IMA Systems Development and Configuration: From UML to Binary. A Proposal of UML Profile to be used with the ARINC 653 Standard.

I Lafoz, M Mozas, O Charrier, C Fernández de la Hoz

IMA Systems Development and Configuration: From UML to Binary. A Proposal of UML Profile to be used with the ARINC 653 Standard.

I. Lafoz[1], M. A. Mozas[2], O. Charrier[3], C. Fernández de la Hoz[4]

1, 2 and 4: EADS-CASA, Pº John Lennon, s/n, 28906 Getafe (Madrid), Spain
3: Wind River, 3 avenue du Canada 91975 Les Ulis (Paris), France

Abstract:

The ARINC 653 standard is used more and more often for Safety-Critical Systems in Aerospace. The experience from the design and implementation of this kind of systems introduces new considerations to take into account compare to classical software development workflow, mainly related to the objective of multiple applications running on the same system. The return of experience that is presented in this paper includes a proposed UML extension (via profile) that resolves the lack of UML modelling formalism for the ARINC 653 artefacts, the XML schema updates to fit the objective of application independency, a validated way of automatically generating code relating to all the ARINC 653 elements, an optimal framework for defining tests and required stubs, and the use of qualified tools to verify and generate the binary version of the system configuration tables.

Keywords: IMA systems, partitioning, ARINC 653, UML profile, XML schema, qualified tools, SW architectural design, VxWorks 653, automatic code generation and testing effort reduction.

1. Introduction.

The advantages of IMA architectures have been widely discussed and demonstrated in recent years (multiple applications sharing and reusing the same computing resources, software isolated from the underlying bus and hardware architecture, maximize reuse, reduction of the cost of changes in terms of re-test, etc.) [2]. Despite a number of IMA architectures and standards have been emerged, the ARINC Specification 653 has achieved the widest adoption in the avionics community [1].

The current paper presents the specific characteristics of the design of ARINC 653 systems in section 2, and proposes and describes in detail a UML extension to deal with this design in chapter 3. Chapter 4 is focused on the description of the use of XML schema for defining the configuration of this kind of system. Once the design and configuration phases are covered, chapters 5 and 6 present an automatic generation of the configuration tables and a qualified validation process of them. Furthermore, automatic generation related to ARINC 653 artefacts is also described in terms of code and, in chapter 7, in terms of partition tests and stubs. Chapter 8 shows the achieved experience with the technology described.

2. ARINC 653 Systems Design Specificities.

The classical software design and development workflows don't usually consider the partitioning requirements of IMA architectures. Moreover, the design of a complete partitions architecture becomes a challenge for the System Architect. The EADS-CASA experience in the design and development of complete IMA systems is here presented in terms of a set of criterions to guide the partitions selection and design.

A partition based design must take into account both the available resources, such as budget (time and staff), CPU and memory, and some system requirements, such as DAL criticality level, flexibility and fault tolerance. The goal of the selection/design of the partitions procedure is to reach the best balance between the use of resources and the accomplishment of requirement by focusing on the impact of using partitioning with respect to the next Quality Factors:

- Certification Effort. A higher DAL a higher development and overall verification cost. Nevertheless, the isolation between software with different DAL assignments allows the application of different processes according to the needs of each software partition.
- Safety. The isolation between partitions prevents failure propagation.
- Reusability. The reusability reinforces the well-defined software component functionality, suitable for software reuse, and reduces undesirable execution coupling of software verified in equivalent contexts.
- Scalability. The isolation between software reduces the needs of regression testing in case of software changes and allows the introduction or removal of software without changing execution conditions.

- Testability. The clearly defined and controlled interface of a partition, besides the standard execution environment definition, makes the testing process and tools easier and more accurate.
- Design Cost. The higher design cost must be considered in terms of data coupling, synchronization, coherence and complexity of interfaces between partitions. Moreover, are there a maximum affordable/reasonable number of partitions/ports from the design point of view?
- Performance. Increasing the number of partitions produces a loss of performance due to the next issues:
    - Switching Partition Time. The context switching between partitions always implies a loss of time. This time directly depends on the operating system features.
    - Worst-Case Time Assignment. The time or minor frame assigned to a partition must support the worst-case execution time.
    - Worst-Case Memory Assignment. The memory assigned to a partition must support the worst-case execution resources usage.
    - Inter-Partition Communication. The spent of time of the standard mechanism for communicating partitions based on APEX ports is higher than other classical or intra-partition communication mechanisms.

