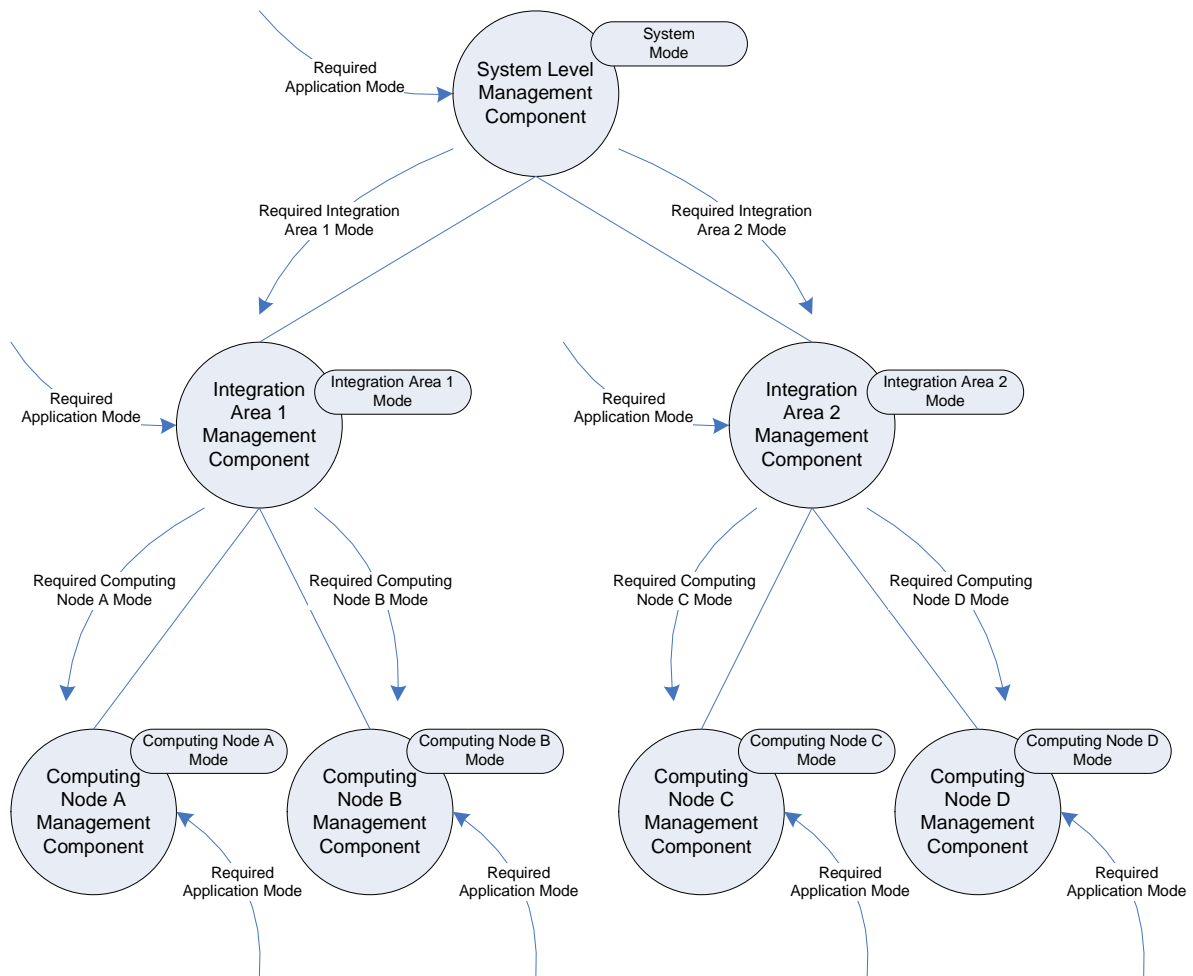
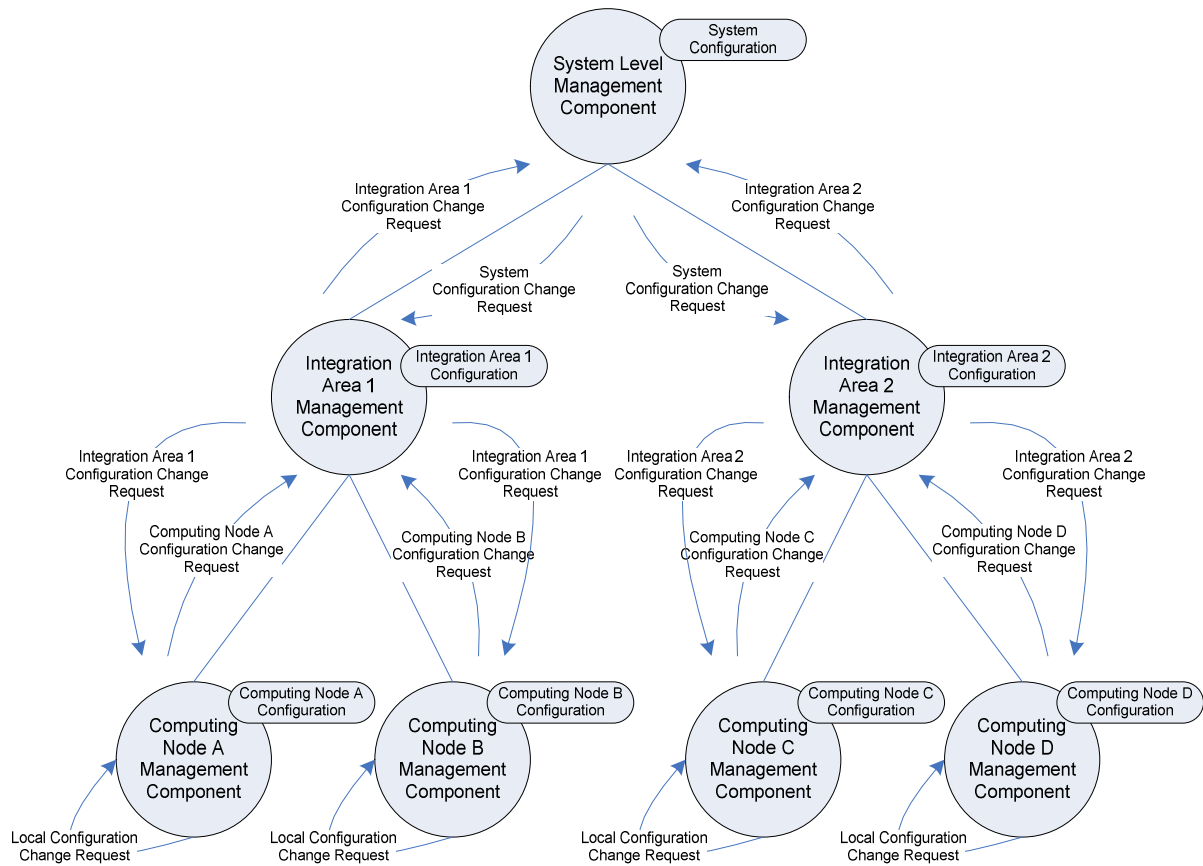


Figure 20 - IMS Mode State Example

If we now consider the mode of the various IMS components in the system we can see that it is more complex than the health state due to the ability for application mode requests to occur at any level within the system. Also, where the health state is passed up through the hierarchy to provide an overall system health state the mode change requests flow down from the top level to the computing nodes. This is due to the nature of how the application software is structured to use the system and integration area levels as modular boundaries around specific grouped functionality. For example, an integration area may be allocated a specific function such as navigation. The navigation application will have internal moding but this will request mode changes at the integration area level. Similarly, a component of the application on a specific computing node may request a local mode change but if this component needed to change the overall navigation mode it would make this request at the integration area level. This principle of requesting mode changes at the appropriate level to ensure all resources are informed also applies to system mode changes. What is not shown on Figure 20 is the link to configuration state. The primary concept for implementation of mode changes using the IMS management hierarchy is reconfiguration. If a mode change request is made to a management component the only method the generic IMS software architecture has for applying the change is to implement a new configuration for the application software. It may not occur directly at the level where the request was made, it may be passed through the hierarchy to the appropriate level. These links will be shown in the complete structural diagram.



IMS Configuration State Example

It can be seen that the configuration state change model is very complex as it allows for configuration change requests to be passed both directions through the hierarchy. The local configuration change requests at each computing node represent the need to reconfigure due to fault conditions at this level, but no other fault or health derived messages are shown. Also, no mode change requests are shown. These would add further levels of complexity to the model. The complete structural diagram, Figure 21, below would be very complex if we tried to show all of these links in the form of the management component hierarchy view we have used so far. In order to allow the links between the various state models to be clearly identified this final diagram will use the state components at each level as the start and end points of communications.

The complete state transition diagram, Figure 21, for the IMS example system shows the complex nature of the state model in the IMS system management hierarchy. One of the most important issues to note with this complex model is the possibility of circular paths and dependencies in the defined transitions. These circular paths are also subject to possible time delays as the different management components may be on separate computing nodes. This possibility of circular dependencies and time delays before state data can be communicated to all nodes presents a significant safety problem. In order to investigate this further we need to create a better definition of the data structure.

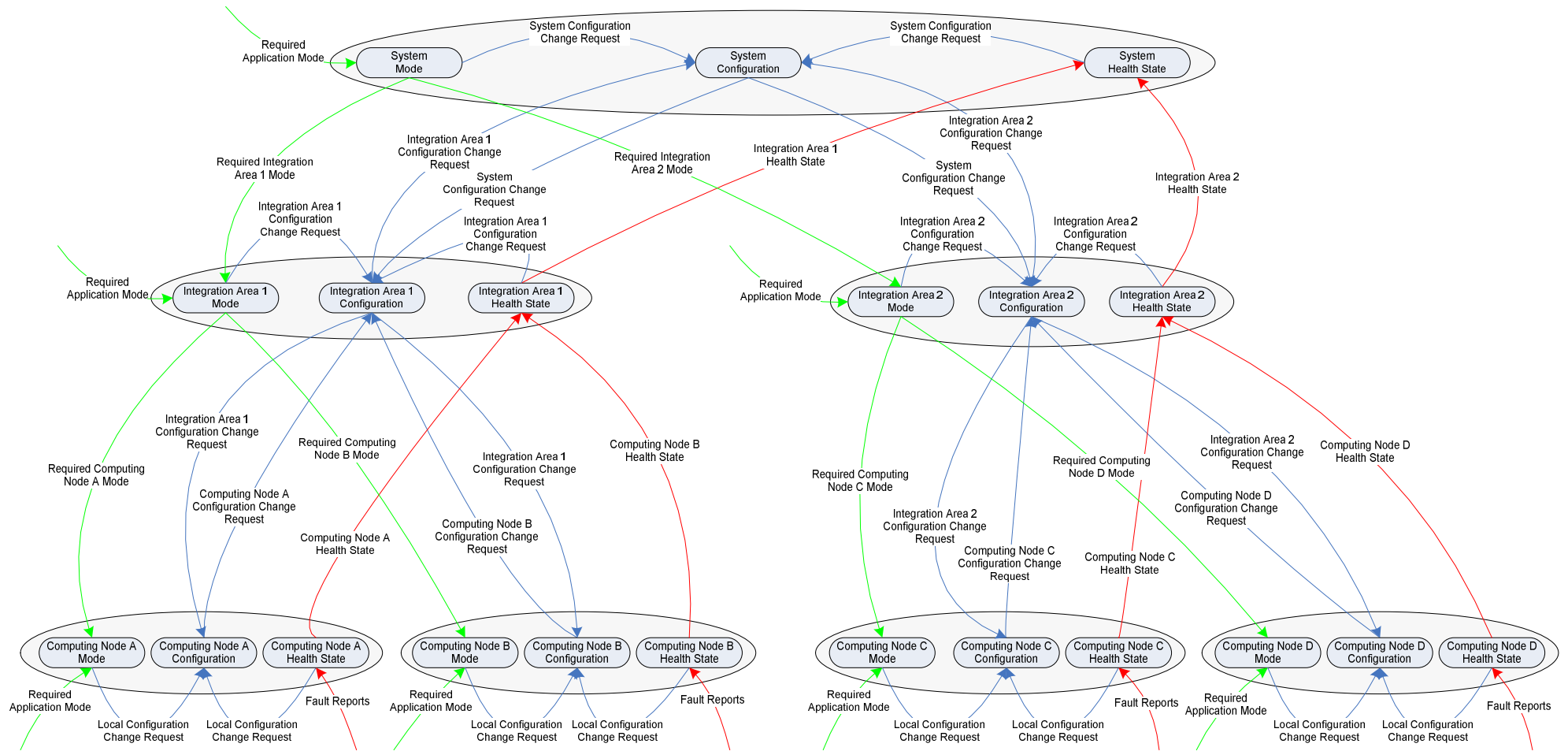


Figure 21 - Complete IMS state diagram example

Information Model

One of the important steps in our suggested safety process is the definition of the data model. However, before we develop the data model we will first develop the information model for each of the types of state data in the system.

