Certification Criteria for Emulation Technology in the Australian Defence Force Military Avionics Context

Flight Lieutenant Derek Reinhardt

Systems Certification and Integrity (SCI)
Directorate General Technical Airworthiness (DGTA)
Bldg L474 (B-2-STH), RAAF Williams, Laverton 3027, Victoria, Australia

derek.reinhardt@defence.gov.au

Abstract

Emulation technology promises to provide a means of addressing obsolescence issues in legacy computer processors in the military avionics domains. It has also been suggested that such technology might apply to safety critical and safety related systems in these domains. Numerous companies either have developed or are developing software components that are capable of emulating different legacy computing platforms. The emulators permit the execution of legacy code on newer computing platforms, without change to existing binary executables or data. Subsequent modifications to the legacy code in question may be made using either the legacy development environment and/or with some emulation technologies using a newer development environment.

The Defence Science and Technology Organisation (DSTO) is presently working with Northrop Grumman Space Technology (NGST) to develop a concept demonstrator utilising NGST's Reconfigurable Processor for Legacy Applications Code Execution (RePLACE) Emulation Technology for the Royal Australian Navy (RAN) Seahawk Display Generator Unit (DGU). To assess how the Australian Defence Force's (ADF's) Technical Airworthiness Authority (TAA) - the Directorate General Technical Airworthiness (DGTA) might accept emulation technology, DGTA has evaluated emulation architectures and specifically RePLACE in the context of the Seahawk DGU. The evaluation has considered the emulation architecture, including identification of risks largely unique to the technology; as well as application of ADF preferred avionics software assurance and software safety standards to this technology.

Evaluation of emulation technology, through exploration of emulation architectures and RePLACE as a case study, has allowed DGTA to define certification and regulatory guidance for the development of emulation technology within the ADF context.

Keywords: Certification, Emulation, Legacy Systems, Software Architecture, Software Safety, Safety Critical.

1 Introduction

The concept of emulation and emulators has been around for many years. Emulators permit the execution of legacy code on newer computing platforms, without changes to existing binary executables or data.

In recent years the power of modern microprocessors has evolved to such an extent that it is now possible to provide real-time software emulation of many legacy microprocessors that were widely used in the late 1970s, 1980s and early 1990s. These advances provide a tremendous opportunity to reuse much of the software developed for these earlier microprocessors without the penalty of having to rehost or translate the software to modern programming languages and microprocessor environments. Furthermore, emulation promises to solve the problem of hardware obsolescence among those legacy systems that are still in use today.

Emulation has been applied to many areas of computing already, including the emulation of older computer game consoles (Atari, Sega, Nintendo Entertainment System, Arcard Platforms, etc), emulation of earlier derivatives of PC, Macintosh, Unix and VaxVMS environments to permit execution of those old applications in a modern environment, and emulation of embedded systems to permit analysis, testing and simulation on development platforms to name but a few. Given the wide application to date of emulation, it is not surprising that emulation technology is now being suggested in the avionics domain, particularly the military avionics domain.

Emulation technology promises to provide a means of addressing many obsolescence issues in legacy computer processors in the military avionics domains, a domain where systems can be subject to comparably longer service lives than equipment in other domains. For example, it is not unusual for military aerospace systems to be in service for 30+ years, although some avionics systems would reasonably be expected to be upgraded over that time period. There have already been a number of programs that have used emulation in this sense. Numerous companies either have developed, or are developing, software components that are capable of emulating different legacy computing platforms for military avionics. One such company is Northrop Grumman Space Technology (NGST) who offer a product called RePLACE. RePLACE has already been applied to numerous avionics systems and microprocessor instruction sets.

Furthermore, it has also been suggested that such technology might be further applied to safety critical and safety related systems in these domains. To date, there is a lack of regulatory guidance and certification criteria relating to how emulation might be applied to safety critical and safety-related systems.

The ADF's Technical Airworthiness Regulator (TAR) – also DGTA, is responsible for defining regulatory and certification criteria for modifications to Australian Military ('State') aircraft. This provides the ADF's TAA with the guidance from which to conduct design acceptance of such technologies. Note that DGTA has a dual responsibility, being both the TAR and TAA. Design acceptance is largely synonymous with type certification within the Federal Aviation Administration (FAA) airworthiness framework.

The Defence Science and Technology Organisation (DSTO) are presently working with NGST to develop a concept demonstrator utilising NGST's RePLACE Emulation Technology for the RAN Seahawk DGU. The DGU hosts several functions that are safety-related, and therefore warrants special consideration within the context of emulation. This development provides DGTA with an opportunity to develop certification criteria for emulation technology and to assess the effectiveness (technical, cost, schedule) of such certification criteria.

The remainder of this paper examines emulation technology, through exploration of emulation architectures and NGST's RePLACE as a case study, to allow DGTA to define certification and regulatory guidance for the development of emulation technology within the ADF context.

2 Examination of Emulation Architectures

In order to define certification criteria for emulation technology, it is firstly necessary to develop an understanding of those software architectures most relevant to emulators. This section introduces the simplest form of emulator architectures.

2.1 A Simple Emulation Architecture (Type 1)

A simple legacy emulation architecture (designated Type 1 for convenience of reference throughout this paper) is detailed in Figure 1 and Figure 2.

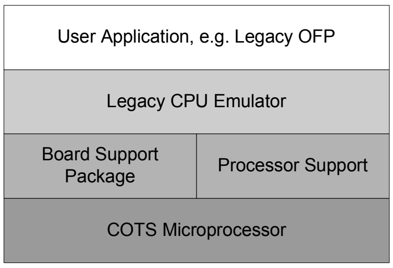

Figure 1: Simple Emulator Architecture (Logical Layers)

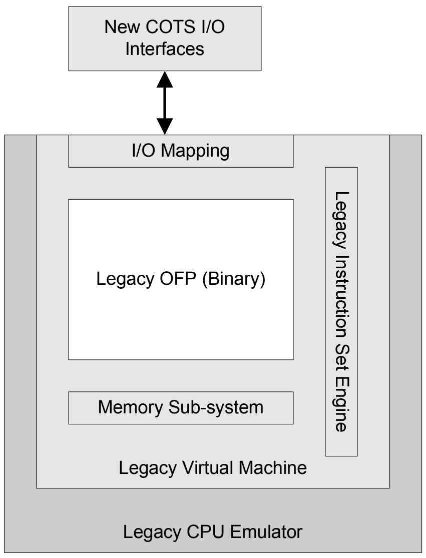

Figure 2: Simple Emulator Architecture (Sub-Elements)

The following paragraphs provide an overview of the emulator components detailed in Figure 2.

The main component of the emulator architecture is the *Legacy Virtual Machine*. The Legacy Virtual Machine consists of the Legacy Instruction Set Engine, Memory Sub-system, I/O Mapping, Legacy Operational Flight Program (OFP) (ie. the binary) and other underlying functionality necessary to emulate the legacy computer environment (eg. Interrupt/exception mechanisms). Encapsulating the Legacy Virtual Machine is the *Legacy CPU Emulator* which provides the interface for the Legacy Virtual Machine to execute in the native processor environment. It is included as a separate element in this architecture for consistency with some more complex architectures considered later in this paper.

The *Memory Sub-System* component's role is to model the memory of the legacy computer environment. This may include logical to physical address translation, memory protection mechanisms and memory regions (non-volatile regions, read-only regions, shared memory regions, etc.).

The *I/O Mapping* component's role is to match the data and control structures, as well as the interfaces of the new replacement I/O devices, to those that are representative of the legacy computer environment.