Some others possible quality factors could be considered for deciding the partitioning as Security, IO Partitions, Development Tool-Chain or Languages, etc., but they are out of the scope of this analysis.
An iterative procedure is shown in Figure 1, in order to drive the trade-off between the impact on the considered parameters and the requirements and available resources and find the most beneficial solution. The detailed description of this iterative decision flow is out of the scope of this paper, nevertheless a brief view on it has been shown in order to justify some of the concepts presented.

Once the partitioning architecture is defined a new UML profile is proposed in order to deal with the lack of modelling formalism of the ARINC 653 elements: partitions, processes, inter-partitions communications, etc. Furthermore, the new UML extension [3] allows integrating in the model the design of the partitions and the rest of the UML design in the first stages of the architectural design. Thus, a complete coherence checks of the whole architecture at the design level, which reduces the possible undetected design errors.

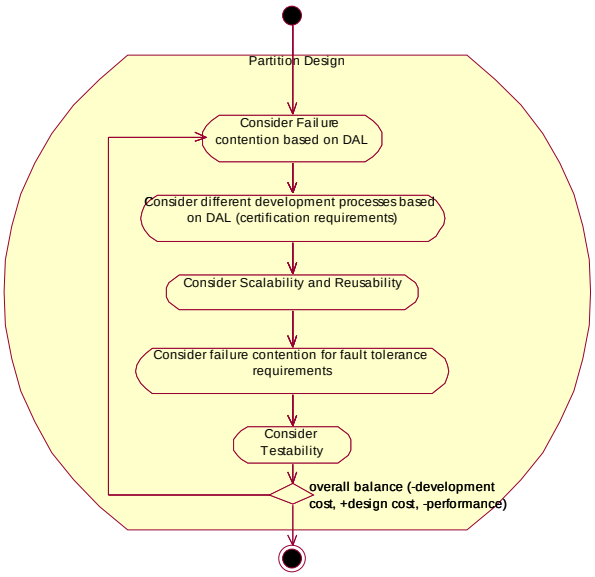

Figure 1: Iterative Decision Flow to Balance the Partitioning Design.

The cost of a re-design of the partitioning architecture, in terms of modification of interfaces (APEX ports) and distribution of functionality, could be, in many cases, quite huge or just unacceptable. So, as soon as the partitioning design is jointly checked with the application software architecture, the reduction in the risk of a bad design is mitigated.

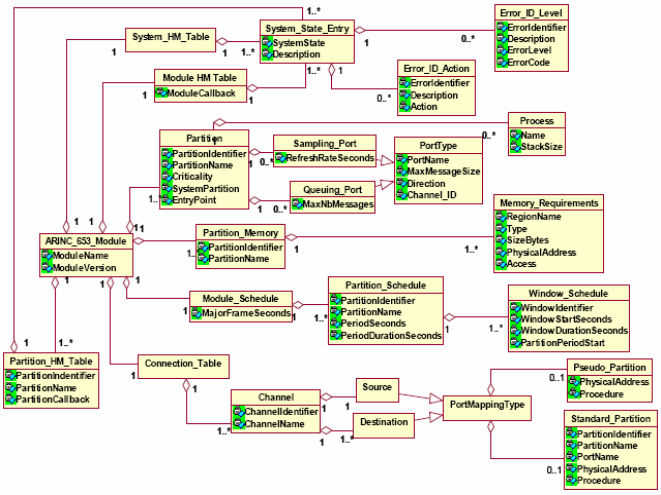

Figure 2: UML Representation of XML Schema Elemnt Relationship.