The health state information model can be represented as:

System Health State is:

- Set of computing node system management components each containing:
 - Computing node health state:
 - Influenced by locally reported fault conditions
 - Influencing integration area health state
 - Influencing computing node configuration state
- Set of integration area system management components each containing:
 - Integration area health state:
 - Influenced by computing node health states
 - Influencing system health state
 - Influencing integration area configuration state
- System level system management component containing:
 - System health state:
 - Influenced by integration area health states
 - Influencing system configuration state

The mode state information model can be represented as:

System Mode State is:

- Set of computing node system management components each containing:
 - Computing node mode state:
 - Influenced by computing node mode requests
 - Influenced by integration area mode state
 - Influencing computing node configuration state
- Set of integration area system management components each containing:
 - Integration area mode state:
 - Influenced by integration area mode requests
 - Influenced by system mode state
 - Influencing computing node mode state
 - Influencing integration area configuration state
- System level system management component containing:
 - System mode state:
 - Influenced by system mode requests
 - Influencing integration area mode state
 - Influencing system configuration state

We can see that the mode state is more complex due to the bi-directional nature of the influences between the hierarchical levels. In the health state model the health status is always passed up through the hierarchy, but in the mode state model mode

changes can be initiated at all levels within the system and may need to be handled by a higher or lower hierarchical layer.

The configuration state information model can be represented as:

System Configuration State is:

- Set of computing node system management components each containing:
 - Computing node configuration state:
 - Influenced by computing node health state
 - Influenced by computing node mode state
 - Influenced by integration area configuration state
 - Influencing integration area configuration state
- Set of integration area system management components each containing:
 - Integration area configuration state:
 - Influenced by integration area health state
 - Influenced by integration area mode state
 - Influenced by system configuration state
 - Influenced by computing node configuration state
 - Influencing computing node configuration state
 - Influencing system configuration state
- System level system management component containing:
 - System configuration state:
 - Influenced by system health state
 - Influenced by system mode state
 - Influenced by integration area configuration state
 - Influencing integration area configuration state

It can be seen that the configuration state is completely influenced by states within the system management component. It does not receive external requests to change configuration as the configuration state is an internal property of the IMS system management hierarchy.

Note that this is actually a simplified model of an IMS architecture as certain assumptions have to be made in order to reduce the complexity. These assumptions are:

- There is a single (un-replicated) system level management component
- There is a single integration area level within the system which may consist of a number of independent integration areas (but without nested integration areas).
- There are no computing node management components directly reporting to the system level component
- Each computing node contains a computing node system management component therefore the integration level and system level management components will reside on computing nodes that also contain computing node management components

- Our information model holds through fault conditions and reconfigurations (i.e. we do not need to take special measures to deal with reallocation of system / integration level components on failed sites)

We can see from this that our information model could be made significantly more complex if these assumptions were not made. This natural English description of the information model allows us to understand the intent of the distributed global state, but it is not precise enough for us to evaluate the safety of the system. For this we need to instantiate this information model into a data model.

For any model of distributed state data we can use the following outline of the structure of the data

- distributed data *is represented by*:
 - multiple data items *where*:
 - overall system state *is represented by*: consistent set of individual data states *where*:
 - each individual data item state *is represented by*: set of properties *where*:
 - properties *are represented by*:
 - value *which is one of*:
 - discrete bounded
 - discrete unbounded
 - continuous bounded
 - continuous unbounded
 - character representation
 - accuracy *is represented by*:
 - resolution
 - accuracy
 - consistency of units
 - stability
 - timeliness *is represented by*:
 - permanence
 - longevity
 - dependency *is represented by*:
 - other dependent data items
 - consistency requires all data item properties be evaluated within a specified timeframe *represented by*:
 - timeframe *is represented by*:
 - value
 - resolution

From this structure we can define a set of parameters that will make up the data model. This set of parameters needs to ensure that all safety properties can be assessed. For the data model for the IMS distributed state problem we will include the following items:

- Structure – a set of data items that represent the information model
- Properties – each data item needs to display the following properties:
 - Value – the form of the data item. This will define whether the value is a discrete set of predetermined values, a continuous numerical value, or some form of representation such as a string of characters.
 - Accuracy – the constraints on the resolution and accuracy of the representation. This will depend on the type of value. For a discrete set, or string representation, there is no accuracy or resolution, but for a continuous value accuracy and resolution are required.
 - Range – the limits on the range of values the data item can contain. This can be defined for all types of data item, even if it is set to the maximum the processor can handle.
 - Timeliness – the permanence / longevity of the current value. This can be limited to a specific value or permanent (where permanent means it is valid until the next operation on the data item).
 - Dependency – any links to other data items. This is used to show direct links to other data items in this specific model (such as data items that form part of a larger data structure). Interactions with other data structures are shown through the manipulation operators.
- Integrity Constraints – constraints on the model that ensure structural integrity such as limitations between two individual data items. An example of this would be the use of a Boolean data item that then specifies the use of a continuous value data item. The value in the continuous item would be constrained by the value of the Boolean data item.
- Manipulation Operators - the operations that can be applied to the data. This would provide a set of operations that can be performed in order to modify the data items. It needs to define the changes made by each operation and the prerequisites for that operation to be performed. An example would be the manipulation operation to update to a health state would need to define:
 - any pre-conditions for the operation to occur,
 - the trigger for the operation to occur,
 - the modification to the state provided by the operation,
 - any post conditions after the operation had completed.
- Consistency requires all data item properties be evaluated within a specified timeframe that has value and resolution

This set of data has been chosen in order to satisfy the RTCA DO-200A set of requirements for data [16], and to ensure the specification of the data structure is robust to allow us to assess the safety aspects. However, for our IMS state data model some of the more complex aspects that deal with continuous data values and timeliness will not apply. This is due to the discrete nature of the states and their representations in the model.

The data model needs to be defined for the three levels of the IMS system management structure. Note although IMS allows for multiple levels of integration area to be defined, we will continue to use a model with only one integration area level between the system and computing node components. This may modify the

data model as we do not have to deal with interactions between integration area levels. Whereas the information model was defined across the management hierarchy for a specific type of state data, the data model will be level specific for all types of state data. This is because the data model needs to define the data structure used within a particular software implementation component, and these are separated across the computing nodes. Also, the different management components in the hierarchy are implemented as separate software processes, even when two levels both exist on a single site. This ensures the generic nature of each management component can be enforced.

Computing Node State Data Model

We will initially define the data model for the computing node management component. Although there are multiple computing nodes in the system, the data model is generic. This means the data model is for a single computing node instance.

Computing Node State *is represented by:*

- Health State *is represented by:*
 - Value (one of (healthy, failed, (set of partial fail states)))
 - Accuracy (discrete)
 - Range (set of predefined health states)
 - Timeliness (permanent)
 - Dependency (none)
 - Integrity Constraints (none)
 - Manipulation Operators
 - Change computing node health state due to fault report *requiring:*
 - Pre-conditions (health state not “failed”)
 - Trigger (computing node fault report)
 - Modification (updates computing node health state value, sends new health state to integration area management component, initiates computing node configuration change if required)
 - Post-conditions (computing node health state set to valid new state value)
- Mode State *is represented by:*
 - Value (one of (set of predefined mode states))
 - Accuracy (discrete)
 - Range (set of predefined mode states)
 - Timeliness (permanent)
 - Dependency (none)
 - Integrity Constraints (if health state failed mode state is “none”)
 - Manipulation Operators
 - Change computing node mode state due to computing node mode request *requiring:*
 - Pre-conditions (none)
 - Trigger (computing node mode request)

- Modification (updates computing node mode state value, initiates computing node configuration change if required)
 - Post-conditions (computing node mode state set to valid new state)
- Change computing node mode state due to integration area mode state *requiring*:
 - Pre-conditions (none)
 - Trigger (integration area mode state changed)
 - Modification (updates computing node mode state value, initiates computing node configuration change if required)
 - Post-conditions (computing node mode state set to valid new state)
- Configuration State *is represented by*:
 - Value (one of (set of predefined configurations))
 - Accuracy (discrete)
 - Range (set of predefined health states)
 - Timeliness (permanent)
 - Dependency (depends upon mode state, depends upon health state)
 - Integrity Constraints (if health state failed configuration state is “none”)
 - Manipulation Operators
 - Change computing node configuration state due to computing node health state *requiring*:
 - Pre-conditions (none)
 - Trigger (computing node health state changed)
 - Modification (updates computing node configuration state value, sends new configuration state to integration area management component)
 - Post-conditions (computing node configuration state set to valid new state)
 - Change computing node configuration state due to computing node mode state *requiring*:
 - Pre-conditions (none)
 - Trigger (computing node mode state changed)
 - Modification (updates computing node configuration state value, sends new configuration state to integration area management component)
 - Post-conditions (computing node configuration state set to valid new state)
 - Change computing node configuration state due to integration area configuration state *requiring*:
 - Pre-conditions (none)
 - Trigger (integration area configuration state changed)

- Modification (updates computing node configuration state value,
sends new configuration state to integration area management component)
- Post-conditions (computing node configuration state set to valid new state)

Even for this simple IMS example this data model is complex. Also, this model will be replicated across all computing nodes. The interactions will also be complex, but to understand this we need to develop the model for the integration area, as individual computing nodes only interact with their hierarchical management component.

Integration Area State Data Model

The integration area state data model is again generic and an instance of the model will exist in all integration area management components. In structure it is very similar to the computing node data model, but the details of the states updated and the consequential actions are integration area specific.

Integration Area State *is represented by:*

- Health State *is represented by:*
 - Value (one of (healthy, failed, (set of partial fail states)))
 - Accuracy (discrete)
 - Range (set of predefined health states)
 - Timeliness (permanent)
 - Dependency (none)
 - Integrity Constraints (none)
 - Manipulation Operators
 - Change integration area health state due to computing node health state *requiring:*
 - Pre-conditions (health state not “failed”)
 - Trigger (integration area computing node health state changed)
 - Modification (updates health state value,
sends new health state to system level management component,
initiates integration area configuration change if required)
 - Post-conditions (integration area health state set to valid new state value)
- Mode State *is represented by:*
 - Value (one of (set of predefined mode states))
 - Accuracy (discrete)
 - Range (set of predefined mode states)
 - Timeliness (permanent)
 - Dependency (none)
 - Integrity Constraints (if health state failed mode state is “none”)
 - Manipulation Operators

- Change integration area mode state due to integration area mode request *requiring*:
 - Pre-conditions (none)
 - Trigger (integration area mode request)
 - Modification (updates integration area mode state value,
sends new mode state to computing nodes management components in integration area,
initiates integration area configuration change if required)
 - Post-conditions (integration area mode state set to valid new state)
- Change integration area mode state due to system mode state *requiring*:
 - Pre-conditions (none)
 - Trigger (system mode state changed)
 - Modification (updates integration area mode state value,
sends new mode state to computing nodes management components in integration area,
initiates integration area configuration change if required)
 - Post-conditions (integration area mode state set to valid new state)
- Configuration State *is represented by*:
 - Value (one of (set of predefined configurations))
 - Accuracy (discrete)
 - Range (set of predefined health states)
 - Timeliness (permanent)
 - Dependency (depends upon mode state, depends upon health state)
 - Integrity Constraints (if health state failed configuration state is “none”)
 - Manipulation Operators
 - Change integration area configuration state due to integration area health state *requiring*:
 - Pre-conditions (none)
 - Trigger (integration area health state changed)
 - Modification (updates integration area configuration state value)
sends new configuration state to computing nodes management components in integration area,
sends new configuration state to system level management component)
 - Post-conditions (integration area configuration state set to valid new state)
 - Change integration area configuration state due to integration area mode state *requiring*:

- Pre-conditions (none)
- Trigger (integration area mode state changed)
- Modification (updates integration area configuration state value)
 - sends new configuration state to computing nodes management components in integration area,
 - sends new configuration state to system level management component)
- Post-conditions (integration area configuration state set to valid new state)
- Change integration area configuration state due to system configuration state *requiring*:
 - Pre-conditions (none)
 - Trigger (system configuration state changed)
 - Modification (updates integration area configuration state value)
 - sends new configuration state to computing nodes management components in integration area,
 - sends new configuration state to system level management component)
 - Post-conditions (integration area configuration state set to valid new state)
- Change integration area configuration state due to computing node configuration state *requiring*:
 - Pre-conditions (none)
 - Trigger (computing node configuration state changed)
 - Modification (updates integration area configuration state value,
 - sends new configuration state to computing nodes management components in integration area,
 - sends new configuration state to system level management component)
 - Post-conditions (integration area configuration state set to valid new state)

System Level State Data Model

Finally, in order to complete the definition we need the system level state data model. Again this follows a similar pattern to the other models. This data model will only be instantiated once on the system level management component (this could be replicated to provide fault tolerance in real IMS architecture).