The *Legacy Instruction Set Engine* is a set of native machine code that fetches, decodes and executes the legacy instructions on the fly. Figure 3 describes the relevant data and information flows that might occur in one such implementation of the Legacy Instruction Set Engine. This example has been based on the MIPs processor, which is generally well understood across the computing domain, although the logical interpretation is easily extended to any type or class of microprocessor. Note that the MIPs processor is not used in the Seahawk DGU, which uses the AAMP processor.

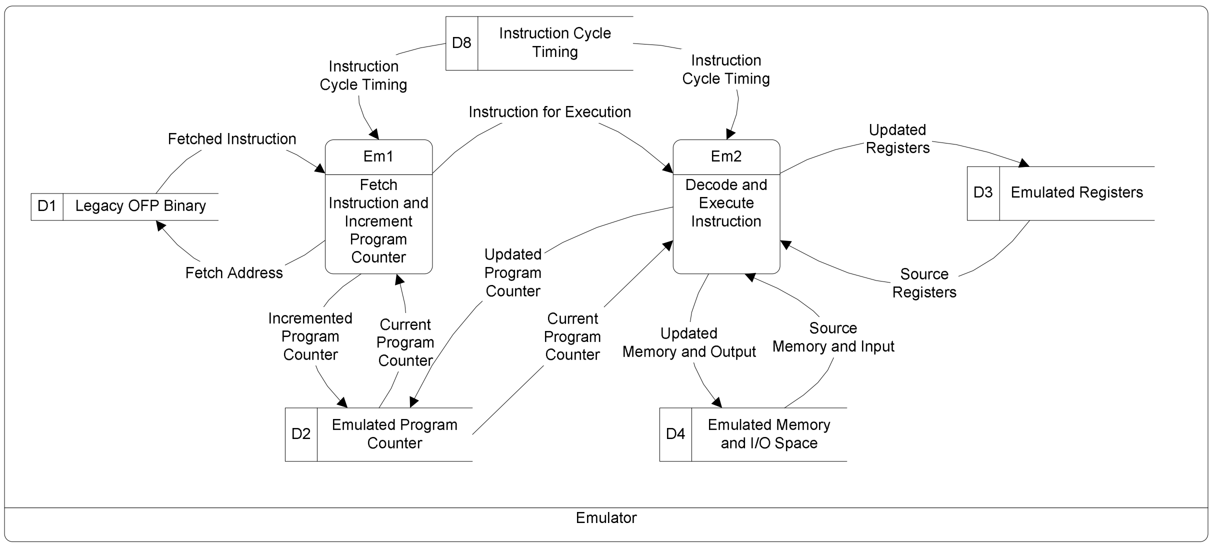

Figure 3: Legacy Instruction Set Engine Data/Information Flow Diagram

2.1.1 Incorporating an RTOS (Type 1A)

Rather than implementing the full suite of system related functions as part of the emulator, it is common for embedded applications of this type to incorporate some form of Real Time Operation System (RTOS). Figure 4 and Figure 5 show the logical layers and sub-elements that such an architecture might consist of. Aside from the incorporation of the RTOS between the emulator and lower level board/processor support firmware/software and the microprocessor and I/O interface, there is no significant change to the components, functional structure or relevant data and information flows within the emulator itself.

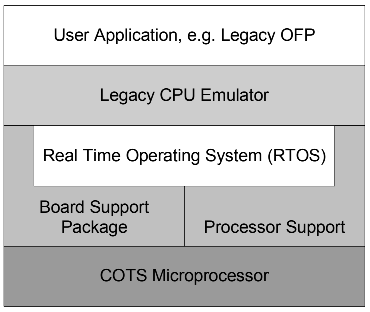

Figure 4: Emulator Architecture (Logical Layers) - Incorporating an RTOS

Of the emulators examined by DGTA, this architecture is the most widely adopted, and will form the starting point for analysis aimed at determining certification criteria for emulation technology.

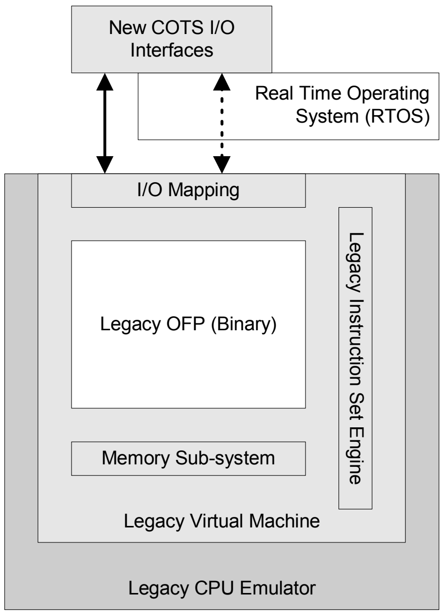

Figure 5: Emulator Architecture (Sub-Elements) - Incorporating an RTOS

3 Analysis of Type 1 Emulation Architecture

To provide an understanding of the software failure modes that might be relevant to the emulation architecture, and importantly what architectural considerations and software assurance activities are required to provide evidence of the absence or handling of these identified failure conditions, it is necessary to conduct some form of software safety analysis. There are numerous software safety analysis techniques that could be applied to such a system including Software Functional Failure Analysis (FFA), Software Fault Tree Analysis (FTA), Software FMEA (FMECA), Software HAZOP (DefStan 00-58 Computer HAZOP), Software Hazard Analysis and Resolution in Design (SHARD) - refinement of Software HAZOP, Markov Analysis and Data Flow Diagrams, Petri Net Analysis and Software Sneak Analysis (ADF 2006, and McKinlay 2001).

Guide Word	Deviation	Cause	Effect	Detection / Protection
Omission	Decode and execute instruction process fails to update memory or output	Programming error within decode and execute instruction process	Memory not updated with new contents Output not updated with new output	Memory and output post update verification
Commission	Decode and execute instruction process invalidly updates memory or output	Programming error within decode and execute instruction process	Memory is corrupted in specific location Output is corrupted	Entry point to decode and execute instruction process limited to following fetch instruction and increments counter process
Early	Decode and execute instruction process updates memory or output before valid processor cycle	Programming error causes instruction implementation to incorrectly replicate cycle synchronisation Cycle synchronisation incorrect	Memory updated out of sequence with other operations Output transitions early	Memory and output update to explicitly check cycle synchronisation Mappings to be established between legacy instruction and emulation implementation and mappings to be verified for functional and temporal equivalence.
Late	Decode and execute instruction process updates memory or output after valid processor cycle	As for early	Memory updated out of sequence with other operations Output transitions late	As for early
Value	Decode and execute instruction process updates memory or output with invalid value or updates wrong memory or output location	Programming error within decode and execute instruction process Incorrect instruction passed to decode and execute process	As for Omission, Commission, Early, Late	As for Omission, Commission, Early and Late.

Table 1: Extract from SHARD on Type 1 Emulator (Updated Memory and Output)

Guide Word	Deviation	Cause	Effect	Detection / Protection
Omission	Decode and execute instruction process fails to update program counter as result of jump instruction	Programming error within decode and execute instruction process	Program counter is not updated with correct value. Emulated program enters incorrect branch of instructions - possible program crash	Program counter is to be verified after operation. Mappings to be established between legacy instruction and emulation implementation and mappings to be verified for functional and temporal equivalence.
Commission	Decode and execute instruction process invalidly updates program counter	Programming error within decode and execute instruction process	Program counter is updated with corrupted value. Emulated program enters incorrect branch or instructions - probable program crash.	Mappings to be established between legacy instruction and emulation implementation and mappings to be verified for functional and temporal equivalence.
Early	Decode and execute instruction process updates program counter before valid processor cycle	Programming error causes instruction implementation to incorrectly replicate cycle synchronisation Cycle synchronisation incorrect	Program counter updated out of sequence with other operations. Emulated program enters incorrect branch or instructions - probable program crash.	Program counter update to explicitly check cycle synchronisation Mappings to be established between legacy instruction and emulation implementation and mappings to be verified for functional and temporal equivalence.
Late	Decode and execute instruction process updates program counter after valid processor cycle	As for early	As for early	As for early
Value	Decode and execute instruction process updates program counter with invalid value	Programming error within decode and execute instruction process Incorrect instruction passed to decode and execute process	As for Omission, Commission, Early, Late	As for Omission, Commission, Early and Late.