3. UML Extension for supporting ARINC 653.

UML/ARINC-653 profile has been created based on "XML Schema Element Relationship" (fig 5.2-1 in [1]), shown in Figure 2, and on "Service Requirements", (chapter 3.0 in [1]). This extension has been built for UML 1.4 [3]. The main difference in doing it for UML 2.1 [4] would be that stereotypes for ARINC-653 communication resources would apply to UML Ports instead of to Classes.
The semantics of most of stereotypes and tags are the same as stated in [1]. Some new information has been added it order to support the actual OS implementation specificities (depending on the OS supplier), and to relate ARINC-653 entities with logical software entities.

3.1 UML Extension Stereotypes Description.

The following tables list the most relevant stereotypes and the related information defined for the profile.

Stereotype	**Base Class**	**Tags**
Module 	Class	ModuleName ModuleVersion TargetType*
Tag	**Type**	**Multiplicity**
ModuleName	NameType	[0..1]
ModuleVersion	NameType	[0..1]
TargetType	Enumeration of: OsManufacturer	[1]

Table 1: Module Stereotype Properties.

**Target Type* represents the specificities (and deviation from the standard) introduced by OS manufacturers.

Stereotype	**Base Class**	**Tags**
Partition 	Class	PartitionIdentifier PartitionName Criticality SystemPartition EntryPoint
Tag	**Type**	**Multiplicity**
Partition Identifier	Integer	[1]
PartitionName	NameType	[0..1]
Criticality	Enumeration of: LEVEL_A, LEVEL_B, LEVEL_C, LEVEL_D, LEVEL_E	[1]
SystemPartition	Boolean	[1]
Entry Point	NameType	[1]

Table 2: Partition Stereotype Properties.

Stereotype	**Base Class**	**Tags**
Pseudo Partition 	Class	Name PhysicalAddress Procedure
Tag	**Type**	**Multiplicity**
Name	NameType	[0..1]
PhysicalAddress	Integer	[0..1]
Procedure	NameType	[0..1]

Table 3: Pseudo Partition Stereotype Properties.

Stereotype	**Base Class**	**Tags**
EntryPointClass*	Dependency	

Table 4: Entry Point Class Stereotype Properties.

<<*EntryPointClass*>> dependency relates a <<Process>> with the Logical class that defines the EntryPoint method for the Process.

Stereotype	**Base Class**	**Tags**
Process 	Class	Name StackSize BasePriority Period TimeCapacity DeadLine EntryPoint Preamble*
Tag	**Type**	**Multiplicity**
Name	NameType	[1]
StackSize	Integer	[0..1]
BasePriority	Integer	[1]
Period	Float	[1]
TimeCapacity	Float	[0..1]
DeadLine	Enumeration of: Soft, Hard	[0..1]
EntryPoint	String	[1]
Preamble	String	[0..1]

Table 5: Process Stereotype Properties.

Preamble represents a method called before *EntryPoint* for specific partition initialization.

Following stereotypes are focusing on the **Inter-Partition Communications** modelling.

Stereotype	**Base Class**	**Tags**
QueuingPort 	Class	PortName MaxMessageSize Direction MaxNbMessages
Tag	**Type**	**Multiplicity**
PortName	NameType	[1]
MaxMessageSize	Integer	[1]
Direction	Enum. of: Source Destination	[1]
MaxNbMessages	Integer	[1]

Table 6: Queuing Port Stereotype Properties.

Stereotype	Base Class	Tags
SamplingPort 	Class	PortName MaxMessageSizeDirection RefreshRateSeconds
Tag	**Type**	**Multiplicity**
PortName	NameType	[1]
MaxMessageSize	Integer	[1]
Direction	Enum. of: Source Destination	[1]
RefreshRateSeconds	Float	[1]

Table 7: Sampling Port Stereotype Properties.

Stereotype	Base Class	Tags
Channel 	Class	ChannelIdentifier ChannelName
Tag	**Type**	**Multiplicity**
ChannelIdentifier	Integer	[1]
ChannelName	NameType	[0..1]

Table 8: Channel Stereotype Properties.

Stereotype	Base Class	Tags
ConnectionTable	Class	

Table 9 : Connection Table Stereotype Properties.

Once the Inter-Partition Communications are considered, the next stereotypes are related to the **Intra-Partition Communications**.