System Level State *is represented by:*

- Health State *is represented by:*
 - Value (one of (healthy, failed, (set of partial fail states)))
 - Accuracy (discrete)
 - Range (set of predefined health states)

- Timeliness (permanent)
- Dependency (none)
- Integrity Constraints (none)
- Manipulation Operators
 - Change system health state due to integration area health state *requiring:*
 - Pre-conditions (health state not “failed”)
 - Trigger (integration area health state changed)
 - Modification (updates system health state value, initiates system level configuration change if required)
 - Post-conditions (system health state set to valid new state value)
- Mode State *is represented by:*
 - Value (one of (set of predefined mode states))
 - Accuracy (discrete)
 - Range (set of predefined mode states)
 - Timeliness (permanent)
 - Dependency (none)
 - Integrity Constraints (if health state failed mode state is “none”)
 - Manipulation Operators
 - Change system mode state due to system level mode request *requiring:*
 - Pre-conditions (none)
 - Trigger (system level mode request)
 - Modification (updates system mode state value, sends new system level mode state to integration area management components, initiates system level configuration change if required)
 - Post-conditions (system mode state set to valid new state)
- Configuration State *is represented by:*
 - Value (one of (set of predefined configurations))
 - Accuracy (discrete)
 - Range (set of predefined health states)
 - Timeliness (permanent)
 - Dependency (depends upon mode state, depends upon health state)
 - Integrity Constraints (if health state failed configuration state is “none”)
 - Manipulation Operators
 - Change system configuration state due to system health state *requiring:*
 - Pre-conditions (none)
 - Trigger (system health state changed)

- Modification (updates system configuration state value)
 - sends new configuration state to integration area management components)
- Post-conditions (system configuration state set to valid new state)
- Change system configuration state due to system mode state *requiring*:
 - Pre-conditions (none)
 - Trigger (system mode state changed)
 - Modification (updates system configuration state value)
 - sends new configuration state to integration area management components)
 - Post-conditions (system configuration state set to valid new state)
- Change system configuration state due to integration area configuration state *requiring*:
 - Pre-conditions (none)
 - Trigger (integration area configuration state changed)
 - Modification (updates system configuration state value,
 - sends new configuration state to integration area management components)
 - Post-conditions (system configuration state set to valid new state)

These data model definitions form the basis of the overall system state model. We can represent the system using instantiations of these models and the interactions defined by the manipulation operators. This results in a model of the same pattern as that in Figure 21. Indeed, this earlier model is an abstract version of the interacting data models. This model is shown below. The model shown is less complex than the previous complete IMS state model as the internal interactions for each system management component are now contained within the data model instance for that component. As these are now internal interactions we can assume that they are not subject to some of the problems that cause our safety concerns. Specifically these internal interactions are not subject to time delays across the communications medium. We can therefore assume for our model that all required internal interactions are handled as part of a single action, or transaction, on the internal state of the data for that component.

In this data model representation we have also used the interactions as specified in the data models. This means that the interactions use the state data itself to inform other system components of the new state of the system. This does in fact change the system behaviour. In the previous model a management component was required to decide upon its new state and then decide what requests to make of other system components. For example, the system level management component would change its state and then was responsible for raising requests to the integration areas to move to a specified integration area state. This requires the system level manager to have access to its own state information from the blueprints **and** to have access to integration area state information from the blueprints. This is not good encapsulation, where the data for a specific node is contained only at that node.

Our new model now only requires the system level management component to have access to its own moding information from the blueprints as it does not have to make decisions about what information to send to other components. The integration area only requires a list of system level configurations to use to combine with its own moding information to make its own decisions.

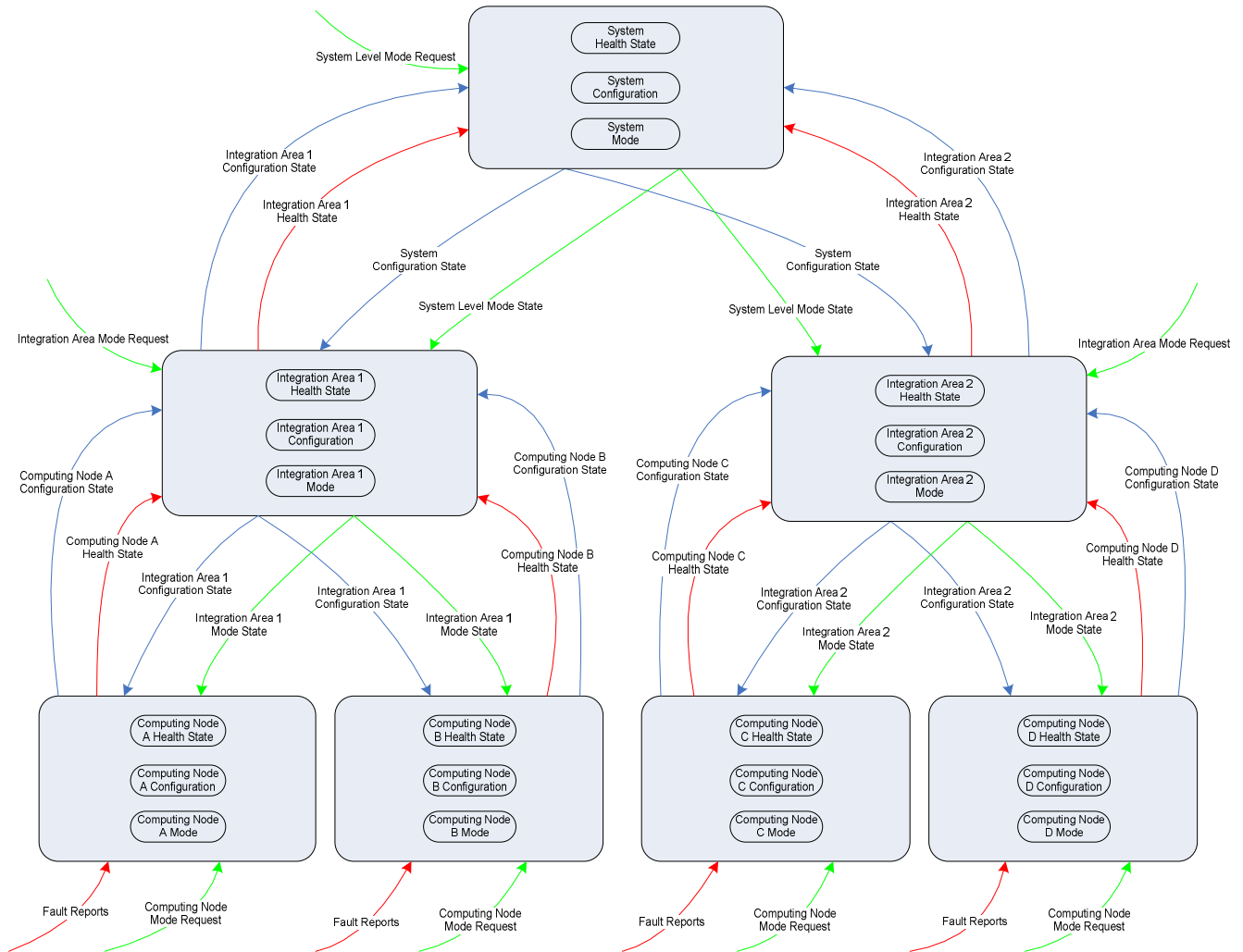


Figure 22 - IMS State Data Model

Our new interaction model only requires the passing of state data around the system, but these interactions are still subject to the distributed systems issues described in the review earlier. Specifically the false assumptions discussed by Van Streen must still be overcome [2]. If we assess which of these false assumptions cause problems with our data model there are a number that impact directly on our problem:

- Latency is zero – in any network the latency is never zero. In a real-time system the nature of the real-time requirement will determine how much latency is acceptable across any communications path. In an IMS distributed architecture the acceptable latency is a significant problem due to the distributed nature of the system state and the distributed points where the system state can change. Some of the IMS communication paths will actually be handled within a single site, as instances of integration and system level

management components will reside on computing nodes that already contain computing node management components. However, communication paths to the rest of the system will be over the communications media chosen for the IMS implementation.

- Bandwidth is infinite – again, in any network bandwidth is never infinite and this is also true of the IMS architecture. However, IMS does employ a point-to-point communications protocol which can use multiple communications paths to route messages. This may reduce the impact of bandwidth problems but it could, under some circumstances cause problems. This is discussed below under network topology.
- Transport cost is zero – the transport cost is measured in terms of both the added time taken for the data to be transported and any extra data required in the message to enable the network to route the message, such as headers used for addressing information.
- The topology does not change – most communications in real-time systems occur over statically defined networks. However, IMS allows the use of point-to-point networks employing multiple links. This type of network can be used to allow communications links to be both re-routed to overcome fault conditions, and routed via different paths to overcome bandwidth restrictions. This can vary the routes taken by messages within the system, affecting both the transport delay and possibly the sequencing of messages within the system. Both of these outcomes will cause safety problems in the IMS system management state model.

We have therefore identified a number of safety issues based upon this data model and the actions around its use that need to be assessed. This will be discussed in the evaluation section below. However, the data model does provide a more formal specification of the data within the system, showing not only the data items, but also the constraints and boundaries on those items and the modifications possible for the items. In fact, although the state data model does not make best use of the data model format, the format provides a rigorous software language independent way to structure the data in any system.

IMS Blueprint Data

The blueprint data is significantly different to the state data model described above. The blueprint is a large data file containing static configuration data for each processor node in the system. As such the complete data model would be too large to detail fully in this document. Instead we will provide the information model at a more abstract level and then use this as an example to show how the blueprint data model would be constructed.

Information and Data Model

The information contained within the blueprint provides all data required to configure each of the IMS processing nodes within a given IMS architecture. The RTBP includes the following information:

- The run-time requirements for all application software in terms of schedule, processing, communications, storage, moding and (allowable) levels of degradation in functionality or performance.

- Software to hardware mapping tables (generated by the off line allocation and / or scheduling tools) required to describe all normal and degraded configurations of the system.
- The capability of all hardware resources in terms of processing, communications, storage and fault modes.

This blueprint information is captured from the design and integration data defined off-line for the applications and the IMS system components. At this stage all data for all processing nodes exists as a single large database. However, in order to provide the required information to the run-time system the blueprint generation tool is used to produce a set of run-time blueprint files, each targeted at a particular computing node within the IMS system. Each of these files contains all of the relevant configuration information for that node throughout the operation of the system.

When we assess the safety aspects of the data in the blueprint, it is these run-time files that must be correct in order for the system to operate. Therefore, in defining the information and data models, it is the run-time information that must be used. In fact, both the information model and the data model for the blueprint represent a single instance of the run-time blueprint file. A single instance is all that we require as all IMS computing nodes are generic and therefore must use the same data structures. When the model is populated the content will obviously be computing node specific, but each must still be of the same form.

The information required in order to provide this set of configurations for the IMS system is comprehensive. The information model for the blueprint is defined below.

- **Blueprint** is represented by:
 - Set of configurations *represented by set of*:
 - Configuration identifier
 - Set of application processes *represented by set of*:
 - Application process *represented by*:
 - Process identifier
 - Process resource requirements
 - Set of application communication links *represented by set of*:
 - Virtual channel *represented by*:
 - Virtual channel identifier
 - Intra site resources *represented by*:
 - Memory map
 - Inter site resources *represented by*:
 - Transfer connection *represented by*:
 - Transfer connection identifier
 - Transfer connection resources *represented by*:
 - Memory map
 - Set of system management components *represented by set of*:
 - System management process *represented by*:
 - Process identifier
 - Process resource requirements
 - Set of system management communications links *represented by set of*:

- Virtual channel *represented by*:
 - Virtual channel identifier
 - Intra site resources *represented by*:
 - Memory map
 - Inter site resources *represented by*:
 - Transfer connection *represented by*:
 - Transfer connection identifier
 - Transfer connection resources *represented by*:
 - Memory map
 - Scheduling information *represented by*:
 - Scheduling table
- Set of faults and valid transitions to be handled *represented by set of*:
 - Fault identifier
 - Fault transition
- Set of modes and valid transitions to be handled *represented by set of*:
 - Mode identifier
 - Mode transition
- Set of configurations and valid transitions to be handled *represented by set of*:
 - Configuration identifier
 - Configuration transition
- Initial Mode
- Initial Configuration

This set of information for the blueprint whilst comprehensively detailing the major area of the blueprint could be broken down to show more detail in some areas. Also, if we were to then take this blueprint information and populate it with real blueprint data a significantly large structure would be produced. This is due to the nested nature of the structure, where multiple levels of information need to be provided for each node within each configuration. In order to take this information model and produce the data model it would be necessary to expand this structure to show all component data items. The nested fields could be handled without further expansion, but each of the major data components, such as the details of the resource needs and details of each transition, would need to be shown in full detail in order to complete the specific requirements of a data model. The resultant definition would be too large to include in this document, and the principle of the data model has been shown using the distributed system state model. It is in the detail of the distributed state data model where many of the safety issues lie, whereas the blueprint safety issues are centred on the validation of the content. Therefore, the blueprint data model is not expanded in this document.

Evaluation of IMS Data Safety Process

In the previous sections we have proposed a process for data safety in IMS systems, outline safety argument modules for use of certain IMS structures, and discussed and developed information and data models for the system state data and the blueprints.

We now need to evaluate these proposals. In order to do this the safety process will be investigated to understand how each of the steps helps us overcome the safety problem. We will also examine the requirements for safety evidence provided by the proposed safety argument modules to understand if it is possible to generate the required evidence.

Our proposed process contained the following steps:

Data Safety Process for IMS

1. Define safety argument structure for IMS data safety
2. Define data lifecycle model for IMS data
3. Identify data model for IMS data structures
4. Justify how this model fits properties of interest
5. Identify safety issues with this data model (hazards and risks)
6. Identify possible mitigation strategies for these safety issues
7. Identify data requirements
8. Populate data models to produce data specification
9. Validate data against system design and ensure traceability of data to requirements
10. Check list of hazards against populated model
 - a. Have the hazards associated with the data been identified?
 - b. Have all the risks associated with the data been identified?
 - c. Have all hazards associated with the data been mitigated?
11. Generate safety evidence for each element in safety argument (some of the evidence may be generic for all IMS systems)

Steps 1 to 3 have been completed for our proposed solution. The steps 4 to 11 actually form the safety evaluation process that would be needed for an IMS implementation. Some of these steps will therefore be used as our assessment of steps 1 to 3. Steps 4, 5 And 6 can be used to generically assess the proposed data models. Steps 7 to 10 deal with the specifics of a particular implementation of the system and are therefore not required to assess our proposed generic solution. Step 11 can be used to ensure that all evidence required for our proposed safety argument structure can be provided. Finally we will determine if the process itself is complete and sufficient.