Table 2: Extract from SHARD on Type 1 Emulator (Updated Program Counter)

While it is possible to apply aspects of each of these techniques to analyse emulation architectures, and indeed the measured application of a number of these techniques would probably be necessary for the developer of such technologies to provide sufficient evidence as part of a safety case, it is not necessary for defining certification criteria. For the sake of defining certification criteria it is only necessary to develop an understanding of how emulation technology might fail and what might be done to either ensure it can't or doesn't fail; or if it can, then verify that it is sufficiently unlikely to fail. Such understanding should then provide insight into what evidence is required to provide sufficient confidence in these aforementioned properties. The SHARD technique is particularly relevant to developing this understanding as it considers failure modes, their causes, effects, and potential detection or protection means.

SHARD employs a series of guidewords to classify how the information flows and associated communication events (and associated services) might deviate from their intended forms. These are as follows:

- Omission - Service not delivered.
- Commission - Service delivered when not required.
- Early - Service delivered, but early.
- Late - Service delivered, but late.
- Value - Service delivered, but with incorrect value.

SHARD requires that the system be analysed "backwards" from the outputs (ie. identify the system level effects first) back towards the inputs. The internal and input deviations are expressed in terms of how they cause or contribute to deviations in downstream items already investigated. Further information on the SHARD technique can be found in Pumfrey (1999).

SHARD analysis was conducted using the Legacy Instruction Set Engine Data/Information Flow Diagram

(Figure 3) as a reference for information flows that might exist in the emulator, and services that are required from a functional perspective. An extract from the SHARD is presented in Table 1 and Table 2 for the updated memory and output, and updated program counter information flows respectively.

Both Table 1 and Table 2 refer to the term 'temporal equivalence'. 'Temporal equivalence' is used in a general sense here for convenience as each emulator implementation will need to define, within high and low level requirements, this property in the context of the relevant instruction set; legacy CPU architecture, including consideration for pipelining and parallel execution paths; and the configured application set. It is used to capture timing considerations at two levels of abstraction. The first is at the instruction level, which deals with the timing constraints placed on individual instructions, or sequences of instructions for some parallel architectures. The second is at the application level, which deals with ensuring that assigned tasks complete within their scheduled execution time, and that implementation quirks, such as processor cycle based synchronisation schemes (as opposed to interrupt timer based schemes) and the use of No Operations (NOPs) for timing synchronisations do not result in undesired effects (e.g. speed up) when emulated. This second level of abstraction is mostly applicable to those architectures considered in Section 4, however it cannot be ruled out in this context due to potential for synchronisation dependencies (e.g. those resident in timing sensitive executives and I/O). This implies that inspection and analysis of the legacy binary will be required to determine if these schemes are part of the implementation.

Having developed an understanding of the types of failure modes, their causes, effects, and detection/protection means, it is then possible to define architectural or verification requirements relative to those failure modes. The ADF preferred standard for software assurance of airborne software is RTCA/DO-178B (ADF 2005). For the purposes of consistency and clarity, verification requirements shall be defined based on those activities documented in RTCA/DO-178B (RTCA 1992). Readers should refer to DO-178B and related information (DO-248B (RTCA 2001), Order 8110.49 (FAA 2003), CAST 5 (CAST 2000)) for further definitions of software assurance activities described in this paper.

However, to understand the logic behind the approach used to define those architectural and verification requirements and associated DO-178B objectives, it is firstly necessary understand some key aspects of the DO-178B software assurance model. According to DO-178B, verification of airborne software has two complementary objectives. One objective is to demonstrate that software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions have been removed. Noting that the prescription of activities against these two objectives is scaled based on software level within the standard, it is worth considering the approach in general terms.

The first objective is largely supported through definition of and verification against high-level requirements. Where insufficient disclosure or ambiguities exist within the high level requirements, then refinement and further definition of and verification against is required in translation to low-level requirements. Therefore, it follows that provided the developer can adequately disclose the requirements at the prescribed level of detail, then this objective is relatively straight forward to satisfy.

The second objective, however, is not quite as intuitive. It deals with eliciting properties about the software which don't necessarily follow from the set of already defined high and low level requirements, with focus on those properties that could potentially lead to unacceptable failure conditions. Eliciting these properties permits one of two outcomes: either the behaviour is appropriate, in which case it should be captured in the high and/or low level requirements; or the behaviour is inappropriate, in which case the software design and implementation should be changed to remove the behaviour. DO-178B approaches this through prescribing requirements coverage analysis and software structural coverage analysis. Furthermore, establishing requirements traceability from low-level requirements to source code, and to object code, supports providing an understanding of software properties commensurate with this second objective. While there are arguably other ways to elicit such properties, this paper, for the reasons previously documented, will restrict discussion to those called out by DO-178B.

For an emulator, the high and low level requirements would generally need to capture the extent of the instruction set, as well as all other supporting functional and non-functional properties of the machine being emulated. For most legacy CPUs, much of this information would need to be extracted from whatever technical documentation is still available. For CPU manufacturers that have been out of business or have since been subsumed into other businesses, it may no longer be possible to elicit much documentation beyond what is already held by the in-service support organisation and associated technical library. Unlike the commercial world where the internet often becomes a repository for obsolete information, rarely does propriety information relating to obsolete military specific equipment find its way into the public domain. As it is not possible to guarantee that the documentation adequately captures all functional and non-functional properties of the legacy CPU, then it follows that the initial set of high and low level requirements assembled for the emulator might well be incomplete. Therefore the activities, and resultant outcomes of the second aforementioned DO-178B verification objective become especially important as one means of eliciting a complete set of high and low level requirements, not only in the context of the emulator itself, but also in the context of the extant legacy application – in particular, where the legacy application relies on undocumented legacy processor properties. An inspection of DO-178B reveals that activities supporting this objective only start to become applicable at Level C or better, with Level B providing the bulk of necessary activities.

Further to ensuring a complete set of high and low level requirements, the activities and resultant outcomes of the second aforementioned DO-178B verification objective are also necessary to explore properties associated with any underlying components of the emulator that might not directly relate to the execution of instructions, or management of I/O and memory. For example, any monitoring or mapping functions, or failure thereof, should not result in any unacceptable failure conditions.

An extract from the architectural and verification requirement assignment against identified failure modes is presented in Table 3. Full details have not been included on each specific assignment. However, it follows that they are appropriate based on the argument presented earlier in this section.