Stereotype	Base Class	Tags
Semaphore 	Class	Name MaximumValue Queuing Discipline
Tag	**Type**	**Multiplicity**
Name	NameType	[1]
MaximumValue	Integer	[1]
Queuing Discipline	Enum. of: FIFO, Priority	[1]

Table 10: Semaphore Stereotype Properties.

Stereotype	Base Class	Tags
Event 	Class	Name
Tag	**Type**	**Multiplicity**
Name	NameType	[1]

Table 11: Event Stereotype Properties.

Stereotype	Base Class	Tags
Buffer 	Class	Name MaxMessageSize MaxNbMessage Queuing Discipline
Tag	**Type**	**Multiplicity**
Name	NameType	[1]
MaxMessageSize	Integer	[1]
MaxNbMessage	Integer	[1]
Queuing Discipline	Enum. of: FIFO, Priority	[1]

Table 12: Buffer Stereotype Properties.

Stereotype	Base Class	Tags
BlackBoard 	Class	Name MaxMessaseSize
Tag	**Type**	**Multiplicity**
Name	NameType	[1]
MaxMessageSize	Integer	[1]

Table 13: BlackBoard Stereotype Properties.

Health Monitoring stereotype are also described below.

Stereotype	Base Class	Tags
System_HM_Table	Class	

Table 14 : System HM Table Stereotype Properties.

Stereotype	Base Class	Tags
Module_HM_Table	Class	ModuleCallback
Tag	**Type**	**Multiplicity**
ModuleCallback	NameType	[0..1]

Table 15 : Module HM Table Stereotype Properties.

Stereotype	Base Class	Tags
Partition_HM_Table	Class	PartitionIdentifier PartitionName PartitionCallback
Tag	**Type**	**Multiplicity**
PartitionIdentifier	Integer	[1]
PartitionName	NameType	[0..1]
PartitionCallback	NameType	[0..1]

Table 16 : Partition HM Table Stereotype Properties.

The **Memory** requirements are also stereotyped as follows.

Stereotype	Base Class	Tags
Memory Requirements 	Class	RegionName Type SizeBytes PhysicalAddress Access
Tag	**Type**	**Multiplicity**
RegionName	NameType	[0..1]
Type	String	[1]
SizeBytes	Integer	[1]
PhysicalAddress	Integer	[1]
Access	String	[1]

Table 17 : Memory Requirements Stereotype Properties.

Stereotype	Base Class	Tags
PartitionMemory 	Class	PartitionIdentifier PartitionName
Tag	**Type**	**Multiplicity**
Partition Identifier	Integer	[1]
PartitionName	NameType	[0..1]

Table 18 : Partition Memory Stereotype Properties.

The **Schedule** or time constraints stereotypes are also considered.

Stereotype	Base Class	Tags
ModuleSchedule	Class	MajorFrameSeconds
Tag	**Type**	**Multiplicity**
MajorFrameSeconds	Float	[1]

Table 19 : Module Schedule Stereotype Properties.

Stereotype	Base Class	Tags
PartitionSchedule	Class	PartitionIdentifier PartitionName PeriodSeconds PeriodDuration-Seconds
Tag	**Type**	**Multiplicity**
PartitionIdentifier	Integer	[1]
PartitionName	NameType	[0..1]
PeriodSeconds	Float	[1]
PeriodDuration Seconds	Float	[1]

Table 20 : Partition Schedule Stereotype Properties.

Stereotype	Base Class	Tags
WindowSchedule 	Class	WindowIdentifier WindowStartSeconds WindowDuration-Seconds PartitionPeriodStart
Tag	**Type**	**Multiplicity**
WindowIdentifier	Integer	[1]
WindowStart Seconds	Float	[1]
WindowDuration-Seconds	Float	[1]
PartitionPeriodStart	Boolean	[1]

Table 21: Window Schedule Stereotype Properties.

3.2 Modelling pattern for UML/ARINC 653.

Based on this UML extension for ARINC-653, we propose a SW Design in two planes: a conventional design to deal with the logical architecture of the solution, based on elements from the application domain, and a component design based on ARINC-653 elements. Both planes are connected in specific points, which are interface realizations and Process/EntryPoint. This approach allows trying out and evaluating different implementation solutions while keeping stable the logical architecture.