Evaluation of the data model

The evaluation of the data model will concentrate of the distributed system state data model. As previously discussed, this model is more complex due to the real-time distributed structure and therefore presents a more difficult challenge when assessing its safety.

There are a number of aspects of the data model that need to be evaluated, but in order to fully understand the behaviour of the real-time distributed model we must first assess how changes in local state data affect the global state. In order to do this we will examine a series of change scenarios and describe the sequences of events that occur within the model. We will use a simplified version of UML sequence diagrams to explain these scenarios. The states and transactions used a directly derived from the data model proposed earlier.

After performing this analysis of the structure of the data we can then understand how we may need to apply constraints to the system or limitations to the model we have derived in order to overcome hazardous states.

State Change Scenarios and Sequence Diagrams

In order to understand the detail of how transactions occur within the global state data model it is necessary to choose scenarios that highlight the specific steps between states.

Firstly we will consider a simple health state change. In this first example we are still using the IMS system of interest, but only two of the computing nodes and one of the integration areas are shown for clarity. The sequence shows the messages required to change the system state when a failure occurs on computing element A.

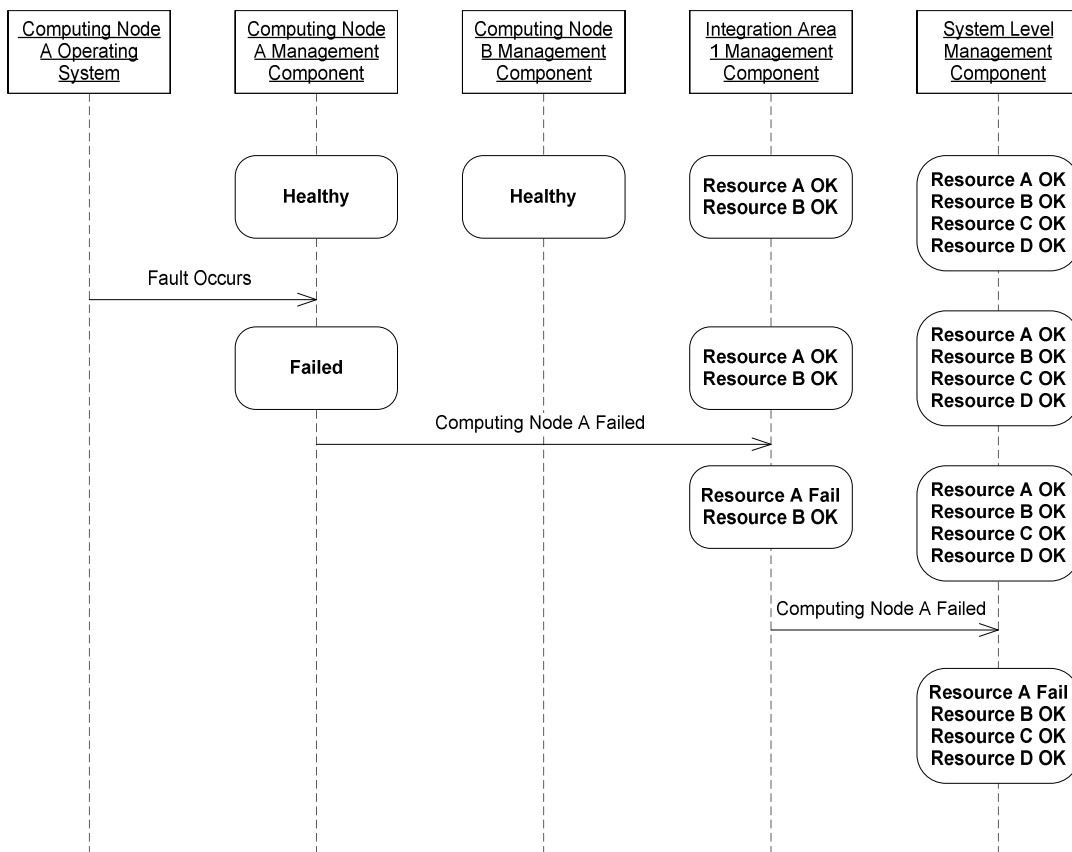


Figure 23 - Simple Health State Change Sequence Diagram

All that is required is to ensure that the integration area and system level manager are aware of the state change. This takes two messages across the system in order to modify the system state and make it consistent. The diagram also includes repeated states to make it clear that even with this simple change there are inconsistencies in the system. After the fault occurs there is a period whilst computing node A changes state locally where both the integration area and the system level do not have correct state information. Even after the failure information

is sent to the integration area the system level is still inconsistent. However, as this information has not been used during the period the inconsistency existed there is no safety problem. Note that two messages are required to pass the fail state information for the computing node to the system level management component. This is due to the rigid hierarchy imposed by the IMS structural model.

If we now consider how this change may affect the configuration of the system we can see that the sequence of events is more complex. We will include the original state change as it is the combination of these chains of events that show us how the safety may be compromised but we have added the configuration state to the health state information. In this example the system management component will change the configuration of the system in order to overcome the fault condition.

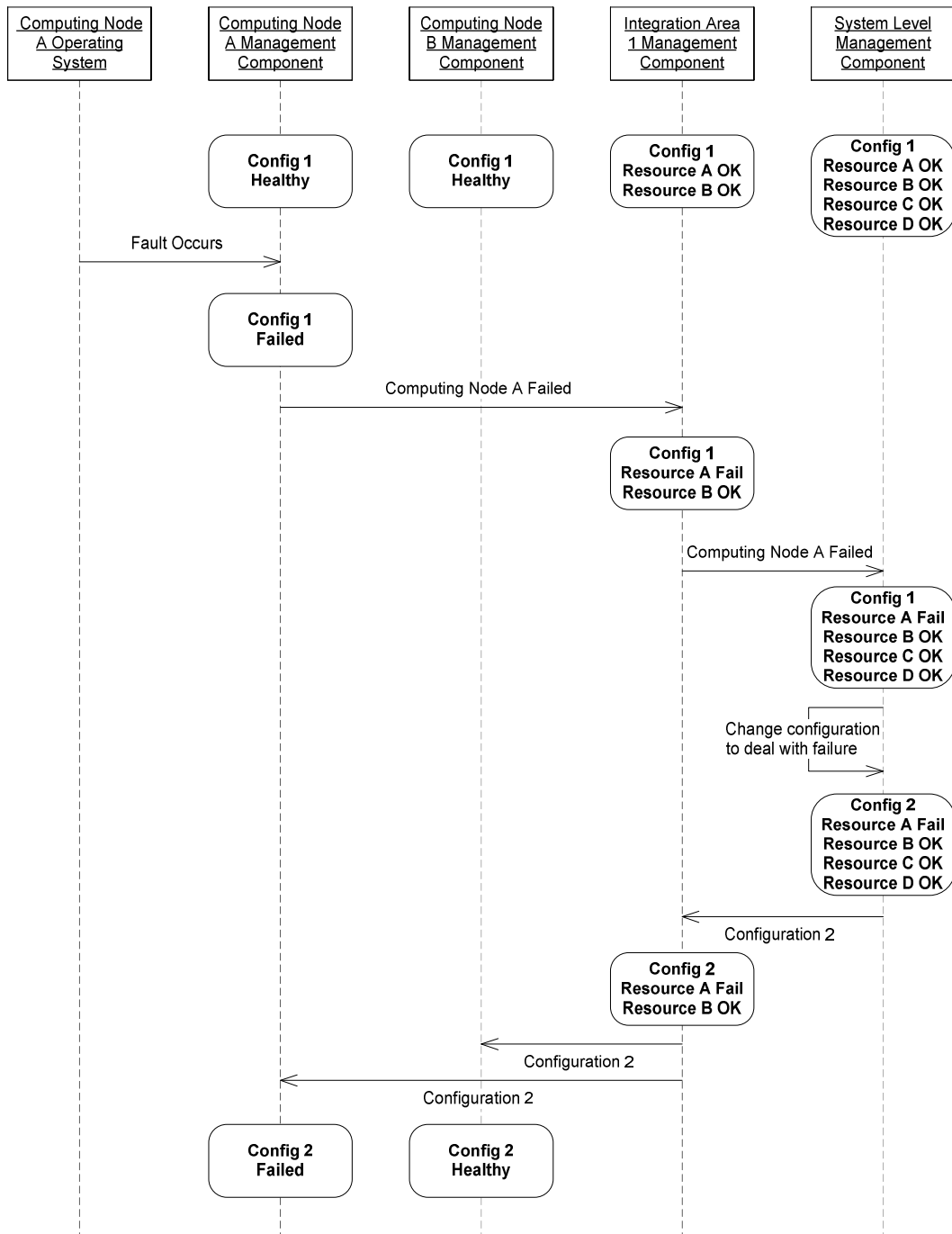


Figure 24 - Health and Configuration State Change Sequence Diagram

In this sequence we can see that in order for the system level management component to change the configuration of the system we need another sequence of events as complex as the initial health state change. We have assumed that we still send the updated configuration state to computing node A, even though it is failed and therefore unable to be used as a resource. So we now have a sequence of messages that need to flow around the system in order to return the system to a valid operational state, and we can see that during this process a number of inconsistencies exist, such as the delay in fault notification and the invalid state on computing node B after the failure until the configuration request arrives. There is an identifiable safety issue due to the continuation of computing node B in configuration 1, as this may communicate with computing node A. In the period between the failure of A and the reconfiguration of B another failure condition may occur due to lack of communication with A.

However, there is a more fundamental safety issue that can occur in such a situation. If we now consider the same sequence of events, but with another local state change, we can see how a complex safety problem can arise.

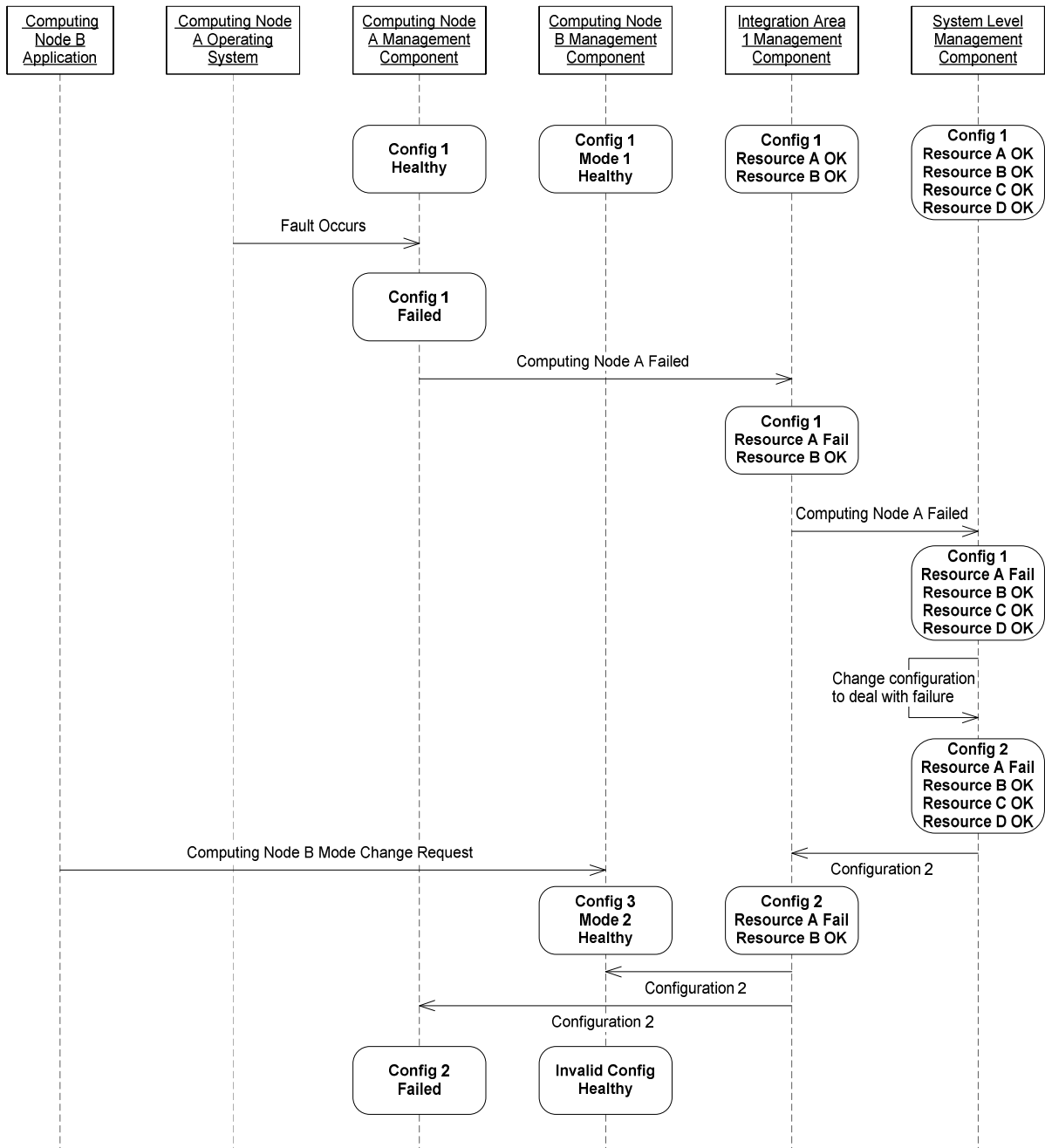


Figure 25 - Invalid State Change Sequence Diagram

The addition in this example is a local mode change request of computing node B. Due to the point at which this occurs the configuration of B is modified before it receives the configuration2 request. We can assume for this example that a configuration change from 3 to 2 on B is an invalid state transition. So, with a relatively simple example of the state changes in an IMS system we can see that the complex distributed nature of the system will allow significant safety issues to occur. It is possible to produce numerous sequences and scenarios that would cause this state contention to exist, but the mechanism that causes the problem would be the same. The issue lies in the ability to update the components of a global state independently in multiple locations where delays occur in the transmission of state data around the system.