Detection/Protection	Architectural or Verification Requirement
Mappings to be established between legacy instruction and emulation implementation, and mappings to be verified for functional and temporal equivalence.	Software high level requirements comply with system requirements High-level requirements are accurate and consistent High-level requirements are compatible with target computer Low-level requirements comply with high-level requirements Low-level requirements are accurate and consistent Low-level requirements are compatible with target computer Source Code complies with low-level requirements Executable Object-Code complies with low-level requirements Test coverage of software structure (decision coverage) is achieved
Memory and output post update verification. Memory and output update to explicitly check cycle synchronisation Entry point to decode and execute instruction process limited to following fetch instruction and increments counter process Program counter is to be verified after operation. Program counter update to explicitly check cycle synchronisation Registers are to be verified after operation. Registers update to explicitly check cycle synchronisation Monitoring of pass instruction to decode and execute instruction process to ensure graceful recovery from failure mode Program counter is to be verified after each increment operation. Program counter increment to be explicitly synchronised to fetch instruction. Fetch instruction and increment program counter process to be synchronised with decode and execute instruction process to ensure one decode and execute for each instruction fetched.	Software high level requirements comply with system requirements High-level requirements are accurate and consistent High-level requirements are compatible with target computer Low-level requirements comply with high-level requirements Low-level requirements are accurate and consistent Low-level requirements are compatible with target computer Source Code complies with low-level requirements

Table 3: Determination of Architectural or Verification Requirements for Type 1 Emulator

An inspection of the software assurance activities called out in Table 3, considered in the context of the previous discussion on the DO-178B software assurance model, reveals that these objectives come largely from the set of objectives core to DO-178B Level B. Therefore, it follows that for most safety related systems, the most appropriate software assurance level will be DO-178B Level B. This is further addressed, later in this paper.

4 Further Examination of Emulation Architectures

This section introduces an extension of the emulator architecture that permits changes to be made to the functionality of the legacy binary using a new development environment with code hosted directly into the native environment.

4.1 Incorporating New Functions Developed in Native Code (Type 2)

A legacy emulation architecture that permits the incorporation of new functions developed in native code (designated Type 2 for convenience of reference throughout this paper) is detailed in Figure 6 and Figure 7.

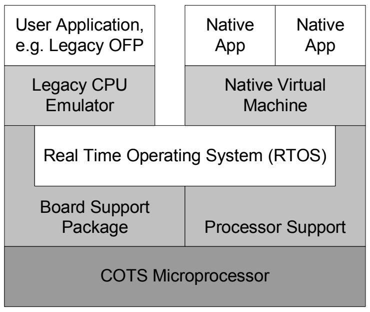

Figure 6: Emulator Architecture (Logical Layers) - Incorporating New Functions Developed in Native Code

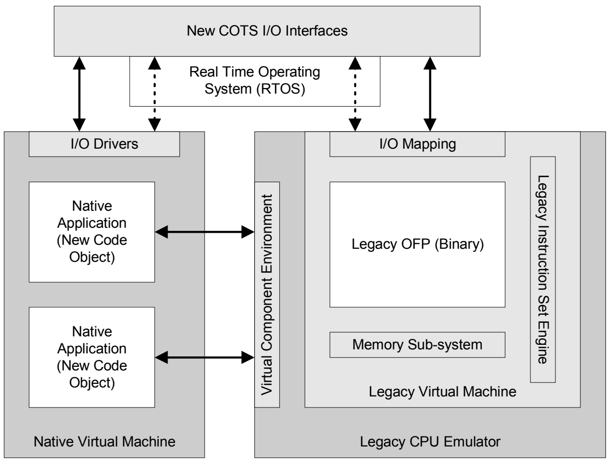
New COTS I/O Interfaces
Real Time Operating System (RTOS)
I/O Drivers
Native Application (New Code Object)
Native Application (New Code Object)
Native Virtual Machine
Virtual Component Environment
I/O Mapping
Legacy OFP (Binary)
Memory Sub-system
Legacy Instruction Set Engine
Legacy Virtual Machine
Legacy CPU Emulator

Figure 7: Emulator Architecture (Sub-Elements) - Incorporating New Functions Developed in Native Code

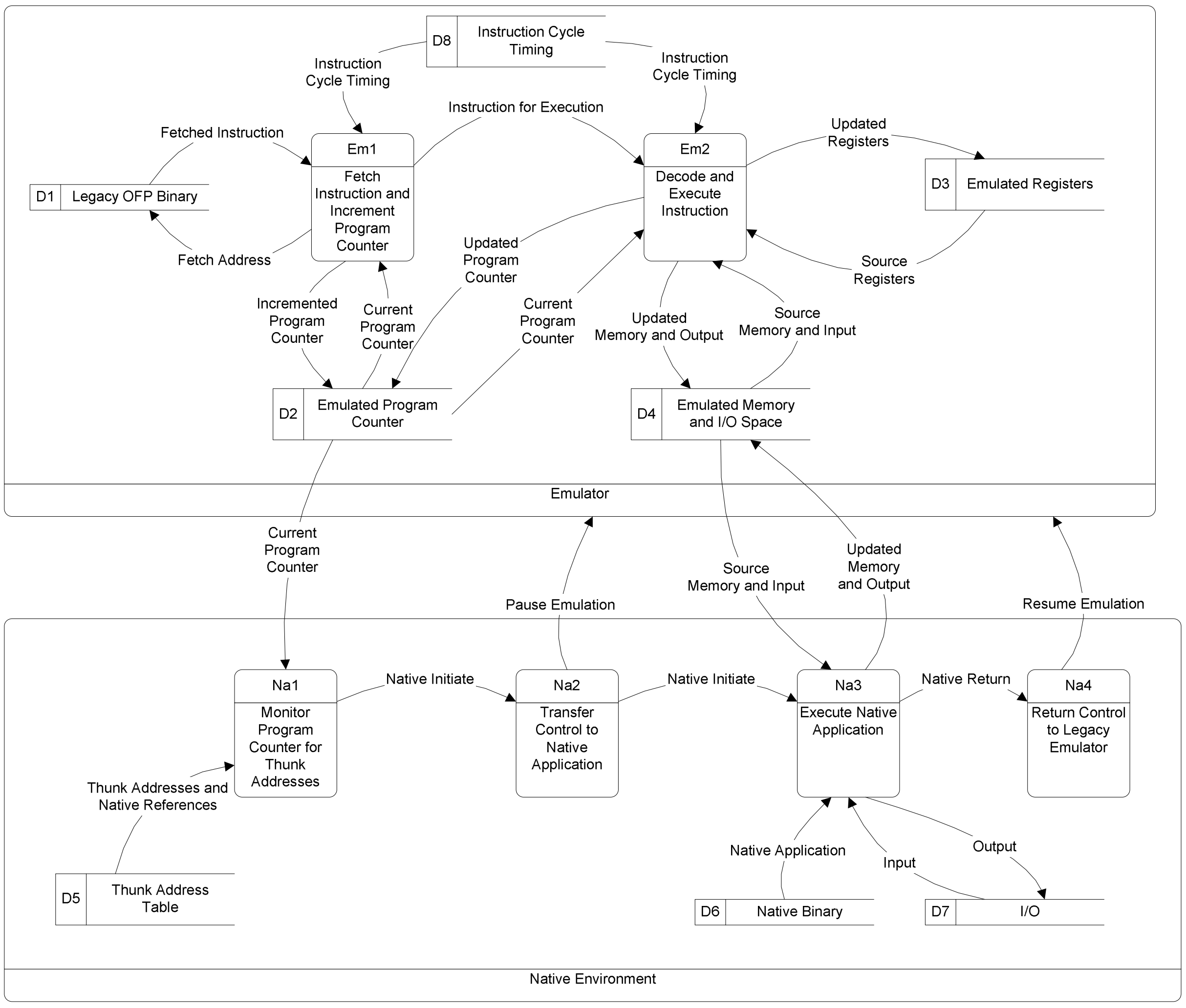

Figure 8: Legacy Instruction Set Engine and Virtual Component Environment Data/Information Flow Diagram

The following paragraphs provide an overview of the emulator components detailed in Figure 7.

Many components within the Legacy Virtual Machine that are common with the Type 1 architecture. As before, encapsulating the Legacy Virtual Machine is the *Legacy CPU Emulator* which provides the interface for the Legacy Virtual Machine to execute in the native processor environment.

However, in this architecture the Legacy CPU Emulator also includes a component labelled the Virtual Component Environment. The *Virtual Component Environment* provides the mechanisms to switch between legacy and new native code environments and share data between them.