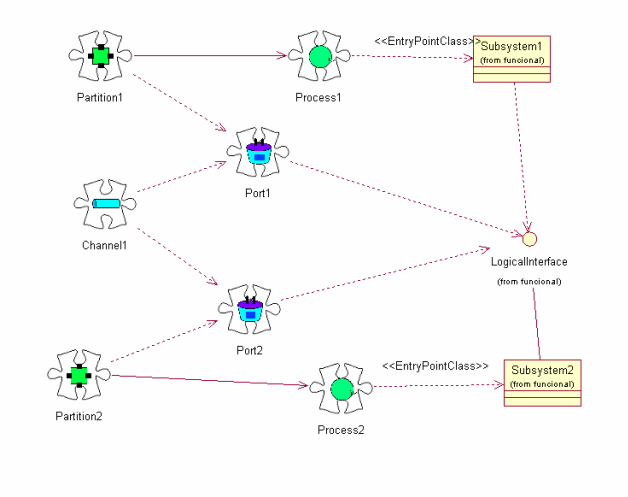

Figure 3: Example. Multi-Partition Solution.

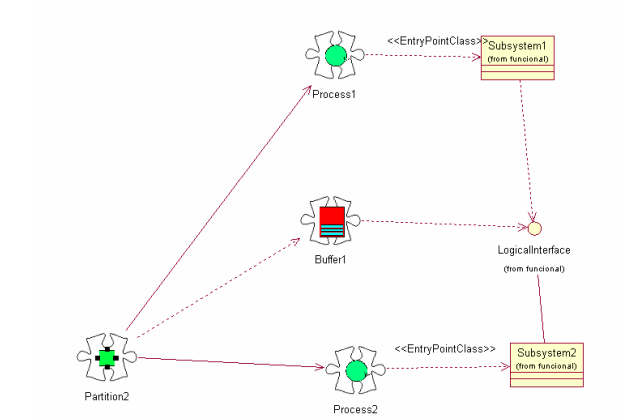

Figure 4: Example Alternative Intra-Partition/Multi-Process solution.

From the logical design point of view, there is complete independence about the actual mechanism used to distribute, to run and to communicate the software components. Even the code derived from logical design is independent of the component plane supposed, on the one hand, that the interfaces act like wrappers of the concrete implementation of the communication mechanism, which depends on the component design and, on the other hand, that active logical entities are not aware about where the execution thread comes from.

In the other side, it is important to take into account the nature of communication mechanisms supported by ARINC-653 resources (asynchronous, blocking-timed-out, unidirectional…) when designing the logical solution, particularly the interfaces definition, to guarantee the feasibility of getting advantage of flexibility in the mapping with the component plane.

Moreover, the policy for realization and usage of interfaces in the logical plane (polling, call-back registration…) must be defined before logical detailed design and implementation, because it can condition detailed design decisions and because it

will became a requirement for the interface wrapper realization.

From other point of view, the fact of having a formal-integrated model for the solution, allows verifying the integral coherence of the whole design, by taking into account both the logical architecture (coming from the application domain), and the ARINC-653 services configuration and usage. Even due to the formality on the description, much correctness verification can be done automatically.

Besides, the integrated model allows making time analysis, by considering the performance constrains at logical plane (time and memory) in front of the ARINC-653 services configuration and behaviour (partition scheduling configuration, communication performance and memory configuration).
Performance and Time constrains specification could be defined by using [5] or [6].

4. ARINC 653 XML Configuration Schemas.

As described in the ARINC 653 Specification Part 1 [1], an ARINC 653 system is configured using "***Static Data Areas accessed by the OS***".
The ARINC 653 specification uses XML to describe such configuration data and provides an XML Schema reference.
This schema is the start point of a reference process provided in the ARINC 653 standard to suggest an approach usable by application Developers and System Integrators to generate system configuration tables. The different steps include generating an XML instance, validate it and translate it to the Operating System required format.
The VxWorks 653 Platform provided by Wind River follows this reference process. In particular, it includes DO-178B Qualified Verification and Development tools respectively usable to Validate and Translate the XML instance into a binary version representing the Configuration table of the System.