So we need to identify how these safety concerns can be mitigated and to do this we need to constrain the possible sequences of events to prevent this contention. We will use this last scenario, containing an invalid state change, in order to assess the constraints applied.

Possible System State Safety Mechanisms

There are a number of possible mechanisms that could be used to constrain the state transition behaviour but we must ensure that the real-time performance of the system is not compromised by any constraints we impose.

System Level Control Over All State Changes

The first possibility is that we constrain all system state changes by ensuring that they have to be endorsed by the system level management component. If a state change request occurs at the integration area or computing node, the node must communicate the request to the system level and then wait for the response before the state change occurs. The sequence diagram, Figure 26 below shows the same series of events but with this new rule applied.

The diagram shows that in this case both the fault and the mode change request are handled correctly and the resulting configuration 4 is chosen by the system level management component to deal with both the mode change request and the fault condition. This would seem to be a satisfactory result. However, there are a number of outstanding issues that still cause problems.

The first of these issues is concerned with the exact sequencing of requests. If the sequence occurs as shown in Figure 26 then the state is resolved correctly. However, if the node B mode change request had occurred after the integration management component had sent out the configuration 2 message then a different set of events would have occurred. It may still resolve to a valid state, but due to the possible variability in the timing of these messages there is still possible contention within the architecture.

The second issue is that of delays. It can be seen from the sequence diagram that the sequence of events is now longer. This extended sequence may seem insignificant, but this is a simple example and more complex sequences could lead to long delays. Also, even a relatively short delay may not be acceptable due to the need to maintain the real-time deadlines of the system. Before the constraint was applied computing node B could change mode locally and continue processing. This has now been prevented, causing node B to wait for a response from the system level. It may not be acceptable to enforce significant wait times on computing nodes

to allow the complex interactions with the system level management component to take place.

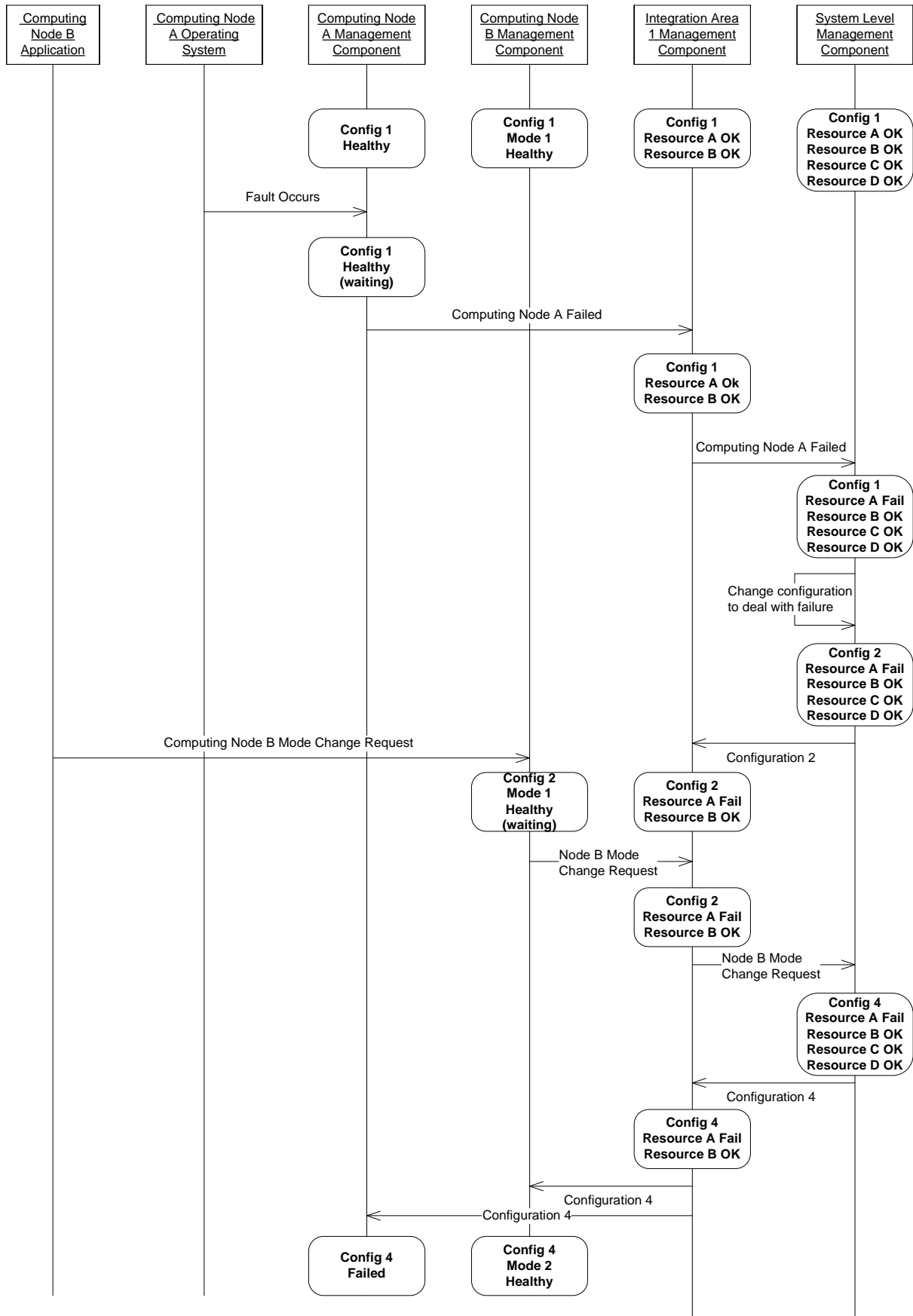


Figure 26 - System State Transitions Constrained Through System Level

Remove Integration Areas

Another possibility to remove some of the overhead is to remove the integration area management components from the architecture. One of the visible issues with the previous scenario was the delay due to the multiple communications required before a message could get from a computing node to the system level. The scenario would be simplified by removing these intermediate management stages.

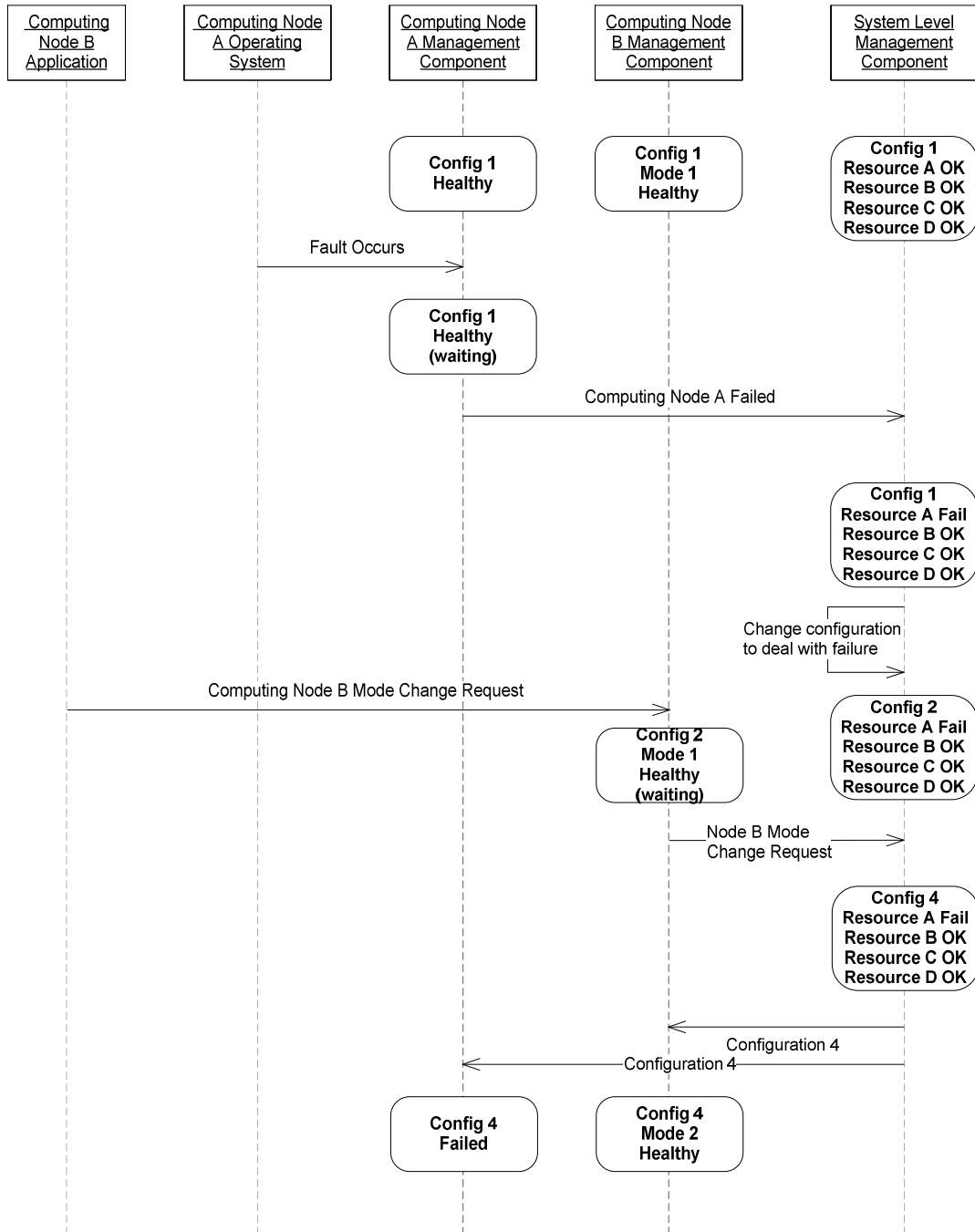


Figure 27 - System State Change – No Integration Areas

The scenario has indeed been simplified, removing much of the complex interactions required in the earlier example. It would seem that the system has been improved by this modification. However, although the complexity is reduced in this simple system a more complex IMS system may not achieve the same benefits. The integration area concept was introduced into IMS in order to remove a bottleneck at the system level management component. Our scenario only requires interaction between two computing nodes and the system level. If we had a large number of computing nodes

that all needed direct access to the system level management component all we have achieved is to create a new form of contention. The system management component may quickly reach a point where it cannot respond to all of the state change requests from every computing node. In this situation we would again fall into a consistency problem where multiple requests were arriving before the response to the first was completed.

Remove Hierarchical Components

It may be possible to remove the hierarchy completely. In such an IMS architecture all computing nodes would be responsible not only for local state changes, but also global state. In order to come to some form of consensus on the global state some form of voting would be required. However, this scenario has all the disadvantages of the previous “no integration areas” case, including the large communications overhead, but with the added complication of voting. Voting is an acceptable way to control a safety related system, but in this case it would lead to those computing nodes that do not agree with the consensus being forced to change to the majority state, or we would create the contention between local state again.

Another variation upon this reduction in hierarchy approach is to remove the system level state, and for the integration area management components to become the top layer in the architecture. Again this suffers from the problems of the previous scenarios. We now retain the localised control over groups of modules, but we have moved our voting mechanisms, and the contention they create, to the integration area level. In a large system there may be many integration areas, adding to the communications overhead.

Distributed Coordinated Atomic Actions

As we have seen the approaches which involve constraints to the standard architecture are not effective in overcoming the consistency and contention issues involved with the IMS distributed real-time system. So, in order to find a solution we need to examine some of the concepts proposed in the review of data consistency research for distributed systems. Specifically we will assess the concept of coordinated atomic actions (CAA) and determined if this can be used to maintain the consistency of our IMS architectural components.

A coordinated atomic action is a bounded set of actions undertaken by a number of execution elements that is designed to complete a specified task. During the action the participating execution threads may influence external resources, but they cannot be interrupted by other threads. When the action completes it is either successful, or all state variables subject to the action will remain / be reset to their state previous to the action commencing.