Figure 8 describes the relevant data and information flows that might occur in one such implementation of this extended emulation architecture. This example has again been based on the MIPs processor although the logical interpretation is easily extended to any type or class of microprocessor. Figure 8 uses the term 'thunk', which is defined as a reference mapping of code addresses from one system specific form (i.e. legacy address space) to another (i.e. native environment).

5 Safety Analysis of the Legacy Emulation Architecture Incorporating New Functions Developed in Native Code

To provide an understanding of the software failure modes that might be relevant to the Type 2 emulation architecture, and importantly what architectural considerations and software assurance activities are required to provide evidence of the absence and handling of these identified failure conditions, it is again necessary to conduct some form of software safety analysis.

In line with the approach adopted for the Type 1 emulation architecture, a SHARD was conducted using the Legacy Instruction Set Engine and Virtual Component Environment Data/Information Flow Diagram (Figure 8) as a reference for information flows that might exist in the emulator, and services that are required from a functional perspective. An extract from the SHARD is presented in Table 4.

Guide Word	Deviation	Cause	Effect	Detection / Protection
Omission	Failure to pause emulation	Programming error causes a failure to recognise thunk address or to recognise need to pause emulation and transfer control to native function Wrong data in thunk table	Emulated program continues to execute beyond thunk address – synch problems with native functions, possible program crash	Establishment of thunk addresses and storage of thunk addresses within thunk table require level of integrity to process Verification of thunk address integrity, and relevant context to application code segment
Commission	Emulation paused when not required	Programming error causes transfer to native application when not required Wrong data in thunk table	Emulated program will pause, with transfer of control to wrong native function, or program halt	Emulated function to be an atomic operation to ensure interruption from thunking only between instructions. Virtual Component Environment shall be able to detect a native applications anticipated violation of the emulated applications real time constraints and deadlines, and be able to return operation to the emulated function gracefully. As for Omission
Early	Emulation paused earlier than required	Programmer error causes transfer to native application earlier than required	Emulated program will pause, with transfer of control to native function early resulting in state synch problems with emulated program	Emulated function to be an atomic operation to ensure interruption from thunking only between instructions. As for Commission
Late	Emulation paused later than required	Programmer error causes transfer to native application later than required	Emulated program may pause, with late transfer of control to native function resulting in state synchronisation problems with emulated program	As for Early
Value	Emulation pauses with wrong state	Programmer error causes program counter, register and memory/output state of emulator to be incorrectly captured	Native function accessing emulated state may perform operations on incorrect data. Return of execution of emulator likely to result in program crash, or operations on invalid data	Emulated function to be an atomic operation to ensure interruption from thunking only between instructions. Transfer control is not permitted access to emulator state unless otherwise justified.

Table 4: Extract from SHARD on Type 2 Emulator (Pause Emulation)

Detection/Protection	Architectural or Verification Requirement
Verification of thunk address integrity, and relevant context to application code segment Emulated function to be an atomic operation to ensure interruption from thunking only between instructions. Emulated function to be an atomic operation to ensure interruption from thunking only between instructions. Transfer control is not permitted access to emulator state unless otherwise justified. Virtual Component Environment must be able to detect a native application's anticipated violation of the emulated application's real time constraints and deadlines (to achieve temporal equivalence as previously defined), and be able to return operation to the emulated function gracefully.	Software high level requirements comply with system requirements High-level requirements are accurate and consistent High-level requirements are compatible with target computer Low-level requirements comply with high-level requirements Low-level requirements are accurate and consistent Low-level requirements are compatible with target computer Source Code complies with low-level requirements Test coverage of software structure (decision coverage) is achieved
Establishment of thunk addresses and storage of thunk addresses within thunk table require level of integrity to process	The development of native functions, their effect on the emulated systems state, and the integrity of the overall system are closely linked. Therefore, it may be necessary to apply more rigorous software assurance activities than associated with the severity of failure of the native function alone. Similar software assurance activities may be required as for emulator itself. This is dependant on the nature of the native function. Those that have significant effect on the state of the emulated system are likely to require additional assurance activities (i.e. equivalent to those defined for the emulator). Those functions that don't may be conducted at a software assurance level commensurate with the severity of failure of that function.
Transfer control is not permitted access to emulator state unless otherwise justified.	Protected Domain – Partitioned RTOS

Table 5: Determination of Architectural or Verification Requirements for Type 2 Emulator

Having developed an understanding of the types of failure modes, their causes, effects, and detection/protection means, it is then possible to define architectural or verification requirements relative to those failure modes. Section 3 has already discussed the relationship of the DO-178B software assurance model and the associated critical properties elicited from relevant activities. The same logic is applied in this case. An extract from the architectural and verification requirement assignment against identified failure modes is presented in Table 5. Full details have not been included on each specific assignment. However, it follows that they are appropriate based on the argument presented earlier in this section.

An inspection of the software assurance activities called out in Table 5 reveals that these objectives again come largely from the set of objectives core to DO-178B Level B. Therefore in a general sense, it follows that for most safety related systems, the most appropriate software assurance level will be DO-178B Level B. This is addressed in greater detail later in this paper. It should also be noted now that a requirement is identified relating to the interaction between the emulator and native environment. A robust means of addressing this requirement is through a protected domain (partitioned) RTOS. A broader inspection of the SHARD analysis, beyond the extent of that presented in this paper, also dictates a requirement for complete isolation of the emulator from the new COTS hardware (including I/O) by means such as the protected domain (partitioned) RTOS.

5.1 Incorporating a Protected Domain RTOS (Type 3)

A legacy emulation architecture that extends the Type 1 architecture to incorporate a protected domain RTOS (designated Type 3 for convenience of reference throughout this paper) is detailed in Figure 9 and Figure 10.

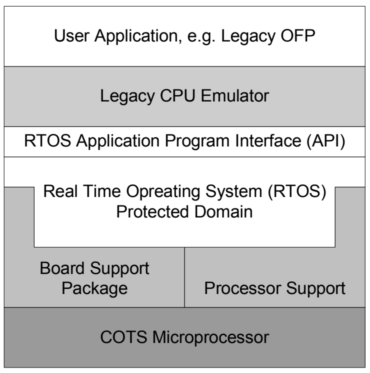
User Application, e.g. Legacy OFP
Legacy CPU Emulator
RTOS Application Program Interface (API)
Real Time Opreating System (RTOS) Protected Domain
Board Support Package
Processor Support
COTS Microprocessor

Figure 9: Emulator Architecture (Logical Layers) - Incorporating a Protected Domain RTOS

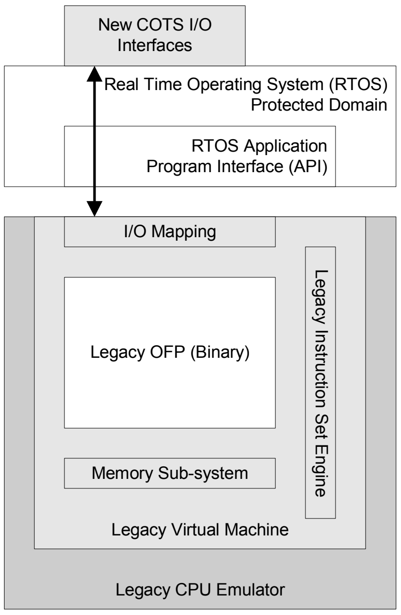
New COTS I/O Interfaces
Real Time Operating System (RTOS) Protected Domain
RTOS Application Program Interface (API)
I/O Mapping
Legacy OFP (Binary)
Legacy Instruction Set Engine
Memory Sub-system
Legacy Virtual Machine
Legacy CPU Emulator

Figure 10: Emulator Architecture (Sub-Elements) - Incorporating a Protected Domain RTOS

The significant change compared with the Type 1 architecture is the complete isolation of the emulator from the new COTS hardware (including I/O) by the protected domain (partitioned) RTOS.