Wind River's return of experience on this process, in both Development and Certification usage, has shown restrictions in the proposed ARINC 653 XML schema preventing:

- Re-usability
- Independency

These restrictions include:

- HM Table / Partition relationship.
  In the ARINC 653 standard, the HM Table reference a Partition, this implies that each table must be unique and cannot be re-used for several Partitions. Wind River proposed Schema reverses this relationship.
- Window Schedules are grouped per Partition to define the Module Schedule. In result, a change to partition schedule affects the entire Module schedule.
  It is also hard to identify the overall schedule and schedule conflicts.
  In Wind River's proposed schema the Module Schedule is represented in a list of Window Schedule linked to the corresponding Partition and not the opposite.

This XML Schema definition is the return of experience of Wind River and its customers, which includes hosting Flight Management, Data Management or Mission Computer applications on the same system in both civil and military aircrafts.

This Schema has been proposed as a start point for the ***ARINC 653 XML Definition Subcommittee***, led by Wind River, and counting for ARINC 653 Part 1 Supplement 3.
This XML Schema definition has been used as reference for the UML profile definition.

5. Automatic Code Generation of ARINC 653 Artefacts.

One important advantage of formal models is to get the possibility of generating automatically some code, which is typically routine, tedious, and often error prone. In our scenario, it is not only one advantage, but also a requirement for making feasible the feature of trying out easily (in an efficient way) different architecture designs.
At least, we consider beneficial the automatic generation of code for ARINC-653 configuration tables, interfaces implementation (wrapper and body), partitions procedures and processes procedures.
Anyway, the activity of code generation should be flexible enough to support different possible scenarios:

- With respect to the source code, some aspects can differ from a project to other.
  - Coding language.
  - Safety level, and associated implementation constrains.
  - Policy for usage/realization of interface wrappers, etc.
- With respect to the ARINC-653 configuration tables, there can be specificities and even standard deviations, depending on the OS manufacturer.

So, the activity of code generation must take into account the architecture information and the pattern applicable for the output.

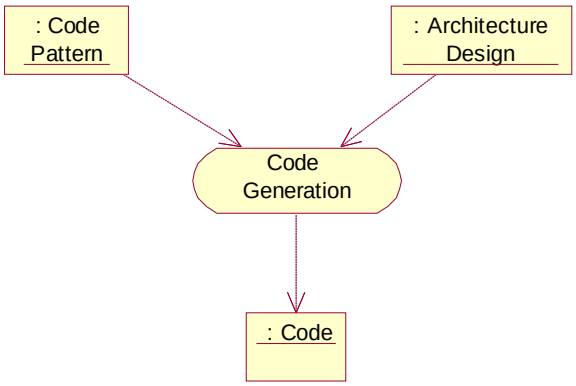

Figure 5: Code Generation Dependencies.

6. Qualified Validation of the ARINC 653 Configuration Tables.

The generated Configuration Tables can now be validated against the reference schema.

This is performed thru the DO-178B Qualified Verification tool VerIMAx Checker which verifies not only the consistence with the XML Reference Schema but also from a Module point of view:

- Consistence of APEX Channel definitions
- Memory overlapping
- Missing Partition in Schedule
- Etc.

After Validation, the Configuration Tables can be directly translated into a binary format understandable by the Operating System using the DO-178B Qualified Development VerIMAx Compiler.

This Validation and Translation process can be represented by the slide on Figure 6.

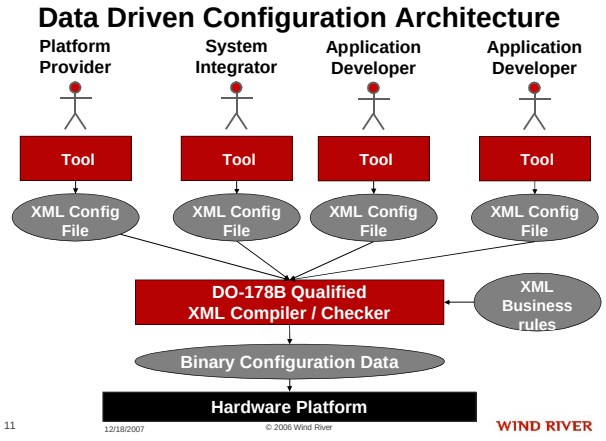

Figure 6: Validation and Translation Process using VerIMAx.