However, in order to use CAAs in our IMS system we must extend the scope and meaning of the concept. CAAs specifically coordinate the actions of threads in a multithreaded system. The model expects the availability of shared local resources in order to allow synchronisation of the interacting threads and access to required shared data. In our distributed environment this shared resource does not exist. So, whilst we require the bounded nature of CAA interactions, we need to be able to set this bound around a set of concurrent interacting processing nodes. In order to do this an extended CAA concept is proposed, the Distributed Coordinated Atomic Action (DCAA). This can be defined as a bounded set of interacting distributed software components that communicate via message passing across a network. Also, due to the lack of shared common resources, the ability for a CAA to influence

shared external resource could cause contention in a DCAA. Therefore the DCAA concept is limited, and does not allow access to external resources during the DCAA operation. The other CAA properties hold, specifically the output of the CAA must be a successful transition (to a valid state) or the original input state is retained.

So this mechanism gives us a process by which IMS management components can interact in a closed environment to perform some update to a state. If the state is consistent before the update, the mechanism will ensure that the system state remains consistent, as the only possible outcomes are the successful updating of the state or a continuation of the consistent state that existed before the update. This mechanism will overcome the contention and consistency problems that affect our other suggested solutions.

The problem with this concept is, of course, the problem of implementing such a mechanism. Again, in order to describe a proposed implementation using the IMS management components we will define a scenario with a sequence of transitions.

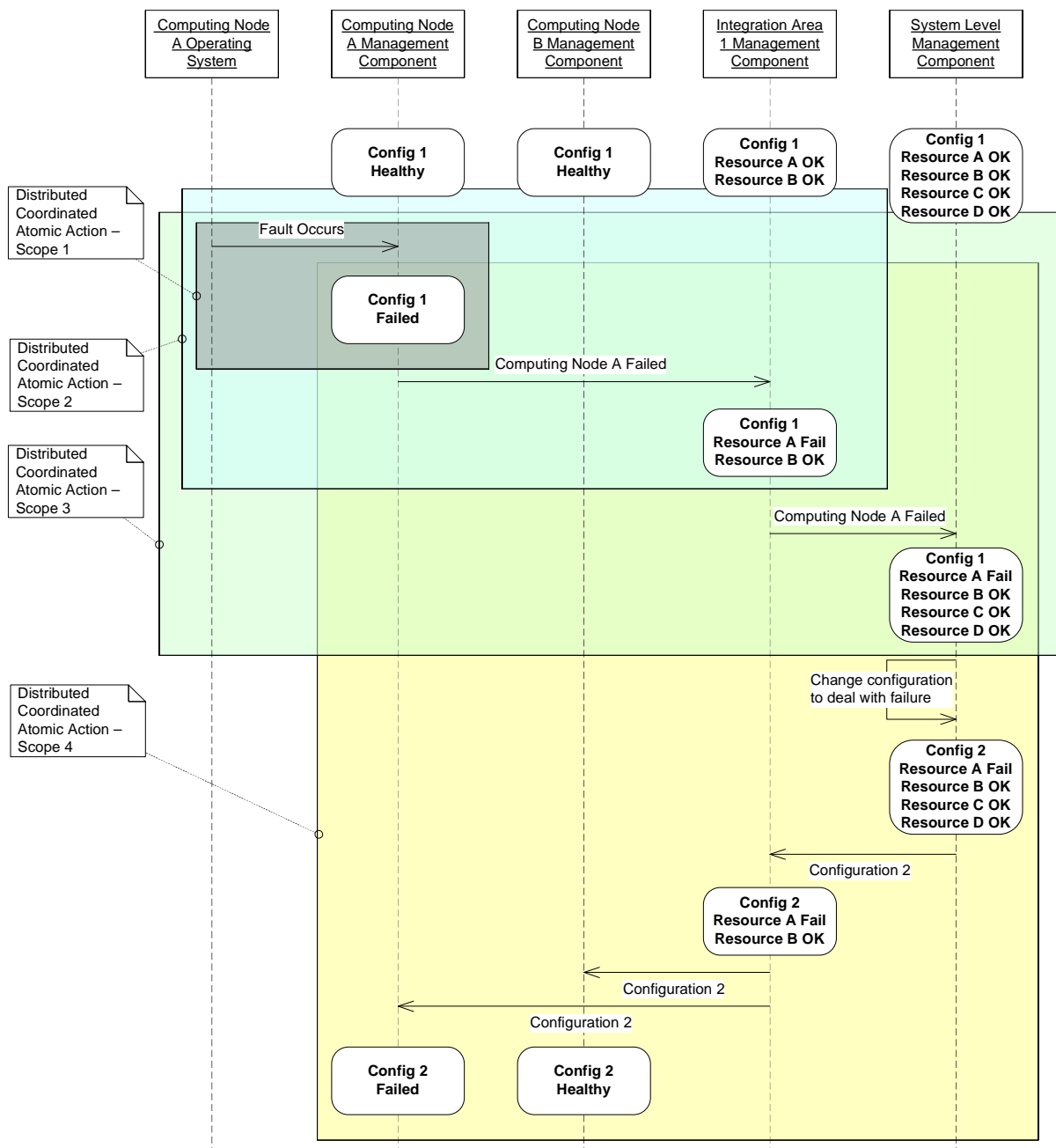


Figure 28 - Scope Choices for Distributed Coordinated Atomic Actions

If we examine how the DCAA concept could encapsulate the same state change used in the previous example the initial diagram does not seem significantly different. A decision has to be made for the scope of the DCAA. There are a number of solutions that could be considered:

1. A DCAA for each update action for a specific management component. This would limit the DCAA to a single management component, reducing the time the rest of the system is blocked from access to the DCAA resources, but would limit the advantages of the DCAA concept, as the complex interactions between components would be outside the scope of the DCAA.
2. A DCAA for each interaction between a management component and its parent. This would prevent contention at a single hierarchy level, but would not prevent the contention we saw in our earlier example as this contention was due to simultaneous interactions at different levels and nodes within the system. Limiting the DCAA to a specific interaction between two nodes would not prevent this.
3. A DCAA for each state change, concluding when the single state change has been recognised across all system components. This would prevent contention during the process of recording a state change across the system, but our earlier example failed due to contention in the return of the response to the original state change; i.e. the failure state change would be a single DCAA but the response to change the configuration would be a separate DCAA. There is still the possibility of contention between the original fault state change and a new valid state being applied across the system.
4. A DCAA for each state change, including all sub-actions required to complete the response back to the originating system. This ensures that the entire process to move from a failure to a valid new state is not interrupted and contention is prevented. However, the rest of the system is blocked from access to the involved management components for the longest period.

In order to understand the impact of these different scopes Figure 28 presents the earlier fault / configuration change example with the various choices highlighted. This shows not only the components contained within the DCAA, but also the relative periods that other nodes will be blocked.

We must balance the possible interference between concurrent transactions with the delay to transaction events. However, in making this choice it is important to ensure that we provide a solution to the contention problem. For this reason the proposed solution is to use DCAAs which encompass the end-to-end state changes described in 4 above. This gives the best opportunity to remove the contention. It is also necessary to assess what will occur when a second state change is requested during the period of the DCAA. This is shown in Figure 29.

The second state change request cannot affect the components involved in the ongoing DCAA. Therefore, this second state change is blocked until the DCAA completes. At this point the global state of the system has been set to a valid consistent state (either a new valid state, or returned to the previous valid state). Since we are now once again in a valid system state we can now accept the next state change request and a new DCAA begins (this is only partially shown in Figure 29).

It is therefore clear that by constraining components to use DCAAs we can protect the system from contention and invalid states. However, we have blocked the second state change until the first completes. This is a necessary effect of the consistency resolution and must be balanced against the predictable behaviour the DCAA provides.

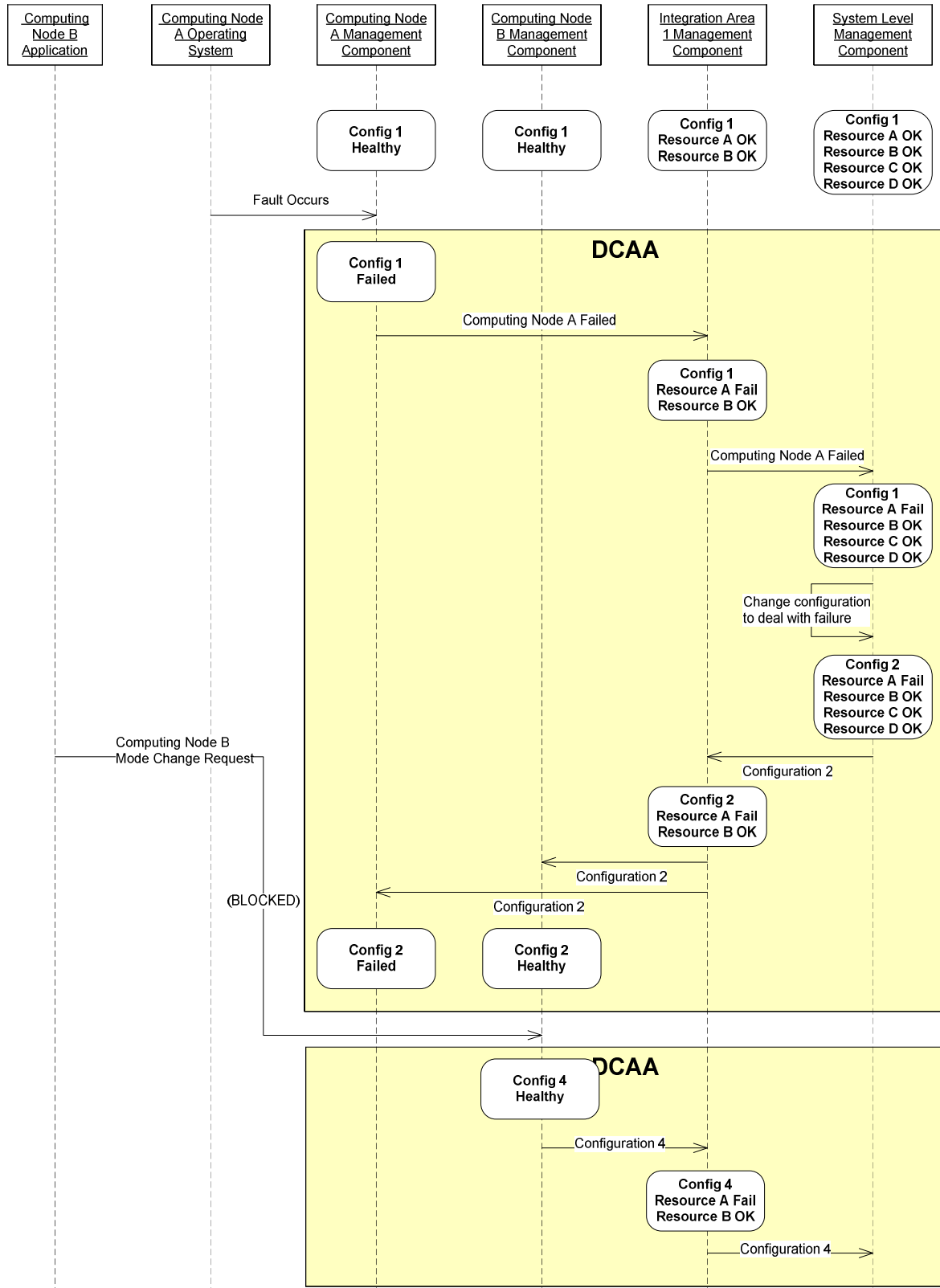


Figure 29 - Multiple State Changes Using DCAAs

So, we can ensure the correct system behaviour using DCAAs, but we still need to understand how to implement this concept across a distributed system. The series of steps that we previously defined for DCAA implementation must be understood in the light of the IMS management components. This list of implementation steps for CAAs were taken from Schaefer [40] but must now be assessed for DCAAs:

1. Synchronisation – All participating processes must be synchronised at the start of the DCAA. For the IMS system management components this synchronisation is the most difficult aspect of DCAAs as it requires messages to be sent across the network between the participating nodes. In order that this is done without the possibility of further contention issues this synchronisation must be performed using a contention resistant algorithm. This will be explored below.
2. Input Validation – All input parameters to the DCAA must be validated upon entry. For IMS this requires that all interacting management components have validated their state information as the DCAA begins. An important pre-condition for the DCAA is a valid system state at entry.
3. Normal Operation - Perform the normal operational functions in order. In IMS this is the actual state change process.
4. Exception Handling – Whilst performing normal operation handle any failure conditions locally within the DCAA. If any invalid states are reached during the operation of the DCAA the exception handling should either deal with this and therefore choose a new valid output state or return the system state to the valid entry state. The DCAA must always exit with a valid system state.
5. Complete Normal Operation – Wait for the operation to complete or timeout.
6. Output – set the calculated output parameters. Ensure the exit state is a valid system state.
7. Synchronisation – ensure all participating processes agree the output conditions, including success or failure. See the entry synchronisation discussion above.
8. External Objects – complete any transactions on external objects, ensuring no intermediate results are visible externally, and ensuring any fault conditions are passed on as required. There should be no requirement for external objects to be modified in the IMS DCAA model.

Therefore the DCAA concept holds as long as the initial and final synchronisation process can be implemented. The proposed solution to this synchronisation problem is the use of DCAA Locks.

As discussed in the research review, the concept of locks is used in distributed data access where multiple users need to update or read from a single data item. Before the data item can be accessed the user must first request a “lock” to be applied to the item. If the item is already locked by another user, the first user is blocked from access until the lock is released. This mechanism can be modified to provide synchronisation for our DCAA.