This approach is ideally suited to those emulator applications where there is no immediate requirement to introduce new functionality into the legacy OFP using the native environment (as described in the Type 2 emulator), but for which future capability introduction may be required. The introduction of the protected domain (partitioned) RTOS provides a future expansion capability that ensures it is possible to later introduce new functionality in the native environment, without significant rework of the emulator. For example, changes to the emulator would likely be restricted to the addition of a virtual component environment.

5.2 Emulation Architecture Incorporating New Functions Developed in Native Code (Type 4)

A legacy emulation architecture that extends the Type 2 architecture to incorporate a protected domain RTOS (Type 3 features) that facilitates the incorporation of new functions developed in native code (designated Type 4 for convenience of reference throughout this paper) is detailed in Figure 11, Figure 12, and Figure 13.

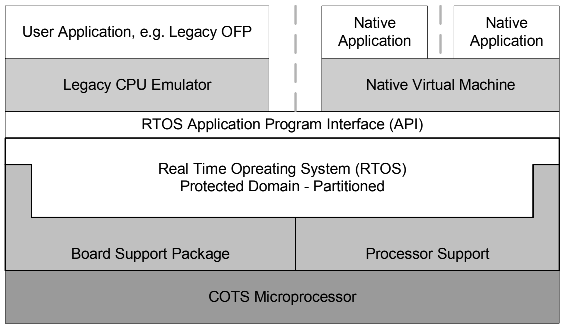
User Application, e.g. Legacy OFP
Native Application
Native Application
Legacy CPU Emulator
Native Virtual Machine
RTOS Application Program Interface (API)
Real Time Opreating System (RTOS) Protected Domain - Partitioned
Board Support Package
Processor Support
COTS Microprocessor

Figure 11: Emulator Architecture (Logical Layers) - Incorporating New Functions Developed in Native Code

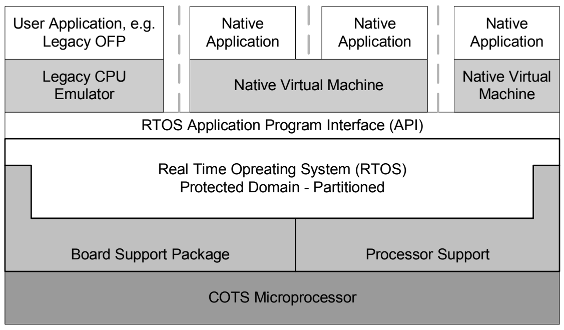

Figure 12: Emulator Architecture (Logical Layers) - Incorporating New Functions Developed in Native Code

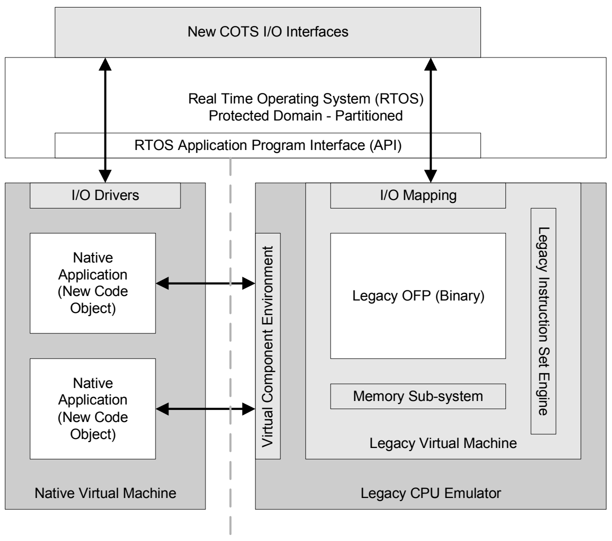
New COTS I/O Interfaces
Real Time Operating System (RTOS) Protected Domain - Partitioned
RTOS Application Program Interface (API)
I/O Drivers
Native Application (New Code Object)
Native Application (New Code Object)
Native Virtual Machine
Virtual Component Environment
I/O Mapping
Legacy OFP (Binary)
Legacy Instruction Set Engine
Memory Sub-system
Legacy Virtual Machine
Legacy CPU Emulator

Figure 13: Emulator Architecture (Sub-Elements) - Incorporating New Functions Developed in Native Code

The following paragraphs provide an overview of the emulator components detailed in Figure 13.

The significant change compared with the Type 2 architecture is the complete isolation of the emulator and native virtual machine from the new COTS hardware (including I/O) by the protected domain (partitioned) RTOS. Furthermore complete temporal and spatial partitioning is now provided by the protected domain (partitioned) RTOS of the legacy CPU emulator and the native virtual machine to ensure adequate separation of legacy and new native functions (represented by the gray dashed line)

While the temporal and spatial partitioning provided by the protected domain RTOS now ensures that the legacy application will not crash as a result of a problem with functions implemented in the native environment, there are some additional architectural issues that need to be addressed. For example, the virtual component environment must now exhibit safety properties to allow the legacy application to continue operating in event the native code fails to return control to the legacy application in a functionally appropriate or timely manner.

6 Recommendations Relating to Emulator Architectures

The analysis conducted in earlier sections of this paper has provided an appreciation of the failure modes that might be associated with the emulation architectures considered. This permits recommendations to be formed on the relevance of particular emulation architectures to the severity of various safety and mission failure conditions. Although this paper is primarily aimed at safety critical and safety related systems, recent guidance in AAP7001.054 Sect 2 Chap 7 (ADF 2005) has provided a framework through which those software assurance activities relevant to safety critical and safety related systems can be applied commensurately to mission systems. Table 6 details the emulation architecture types considered in this paper, and the safety or mission failure conditions for which they are recommended.

Failure Condition		Type1	Type2	Type3	Type4
Safety	Catastrophic	NR	NR	HR	R[1]
	Hazardous	NR	NR	HR	R[1]
	Major	R[1]	R[1]	HR	HR
	Minor	R	R	HR	HR
	No Effect	HR	HR	R[2]	R[2]
Mission	Critical	R	R	HR	HR
	Serious	R	R	HR	HR
	Important	HR	HR	R[2]	R[2]

NR=Not Recommended, R=Recommended, HR=Highly Recommended

Note 1: Recommended only if the sub-elements have been subjected to rigorous software safety analysis that shows the absence or handling of all potential failure modes.

Note 2: Recommended rather than Highly Recommended based on the cost associated with the purchase of a protected domain and partitioned RTOS.

Table 6: Emulation Architecture Recommendations

7 Issues with the Native Code Approach

Subsequent modifications to legacy code hosted on the emulator may be made using either the legacy development environment or a newer development environment.

There may be substantial risks associated with making any more than a small number of changes to the system using the newer development environment and native code. This is because of the difficulty of being able to demonstrate precise knowledge of the pre-conditions to modifications from exit points of the legacy code increases with each subsequent change. Similar difficulties might also exist for the post-conditions of modifications and entry points back into the legacy code. These problems are particularly pronounced for legacy software that has limited available documentation (often the case of legacy systems), or where developer's knowledge of the legacy software is no longer sufficient. The problems may be further exacerbated by poor control over the determination of entry and exit points to and from the legacy code, and the amount of coupling permitted between various native code elements. A robust Application Programming Interface (API) is therefore required to provide tight control of the entry and exit points.

Specific architectural considerations, including partitioning (spatial and temporal as provided by a partitioned RTOS), and related analysis would be required to demonstrate finite, well defined dependencies between subsequent new developments and legacy code. Such analysis would require a thorough understanding of the emulator, the legacy software and the legacy processor. Risks associated with adding new code can be mitigated largely by detailed analysis, as suggested throughout this paper, and planning of new features as part of a appropriately controlled and managed change process. Tool support would also be desirable to assist with providing an understanding of the legacy and native implementations.

Some emulators provide embedded real-time, non-intrusive monitoring and legacy code debugging services as part of the virtual component environment or lower-level CPU emulator. Such services may provide developers with tools necessary to mitigate aspects of the aforementioned problems by providing visibility into the entry and exit points across the boundaries between the legacy and native code elements.

One strategy that might also address aspects of this problem is to eventually translate the executive out of the legacy application into the native environment, with the legacy binary being used as a library of functions. NGST has successfully implemented this approach with some other avionics systems, although proprietary and US State Department restrictions prevent disclosure in the public domain. Specific software safety analysis would be required to provide an understanding of any risks with this approach.

The risks identified above must be weighed against the potential cost and schedule benefits offered by emulation, and the risks of alternative software approaches for upgrading systems.

8 Software Assurance Evidence Requirements for Emulation

The ADF preferred standard for software assurance of airborne software is RTCA/DO-178B (ADF 2005). Although it is acceptable to develop emulation within the framework of other relevant software assurance and software safety standards, this paper will restrict the provision of certification criteria to DO-178B. Comparisons to other standards may be developed through consideration of the critical software assurance activities identified in this paper.

Table 7 defines the DO-178B software levels relevant to emulation based on those critical software assurance activities identified in previous sections of this paper. The levels are determined by a comparison of those critical software assurance activities with those activities normally prescribed by DO-178B at the respective software levels defined in that standard.

Failure Condition		**DO-178B Software Level**	**Software Level for Emulation**
Safety	Catastrophic	Level A	Level A
	Hazardous	Level B	Level B
	Major	Level C	Level B
	Minor	Level D	Level C
	No Effect	Nil	Level D
Failure Condition		**AAP7001.054 Guidance**	**Software Level for Emulation**
Mission	Critical	Level C	Level B
	Serious	Level D	Level C+
	Important	Nil	Level D

Table 7: Software Levels for Emulation

Table 8 details other additional activities required for Level C+, over those activities required for Level C. These activities largely mirror those specific Level B activities identified in the earlier analysis that are critical to meeting and verifying detection/protection requirements, and meeting the desired level of integrity for the system. Where independence, as defined by DO-178B, is believed to provide further assurance to the satisfaction of the relevant DO-178B objective, then a requirement for it has also been documented. Similarly, where independence is not viewed as a key contributor to the outcome of the activity, then it is documented as not required.

DO-178 Reference	**Objective**
A-3-1 (6.3.1a)	Software high level requirements comply with system requirements (satisfied with independence)
A-3-2 (6.3.1b)	High-level requirements are accurate and consistent
A-3-3 (6.3.1c)	High-level requirements are compatible with target computer (satisfied with independence)
A-4-1 (6.3.2a)	Low-level requirements comply with high-level requirements (satisfied with independence)
A-4-2 (6.3.2b)	Low-level requirements are accurate and consistent (satisfied with independence)
A-4-3 (6.3.2c)	Low-level requirements are compatible with target computer
A-5-1 (6.3.4a)	Source Code complies with low-level requirements (satisfied with independence)
A-6-3 (6.4.2.1, 6.4.3)	Executable Object-Code complies with low-level requirements (satisfied with independence)
A-7-6 (6.4.4.2a, 6.4.4.2b)	Test coverage of software structure (decision coverage) is achieved (independence not required)

Table 8: Level C+ Additional Activities Over Level C

Although such prescription detailed in Table 7 departs from the traditional hazard severity / software level alignment of Aerospace Recommended Practice (ARP) 4754 and DO-178B, emulation technology presents specific architectural risks that require specific assurance activities to mitigate. Developers might argue that the increase in software assurance level for emulation will significantly increase the costs associated with the introduction of emulation technology. While there is an element of truth to this argument, there are a number of key points that provide an appropriate tradeoff against the cost increase. These are as follows:

- The size of the emulator (in terms of lines of code) will generally be only a small proportion of ‘real world’ legacy binary (lines of code) for military avionics equipment (e.g. emulator’s lines of code is less than 25% ‘real world’ legacy binary). A typical ‘real world’ legacy binary in currently operating Australian military aircraft is of the order of 150,000 lines of code, although future aircraft and systems will continue to see this figure increase. These figures are based on the RePLACE and DGU example, and are considered typical of such implementations.

Therefore, the number of lines of code to which the more stringent software level should apply is not substantial, and certainly less than the legacy binary. It is important to note that the guidance does pertain to the emulator only, and not to the legacy OFP (binary). This table does not imply that the legacy binary should be redeveloped to the prescribed software level.

- The service history of the legacy software will be yielding a perceived software failure rate or rate of problem occurrences. This will be interpreted by operators both in terms of the reliability or availability, and thus the capability integrity, of the associated system; and also the inherent level of safety currently provided by the system. Service history is one important attribute as it is unlikely that most legacy systems will have been developed with the requirements of most current software assurance or safety standards in mind. Reflecting on the software failure rate, it is generally argued by the software community that such a rate is not actually a reliability (ie. reliability normally being a measure of a systems susceptibility to random failure conditions, whereas this is more synonymous as a measure of the software's exposure to conditions that might uncover systematic errors). However, it does provide a baseline to operators as to the 'apparent reliability', and 'level of safety' of their avionics equipment. Importantly, it also provides technical support staff with an understanding of the software's contribution to any identified failure modes. Therefore, it should be the goal of any program addressing the equipment obsolescence to provide properties commensurate or better than those experienced on the original legacy systems. This places some specific integrity requirements on the emulator. For example, the emulator should not introduce any further failure modes that might reduce the 'apparent reliability' or 'level of safety' of the system. Furthermore, benign failures should remain benign, or be handled by the emulator. One means of achieving this is to apply a greater level of rigour, appropriately targeted, to the emulator than for that required of the original legacy binary, thus providing a greater level of integrity in the emulator software. An appropriate, targeted increase in the software level for the emulator therefore justifies the applicable cost increase.

- The software assurance level, and associated prescription/definition of activities for Minor, No Safety Effect, and Mission Important categories is not significantly greater that the level normally defined under normal circumstances for these systems. Therefore, these systems provide a suitable entry point for the technology into the military avionics domain.

9 RePLACE Dual Instruction Set Computer (DISC)

DSTO are presently working with NGST to develop a concept demonstrator utilising NGST's RePLACE Emulation Technology for the RAN Seahawk DGU. The Dual Instruction Set Computer (DISC) variant of RePLACE, as distinct from other RePLACE variants (eg. X-Port and hybrid), has been identified by NGST as most applicable to emulating the DGU's AAMP processors. This identification is based on consideration of the AAMP processor's performance against a proposed native processor (ie. PowerPC), with due consideration for the RePLACE variant's computational overhead. It is therefore necessary to examine RePLACE DISC in the context of the guidance already formulated in this paper.

9.1 Overview of RePLACE Architecture

Figure 14 details the architecture of the RePLACE DISC. By inspection it is possible to determine that it closely represents the Type 2 architecture already covered in this paper.