When a management component needs to update the system state it needs to request a DCAA lock for the system. This lock is owned by the highest level entity that is required to be part of the DCAA. In our earlier state change example this would be the system level management component. However, the lock is also present on all management components. The node requesting the state change must request the lock from all participating components using a network broadcast

message. This by-passes the normal hierarchical message paths to ensure all participants receive the message at one time. Even so, due to network delays, some nodes may not receive the request at exactly the same time, providing a possible cause of contention. To overcome this, and to ensure the lock is free, the initiating site requires a positive lock response from all participants before beginning the DCAA process. If any response is negative, the lock has failed and the initiating component must ensure all locks are released before making another attempt. At the conclusion of the DCAA the locks must be released. This gives a robust lock mechanism to ensure the DCAA is synchronised upon entry, but it does also increase the time taken to perform the DCAA. This is seen as a necessary trade-off against predictable behaviour.

This new concept of DCAAs, including locks requires thorough evaluation against IMS scenario's to ensure that the concept holds. The following test cases for IMS system management have been assessed with the DCAA concept (the first of these is our original example scenario) using sequence diagrams:

1. Fault condition at the computing node – this is our original test scenario. We have seen that the DCAA concept prevents contention.
2. Mode change request at the computing node – this scenario is very similar to scenario 1. The DCAA concept performs in the same manner. This included injection of state change requests at the integration and system level.
3. Mode change request at the integration area – this scenario is based around the injection of a request at the integration area. Independent requests were injected at the computing node level and at the integration area level.
4. Mode change request at the system level – this included injection of requests at all levels.
5. Conflicting mode change and fault condition in a single integration area – this scenario requires multiple configuration changes to ensure both conditions are handled.
6. Conflicting mode change and fault condition in separate integration areas affecting only integration area configurations – this scenario is interesting as it allows two DCAAs to run simultaneously. Not all configuration changes require changes to the system level configuration. If this is not required these changes can occur concurrently.
7. Conflicting mode change and fault condition in separate integration areas affecting system level configurations – this scenario uses the previous test case but enforces the state change at the system level. The DCAAs are then forced to run in a serial manner.

In all cases evaluations included the use of independent state change requests at various stages in order to ensure that the DCAA lock mechanism and the DCAA concept prevented contention. The sequence diagrams are not included in this report as they are of a significant size. This series of scenario tests did highlight the following issues:

The model of states within the IMS system has a direct relation to the required scope of the DCAA for a scenario. Test cases 6 and 7 highlight this issue. If we use a single configuration identifier across all nodes in the system, i.e. all configurations are unique across the whole system; all state change transactions and DCAAs must involve all levels of the system. In fact, if this suggestion is taken to the extreme each state change requires all computing nodes to be updated and therefore all computing

nodes must participate in all DCAAs. If, as in scenario 6, configurations are hierarchical, with the valid hierarchy defined in the blueprint, then we can allow concurrent state updates to occur in different areas of the system. This also means that the scope of the DCAA can be limited to the smallest required sub-set of processing modules. This increases performance due to the concurrency available and the reduction in blocking behaviour.

This model of hierarchical states is therefore proposed for use in IMS distributed state data.

Evaluation of the Safety Argument Structure

The other major areas for evaluation are the safety argument modules developed for our IMS data. Whilst the arguments hold for the intended data use, in order for the arguments to be useful it must be possible to provide the safety evidence in order to support the arguments. If this is not possible the argument cannot be used to prove the data within the system is acceptably safe.

The three argument modules require evidence to be created to support a number of low level arguments; the leaves of the argument structure. These arguments will now be presented in turn and the necessary steps required to produce the evidence will be assessed. As part of this assessment the arguments used were evaluated against the real-world development processes used for IMS. The BAE Systems blueprint tool and development process have been used to provide a model for this work.

Blueprint Development Safety Argument Structure

- The blueprint tool was developed using an acceptably safe process

The processes used to develop the blueprint tool are outside the scope of this research. However, there is no reason to believe that an appropriate process could not be used to develop this tool. The processes used to create the BAE Systems IMS blueprint tool were subject to the standard coding practices enforced for system support software and these are deemed sufficient to allow the use of this tool as an input to IMS systems for Hawk and Tornado. Also, this assessment need only be performed once for each tool implementation. Therefore, the assumption is made that evidence can be provided to support this argument.

- The blueprint tool enforces IMS design rules

The evidence to support this argument must be based upon the design of the blueprint tool used for a specific IMS implementation. In order to verify that the blueprint tool enforces the IMS design rules, such as ensuring each computing node has a system management component assigned to it, we need to ensure the tool requirements capture the IMS design rules, and then verify the blueprint tool design against the requirements. It may also be necessary to perform a code walkthrough of the blueprint tool implementation in order to provide traceability of the implementation back to the requirements. These processes have been used during the development of the BAE Systems toolset. Also, the BAE Systems tool has been developed around a UML model of the blueprint data. This model allows the design rule enforcement to be more easily checked. Again, although this process may require

significant effort, it is not a novel assessment, and need only be performed once for each tool implementation.

- The blueprint tool enforces the blueprint run-time structure

The evidence to support this argument is the same assessment required for the previous argument. In order to verify that the blueprint tool enforces the IMS structure (the run-time file structure) we need to ensure the tool requirements capture the IMS structure, and then verify the blueprint tool design against the requirements. Again, it may be necessary to perform a code walkthrough of the blueprint tool implementation in order to provide traceability of the implementation back to the requirements, and although this process may require significant effort, it is not a novel assessment. These processes have been used during the development of the BAE Systems toolset. Again, the BAE Systems tool has been developed around a UML model of the blueprint data, allowing the structural enforcement to be more easily checked. It only needs to be performed once for each tool implementation.

- The blueprint tool output can be independently validated against the design data

This assessment is more complex as it requires a mapping of the design data structures used for the system development to the blueprint tool output format. This mapping will be ideally independent from the mapping process used by the tool to provide independent verification that the tool transformations are correct. The difficult issue is the definition of this mapping. This will be a significant task, but it does not require novel academic approaches. This mapping will need to be built once for each combination of design toolset and blueprint tool. This argument deals with the development of the tool, and therefore the use of this mapping does not form part of the evidence. This mapping is used to perform a check on the data produced below. In the BAE Systems toolset the UML model provides an intermediate stage that can be used to break down this problem into two steps.

- The blueprint run-time files are validated against the design data

Above we required a map to be developed of the blueprint tool output format against the design data. Once the mapping has been developed it can be used for each new blueprint created by the tool to perform this assessment. Again, although this process may require significant effort, it is not a novel assessment.

- The blueprint run-time files are validated against the IMS design rules

This assessment requires a checklist-based assessment of the blueprint tool output to ensure a set of design rules have been captured in the output data. We have already checked that the tool is designed to enforce these rules, but this checks the output of the tool for each blueprint to ensure there are no errors. As this assessment could be driven by checklists it may be possible to automate the process. Again, there is no reason to suggest this assessment is not possible. In fact work is underway within BAE Systems to provide some level of automated back-path checking of the data, structure and design rules.

- The blueprint run-time files are formatted correctly

This assessment requires a checklist-based assessment of the blueprint tool output to ensure the correct structure has been captured in the output data. In this respect it is the same format as the previous assessment. Therefore we can assume this assessment is feasible.

- The blueprint data is traceable to design requirements

In addition to validating the blueprint data against the design data, it is also necessary to ensure that the blueprint data can be traced back to the design requirements of the system. An example of this would be a communications link. Ensuring the blueprint description of this link matches a design data link specification is one step. Ensuring this link is required due to a design requirement for two applications to share data is a further validation step and adds more value to the safety argument. Again, although this process may require significant effort, it is not a novel assessment. It must be done for each blueprint created, although the basic checks will only be formulated once for each design.

Blueprint Use Safety Argument Structure

- The process used to install the blueprint run-time data is acceptably safe

This requires the assessment of the installation process for the data files into the run-time system. This will utilise the same process used to download software onto the operational system, and therefore the process will already be subject to stringent safety checks, including checksums for the downloaded elements, etc. Therefore, this process is not novel, and the assessment need only be performed once for each tool implementation.

- The blueprint run-time data is validated against the off-line tool output after installation

This validation process ensures the downloaded data is the same as the off-line original. Again, standard software techniques are available to perform these checks and therefore this is not a novel process. It will be required once for each blueprint download, but the techniques are usually automated and therefore this will not require significant extra effort.

- The run-time blueprint data is interpreted by the IMS software correctly

This assessment is complex. It requires assessment of the IMS software implementation to ensure it is designed to use the blueprint data correctly. The scope of this falls outside our research, but it is worth noting that this form of assessment will already be required for high integrity code such as the IMS software components, and therefore this assessment should be feasible to include in our safety argument.

Distributed System State Safety Argument Structure

- The initial distributed system state is valid

This initial system state is derived from the initial blueprint data. This blueprint data will have been assessed during the safety assessment of

the blueprint above. We can therefore assume that this assessment will be possible by use of the blueprint argument techniques described above.

- During operation all local states are valid

This requires the assessment of two separate items. The first is the list of local states provided by the blueprint. This needs to be checked against the design data to ensure only valid states are contained within the blueprint data for each computing node. Secondly, the software that implements state changes must be implemented correctly. This falls into the scope of the software safety assessment process, but it requires that only states from the valid list in the blueprint can be used.

- During operation all local states are consistent

This requires assessment of the consistency protocol used within the distributed system management hierarchy. This is therefore the safety assessment of the DCAA concept described earlier. This is a complex assessment, as it needs to ensure the real-time behaviour of the system is not compromised. This may require further academic research in order to show the DCAA and DCAA lock mechanisms are predictable.

- Local state transitions are valid

This requires the assessment of allowed state transitions provided by the blueprint. This needs to be checked against the design data to ensure only valid transitions are contained within the blueprint data for each computing node. Secondly, the software that implements state changes must be implemented correctly. This falls into the scope of the software safety assessment process.

- Global state transitions are valid

This is a complex problem as it requires an assessment across the blueprint data for all nodes to ensure global states are supported correctly by local state and sets of local states. Also the DCAA concept must work predictably in order to successfully transition between valid global states. Whilst the validation of combinations of local states is possible but difficult, the assessment of the DCAA model may require further modelling and assessment.

- During transitions global state remains consistent

This also requires assessment of the DCAA concept. Again, this is a complex assessment, and may require further academic research in order to show the DCAA and DCAA lock mechanisms are predictable.

Therefore, it is possible to provide evidence to support our safety argument in almost all cases by the use of current understood processes. The main exception to this is the assessment of the system management consistency mechanisms, coordinated atomic actions. Although this research has proposed a model for their use, the concept requires further investigation in order to provide the required level of evidence. However, before this evidence would be needed an implementation of the DCAA concept would have to be developed, and this development process would have to ensure that mechanism was robust.

In order to be sure that our proposed safety argument structure is robust we must also justify it against some of the data problems proposed in the literature review.

Specifically the areas of concern are:

- Validation of the data against its requirements – through the proposed process and as part of a number of arguments within the safety argument structure the proposed solution specifically ensures that both the structure of the tool and the data in each created blueprint is validated against the IMS design rules and the specific requirements in the design of the system using the IMS architecture. It is not possible to provide this validation in a generic sense, but the processes proposed ensure that it forms a key element of our overall safety process.
- Separation of the data lifecycle from the software lifecycle – the proposed solution defines a separate data process and this is the basis of a separate data lifecycle. This research has not attempted to cover the design process for an IMS system, but this is still an area where the software design and the blueprint data requirements are combined. In fact this is necessary as a large part of the IMS blueprint is actually an abstraction of application design. The blueprint data does not exist as a separate entity with its own requirements. It is information taken from the application and system requirements.
- Separation of the data safety argument from the software safety argument – the proposed safety argument modules perform this function. Although designed to be used within an overall modular safety case, together with the modules covering the software of the system, these safety case modules now require specific evidence for data safety to be produced.
- Definition of structure, syntax and semantics for the data model – the proposed data model for the system state data includes all of these properties, defined as the data model structure, the constraints and limits of the data items within the structure, and a set of modification operators to constrain how the data can be manipulated. Data models of this type provide a far more robust specification of the data, allowing for a more predictable use within the system, and a more straightforward approach to safety assessment due to the reduced variability.

The proposed safety argument modules have been evaluated against both our theoretical model of the blueprint development process and the real world example of the BAE Systems toolset. This assessment has confirmed that the proposed argument modules cover the safety concerns of the data models identified, and therefore provide a sufficient argument over the safety of both blueprints and distributed state data. The proposed modules are constructed in a manner that allows them to be used in a modular safety case approach as this is likely to be adopted for use in IMS and similar systems. We have also shown that it is possible to generate the required evidence to support these arguments, identifying where this sort of evidence has already been developed as part of the BAE Systems IMS Blueprint Toolset development.

Conclusions and Further Work

This research has attempted to derive a safety process to address the use of two diverse data models used within distributed data-driven real-time systems. It has used the example of IMS as the basis to derive this process and understand the safety implications that it presents.

There are many forms of data used within these systems and not all can be subjected to the same safety process.

We have identified the use of large structures of configuration data such as blueprints, which present us with safety problems mainly in the development phase. Having evaluated the safety issues with these large data files we have proposed a process by which we can assess their safe use. We have also suggested that it is possible to evaluate the safe use of such configuration data based upon the well understood lifecycle practices used for software. This process suggests the use of some techniques that have not been addressed in the evaluation. The identification of hazards and risks has been limited to those directly impacting the two areas being studied. A more comprehensive hazard analysis, such as a software HAZOP approach, may drive out other safety problems. Although this has not formed part of the research, it would be useful to complete a comprehensive assessment using such a technique in order to ensure no major hazards have been overlooked.