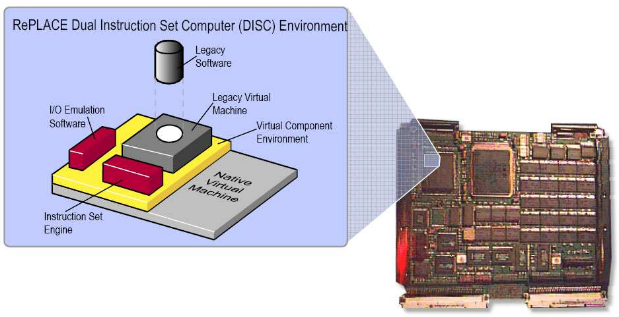

Figure 14: RePLACE Dual Instruction Set Computer (NGST 2005)

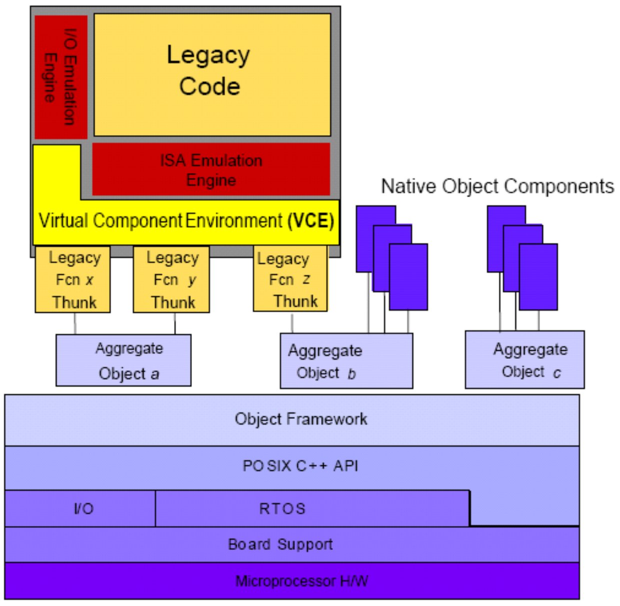

Figure 15: RePLACE DISC (Logical Layers) (NGST 2005)

9.2 Assessment of RePLACE

DGTA funded a US based company, Certification Services Inc (CSI), under a DGTA standing offer to conduct a DO-178B audit of the RePLACE program. The audit considered both DO-178B Level B and C, with a specific goal to identify the practicality of applying such objectives to the RePLACE development, and to assess any issues (technical, cost, schedule) that might exist in transitioning an existing program within a framework that would meet the objectives identified throughout this paper. CSI are highly skilled in evaluating the application of DO-178B to avionics developments, and specifically Mike DeWalt, whom conducted this audit, is considered an authority on DO-178B.

The audit (CSI 2005) found that the RePLACE development does not yet satisfy all relevant DO-178B objectives, and although the RePLACE program in its current state is mostly close to satisfying both Level B and Level C, many of those objectives that it does not yet address are considered essential in this context. Those objectives not presently addressed predominantly relate to requirements traceability, verification and some software configuration management activities. It is important to note that RePLACE Seahawk DGU program is presently being conducted as a concept technology demonstrator, and therefore satisfaction of many of these objectives is beyond the scope of funding available in such programmes.

CSI's assessment is that there is little technical risk of the RePLACE program not being able to meet DGTA's expectations with respect to avionics software assurance. However, cost and schedule risk were identified relating to the generation of software assurance artefacts and the rework necessary to ensure that RePLACE fully meets the relevant DO-178B requirements. However, it was assessed that through some targeted certification risk reduction activities, it is possible to constrain cost and schedule risks to suitable levels.

Following the audit, DGTA published a series of papers on how the Commonwealth might accept RePLACE as part of the design acceptance process used for modifications to Australian State aircraft (DGTA 2005). These papers formed the starting point of further negotiation and development with NGST and DSTO. The guidance in these papers was principally based on the analysis which forms the background of the material presented in this paper.

Post-audit work conduct between DGTA, DSTO and NGST, which is still on-going, has recently resulted in NGST delivering a white paper that demonstrates a qualified understanding of cost and schedule risks. Furthermore, DGTA assesses that the identified cost and schedule reflect that emulation is a cost effective option for addressing legacy obsolescence in some safety related and mission systems. Further details relating to cost and schedule are commercially sensitive and cannot be discussed further in this paper.

RePLACE for the Seahawk DGU program is presently hosted on the Wind River VxWorks OS, a non-DO-178B compliant RTOS. VxWorks was selected for the DGU emulator demonstrations due to the high cost of other DO-178B compliant RTOS's, and the limited funds available for the Seahawk DGU emulation demonstrator. There is some work NGST would be required to undertake to modify any system calls and software structure within this implementation of the RePLACE application to accommodate a different RTOS. Other RePLACE products have already been hosted on protected domain RTOS's, indicating that there is unlikely to be any technical barriers to moving to a protected domain RTOS (eg. Green Hills Integrity OS) for the Seahawk DGU RePLACE application .

10 Summary

Evaluation of emulation technology, through exploration of several emulation architectures and of RePLACE as a case study, has allowed DGTA to define certification and regulatory guidance for the development of emulation technology within the ADF context. The trial application of this certification guidance with the Seahawk DGU RePLACE concept technology demonstrator has permitted an evaluation of the effectiveness of the prescribed DGTA certification criteria. At this time DGTA is satisfied that this guidance will promote an acceptable level of safety for emulation on legacy military avionics while still ensuring emulation is a cost effective option for addressing legacy obsolescence.

11 Acknowledgments

I would like to thank Systems Certification and Integrity (SCI) – DGTA staff including Mark Wade, Squadron Leader Ben Musial and Flight Lieutenant Wendell Fox for their input to and review of all my work relating to emulation.

I would also like to thank Paul Vicen, Tamy Staub, Curt Pflasterer and other RePLACE staff at Northrop Grumman Space Technology (NGST) for their input to this paper relating to RePLACE and enthusiasm to explore certification criteria with DGTA.

Finally I would like to thank Dr Rob O'Dowd, Mark Davies and David Culpin of Air Operations Division – Defence Science and Technology Organisation for their coordination of RePLACE development activities with NGST, and their on-going liaison with DGTA.

12 References

The following documents, papers and publications are referenced throughout this paper. A number of these documents are not available in the public domain for propriety or confidentiality reasons. Readers wishing to seek further information should direct their queries to the author of this paper, or the relevant standards body.

Aerospace Recommended Practice ARP4754 (1996) Certification Considerations for Highly Integrated or Complex Avionics Systems, Society of Automotive Engineers.

Australian Defence Force (2005) Australian Air Publication (AAP) 7001.054 Airworthiness Design Requirements Manual AM1.

Australian Defence Force (2006) Aircraft System Safety Engineering Course – Software Safety Course Notes developed jointly by Systems Certification and Integrity – DGTA and Ball Solutions Group.

Certification Authorities Software Team (2000) Position Paper Cast 5 – Guidelines for Proposing Alternate Means of Compliance to DO-178B, Federal Aviation Authority.

Certification Services Inc (2005) Evaluation of the NGST RePLACE Product, CSI Document 05-276-1246 Rev03.

Directorate General Technical Airworthiness (2005) Paper on How the Commonwealth Might Accept RePLACE – Issue 3, Australian Defence Force.

Federal Aviation Authority (FAA) Order 8110.49 (2003) Software Approval Guidelines, USA.

McKinlay, A. (2001) Software Safety Course Notes, Aviation Safety, School of Engineering, University of Southern California, USA.

Northrop Grumman Space Technology (2005) RePLACE Technology – Bringing 20th Century Systems into the 21st Century - Marketing Brief, Dayton Ohio, USA.

Pumfrey, D. (1999) The Principled Design of Computer System Safety Analyses, PhD Thesis, Department of Computer Science, University of York, UK.

RTCA Inc (1992) RTCA/DO-178B Software Considerations In Airborne Systems and Equipment Certification, Washington, D. C. USA.

RTCA Inc (2001) RTCA/DO-248B Final Report for Clarification of DO-178B Software Considerations in Airborne Systems and Equipment Certification, Washington, D. C. USA.