It is necessary to ensure this data lifecycle is supported independently of the software lifecycle due to the possible separation, in both time and location, between the development of the software and the data. It is quite often the case that the data for such IMS systems is created separately to the generic IMS system components, and will be updated as modifications are made to the system. If we consider other large configuration files, such as terrain databases, the development of the data is not tied to the system development or modification at all, giving even more reason to ensure the lifecycles are independent. Additionally to the overall process lifecycle we have defined a blueprint lifecycle centred on use of the tool rather than the overall development of safety related data. This is important, as some aspects of the generic lifecycle do not align to blueprint development.

The safety arguments proposed must also be consistent with the concept of modular and incremental certification, as this technique will become that standard method of developing safety cases for modular systems. The proposed safety argument modules for IMS blueprints and state data have been developed with this approach in mind.

The proposed solution for blueprints is therefore an extension to techniques previously developed for the management of the safety of software.

A more difficult data problem to manage is the issue of distributed data such as the system state data.

This has consistency and contention problems to overcome due to the concurrent nature of the system and the shared global state made up of local states of independent nodes. It is also based upon a hierarchical structure.

We have proposed a consistency model for this based upon the concept of coordinated atomic actions (CAAs). These have the advantage of preventing contention whilst maintaining the consistency of the global state, but the traditional model of CAAs is for use in multithreaded systems that can share global resources such as memory. Our approach requires a more complex model and therefore the

CAA model has been extended to include distributed coordinated atomic actions (DCAAs). This allows the use of a CAA based approach across a set of distributed nodes and requires the addition of a distributed synchronisation mechanism, DCAA Locks, in order to ensure synchronisation at the initialisation of the DCAA. The lock also allows us to avoid initial contention problems.

In order to more fully understand how the DCAA concept could be implemented in an IMS architecture we need to model the behaviour using a modelling tool.

Of particular concern is the behaviour of the DCAA Lock. The current definition seems to be predictable but there is a possibility that due to competing state change requests the lock process could be deadlocked. This could result in two processing nodes with conflicting requests never managing to achieve a lock. In order to assess the behaviour in this area it requires real-time modelling of the proposed mechanism to check its feasibility. This would also allow evaluation of the real-time impact of a DCAA process on the overall IMS system management behaviour.

One interesting outcome of the safety assessment is the variation in the nature of the safety problem. We have suggested that there is a spectrum of data usage in real-time distributed systems. This ranges from large static data structures to small distributed data structures. The research has highlighted that as we move across this spectrum the nature of safety problem faced by the system implementer changes.

For a large static data structure, such as the IMS blueprints, the nature of the safety problem is mainly concerned with the off-line validation of the data content.

For a small distributed data structure, such as the distributed system state, the nature of the safety problem is mainly concerned with the proof of on-line mechanisms to handle and manipulate the data.

This leads us to the conclusion that it is not possible to derive a specific detailed solution to the entire spectrum of data safety problems. We need to propose a generic process to assess safety needs and assess each form of data using this process.

In this research we have proposed solutions to the IMS safety issues for both types of data. However, there are other possible solutions to some of these IMS issues. One of the main issues faced for the distributed state data problem was the rigid system management hierarchy. The evaluation of options could not identify changes to the current hierarchy that provided a better solution, but a possibility is to remove the hierarchy altogether.

A different approach that could be looked at is to replace the system management components with an agent based approach. This would then alter the system management functionality to a truly distributed system. Agent based approaches would cause less of an overhead on the overall system performance, but this approach brings its own set of safety issues. Agent based software can be far from deterministic and there are real-time issues with the variability of agent performance characteristics. It may be possible to use an agent based approach with DCAAs in order to allow a more lightweight system manager to be developed, but this would be the subject of further research.

We could also look at more fundamental changes to the way of overcoming the data safety problem, such as other ways of initially approaching the safety problem. If we look at other types of system that use data the safety solution employed falls into a number of categories

- Constrain system behaviour – the system itself is modified to ensure the safety of the system is maintained and reliance upon the data reduced.
- Constrain data used – the data used by the system is constrained to a known “safe” subset.
- Limit reliance on the system component by employing an external module that checks that the safety requirements are met. An example of this approach is the use of “wrapper” technology to provide a safe buffer zone around a suspect component that prevents unsafe actions from impacting the rest of the system.

These solutions could be investigated as alternative approaches to the IMS data safety problem.

However, the main area for further work identified by this research is the modelling of the DCAA concept in relation to IMS global state consistency. This work would include:

- Modelling of the DCAA interactions at a more detailed level.
- Assessment of the DCAA Lock mechanism, ensuring no deadlock situation exists.
- The best method to implement the blocking behaviour for other nodes during DCAA operation. This has a direct impact on the application software if it requests a mode change that is then blocked.
- Understand the impact of the DCAA mechanism on the real time behaviour of the IMS system.

These techniques have been developed using IMS as the example system, but the mechanisms and safety approaches are applicable to a wide range of systems that incorporate similar techniques.

At this time IMS is being deployed on real platforms such as Hawk and Tornado, and some of the complex behaviour is already included in the software, such as reconfiguration and fault handling. These more complex system functions are not yet being used in complex situations, but they are being used in simple configuration changes. As the complexity of IMS systems increases they will rely more and more on complex blueprints with more configurations and more system management interactions.

This requires a more robust approach to the safety aspects of the use of blueprint data and distributed real-time behaviour.

We need to ensure that the safety argument for data is included in the modular certification approach, and we need to ensure the blueprint toolset is robust in line with our suggested safety argument. We also need to ensure the consistency is enforced in all distributed state data. Only then will IMS be able to show that the data it uses is acceptably safe.

References and Bibliography

1. University of York Real Time Systems Group. *Introduction to the Real-Time Systems Group* [cited 2008 20th July]; Available from: http://www.cs.york.ac.uk/rts/documents/rts_intro.pdf
2. Van Streen, M. *Distributed Systems - Principles and Paradigms course notes*. 2007 [cited 2008 20th July]; Available from: <http://www.cs.vu.nl/~steen/courses/ds-slides/notes.01.pdf>
3. Emmerich, W. *Distributed Systems Principles course notes*. 1997 [cited 2008 20th July]; Available from: <http://www.cs.ucl.ac.uk/staff/ucacwxe/lectures/ds98-99/dsee3.pdf>
4. Free On-line Dictionary of Computing. *Data Driven Definition*. [cited 2008 20th July]; Available from: <http://foldoc.org/>.
5. Faulkner, A., Storey, N. . *Data: An Often Ignored Component of Safety Related Systems*. in *MoD Equipment Assurance Symposium ESAS02*. 2002.
6. Bradley, A., Fletcher, MA., Miller, P., Moxon, P., Wake, AS., *Integrated Modular Avionics and Certification - An IMA Design Team's View*, in *IEE Certification of Ground/Air Systems Seminar* 1998. p. 1-7.
7. Ministry of Defence, *Interim Defence Standard 00-74 in ASAAC Standards Part 1 - Proposed Standards for Software*. 2005, Defence Procurement Agency.
8. Wake, A., *Modular Systems - An Air Platforms Perspective*, in *ASSC Open Systems Symposium*. 2008.
9. Ministry of Defence, *Defence Standard 00-56 Issue 4*, in *Safety Management Requirements for Defence Systems - Part 1 Requirements*. 2007, Defence Equipment and Support.
10. Ministry of Defence, *Defence Standard 00-56 Issue 4*, in *Safety Management Requirements for Defence Systems - Part 2 Guidance on Establishing a Means of Complying with Part 1*. 2007, Defence Equipment and Support.
11. RTCA, *DO-178B*, in *Software Considerations in Airborne Systems and Equipment Certification*. 1992, RTCA.
12. Ministry of Defence, *Defence Standard 00-55 Issue 2*, in *Requirements for Safety Related Software in Defence Equipment - Part 1 Requirements*. 1997, Defence Equipment and Support.
13. Ministry of Defence, *Defence Standard 00-55 Issue 2*, in *Requirements for Safety Related Software in Defence Equipment - Part 2 Guidance*. 1997, Defence Equipment and Support.
14. IEC, *IEC 61508*, in *Functional safety of electrical/electronic/programmable electronic safety-related systems*. 2005, IEC.
15. Meyers, J., Miller, B., *Policy for Terrain and Obstacle Data*, in *SVS Workshop*. 2006: Seattle.
16. RTCA, *DO-200A*, in *Standards for Processing Aeronautical Data*. 1998, RTCA.
17. Department of Defense, *MIL-STD 882E*, in *Standard Practice for System Safety*. 2005, Department of Defense.
18. Storey, N., Faulkner, A. *The Role of Data in Safety-Related Systems*. in *19th Systems Safety Conference, Huntsville AL*. 2001.
19. Storey, N., Faulkner, A. *Data Management in Data Driven Safety Related Systems*. in *20th Systems Safety Conference*. 2002. Denver CO.
20. Faulkner, A., Storey, N. *Strategies for the Management of Data-Intensive Safety-Related Systems*. in *21st Systems Safety Conference*. 2003. Ottawa.
21. Knight, J., Strunk, EA., Greenwell, WS., Wasson, KS., *Specification and Analysis of Data for Safety Critical Systems*, in *22nd International System Safety Conference*. 2004.
22. Faulkner, A., Bennett, P., Pierce, R., Johnson, IAH., Storey, N. *The Safety Management of Data Driven Safety Related Systems*. in *19th International Conference Safecom 2000*. 2000.

23. Storey, N., Faulkner, A., *The Characteristics of Data in Data-Intensive Safety-Related Systems*. Lecture Notes In Computer Science, 2003(2788): p. 396-409.
24. Storey, N., *The Importance of Data in Safety-Critical Systems*. Safety Systems, 2004. **13**(2): p. 1-4.
25. Storey, N., Faulkner, A. *Data Requirements for Data-Intensive Safety-Related Systems*. in *21st Systems Safety Conference*. 2003. Ottawa.
26. Hollow, P., McDermid, J., Nicholson, M. *Approaches to Certification of Reconfigurable IMA Systems*. in *INCOSE 2000*. 2000.
27. Nicholson, M., Conmy, P., Bate, I., McDermid, J., *Generating and Maintaining a Safety Argument for Integrated Modular Systems*, in *5th Australian Workshop on Safety Critical Systems and Software*. 2000. p. 31-41.
28. Joliffe, G., *Producing a Safety Case for IMA Blueprints*, in *24th Digital Avionics Systems Conference*. 2005. p. 14.
29. Conmy, P., Nicholson, M., McDermid, J. *Safety Assurance Contracts for Integrated Modular Avionics*. in *Eighth Australian Workshop on Safety Critical Systems and Software (SCS 2003)*. 2003.
30. Kelly, T. *Using Software Architecture Techniques to Support the Modular Certification of Safety Critical Systems*. in *Eleventh Australian Workshop on Safety-Related Programmable Systems*. 2005.
31. Blackwell, N., Leinster-Evans, S., Dawkins, SK., *Developing Safety Cases for Integrated Flight Systems*, in *IEEE Aerospace Conference*. 1999.
32. Huang, Y., Kintala, C., *Software Implemented Fault Tolerance: Technologies and Experience*, in *The 23rd Annual International Symposium on Fault-Tolerant Computing*. 1993. p. 2-9.
33. Babaoglu, O., Marzullo, K., *Consistent Global States of Distributed Systems - Fundamental Concepts and Mechanisms*, in *Technical Report UBLCS-93-1*. 1993, University of Bologna.
34. Chandy, K., Lamport, L., *Distributed Snapshots - Determining Global States of Distributed Systems*. ACM Transactions on Computer Systems, 1985. **3**(1): p. 63-75.
35. Belalem, G., Slimani, Y. *A Consistency Protocol Multi-Layer for Replicas Management in Large Scale Systems*. in *World Academy of Science, Engineering and Technology*. 2006.
36. Audsley, N., Burns, A., Richardson, MF., Wellings, AJ., *Data Consistency in Hard Real Time Systems*, in *Technical Report YCS 203*. 1993, Department of Computer Science, University of York.
37. Barbara, D., Garcia-Mollina, H., *Replicated Data Management in Mobile Environments: Anything New Under the Sun?*, in *IFIP Conference on Applications in Parallel and Distributed Computing*. 1994.
38. Frohofer, L., Baumgartner, M., Osrael, J., Goeschka, K. *Data Partitioning through Integrity Constraints*. in *37th Int. Conference on Dependable Systems and Networks (DSN'07)*. 2007.
39. Son, S., Kouloumbis, S. *Replication Control for Distributed Real-Time Database Systems*. in *12th International Conference on Distributed Computing Systems*. 1992.
40. Schaefer, R., *Systems of Systems and Coordinated Atomic Actions*. ACM SIGSOFT Software Engineering Notes, 2005. **30**(1): p. 6.
41. Xu, J. *The CA Action Concept*. 1999 [cited 2008 20th July]; Available from: <http://homepages.cs.ncl.ac.uk/alexander.romanovsky/home.formal/CAA-concept-intro.pdf>.
42. Kelly, T., Weaver, R. *The Goal Structuring Notation - a Safet Argument Notation*. in *Dependable Systems and Networks 2004 Workshop on Assurance Cases*. 2004.