7. Application of the Proposed Extension to the Partition Testing.

One main advantage of using ARINC-653 (IMA) is the increase of testability at partition level, because of the clearly defined interfaces and the well defined and controlled partition execution environment (time and memory).

Once reached this point, it is a temptation to trying to take advantage from the ability of code generation described in chapter 5 (mainly relating with interfaces and configuration tables), as well as of the formalism required for defining the external partition interfaces (interface wrapper and realization/usage policy). So, a generic test framework solution, could hook with partition code, just by complementing the partition under test; that is, by realizing the interfaces required by the partition, using the interfaces realized by the partition, and respecting the budget of time and memory used by the partition.

As long as the required partition behaviour has been defined by means of formal UML behaviour diagrams, the test case could be described by using instances of Sequence diagrams extracted from State chart diagrams and Sequence diagrams (for checking requirements about sequencing and time constrains) with concrete data values (for checking requirements about algorithms and data transformation).

8. Return of Experience.

The technology described along this paper has been defined, implemented and successfully used in the framework of the Advanced Air Refuelling Boom System Project developed by the Military Transport Aircraft Division of EADS-CASA.

The UML profile has been applied in an actual ARINC 653 system created, tested and validated using Wind River's VxWorks 653 OS. It implements a separation in two layers at the Software Design Level, which allows multiple capabilities :

- Evaluation of different implementations keeping the Logical solution stable.
- Verification of the coherence of the whole design and application time analysis.
- Easy creation of test stubs for code running inside an ARINC 653 partition.

The return of experience shows easy modelling of an ARINC 653 approach with optimal results and important reduction of the coding time and the proportion of coding errors.

Configuration and development processes are also *key factors* for successful certification

- Time and integration are where the challenges are, not the applications, the OS or the hardware

Special *emphasize* should be put on these two areas from the start of a program. Both areas need to be carefully *designed*.

The right *balance* needs to be found between *built-in* configuration and *data driven* configuration for all parts of a system

- Data driven configuration allows changes to be bring in with minimal impact, even during the certification cycle

Use a *unified* and *qualified* process to manage all data driven configuration data

9. Conclusion

The presented ARINC 653 UML profile optimally extends standard UML for supporting partitioning design and allows connect it with a classical software component based design. The use of this new profile provides some clear advantages, as flexibility, feasibility, verifiability, reusability, etc., widely described along the paper. Besides, automatic code generation of both ARINC 653 elements and configuration tables is also provided, which increase the reliability and the reduction of effort and errors cost. Related to the configuration tables of an ARINC 653 the followed XML schemas policy has been described and the main advantages of its use have been also highlighted. Taking into account the application of all these concepts to Safety-Critical Systems, the certification point of view has been also considered. Special interest has been put in the qualified validation process of the mentioned ARINC 653 configuration tables, which allows directly translate them to binary. Finally, a feasible and reliable testing framework has been also proposed mainly focused to the partition testing.

10. Acknowledgement

Special thanks to the Military Transport Aircraft Division of EADS-CASA for the support during the development of this paper and for providing the frame of a concrete project environment, in which this technology has been developed.

Thanks to Thierry Preyssler, Wind River expert, leading the *ARINC 653 XML Definition Subcommittee*, for his input on the ARINC 653 XML Configuration Tables.

11. References

[1] AEEC: "*ARINC Specification 653. Avionics Application Software Standard Interface.*", ARINC, 1997.

[2] P. Parkinson & L. Kinnan: "*Safety-Critical Software Development for Integrated Modular Avionics.*", Wind River, 1997.

[3] OMG: "*UML 1.4 Specification*", www.omg.org, September 2001.

[4] OMG: "*UML 2.1 Specification*", www.omg.org, August 2007.

[5] OMG: "*UML Profile for Schedulability, Performance and Time*", www.omg.org, January 2005.

[6] OMG: "*UML Profile for Modelling and Analysis of Real-time and Embedded Systems (MARTE)*", www.omg.org, August 2007.

12. Glossary

APEX:	APplication/EXecutive
ARINC:	Aeronautical Radio Inc
DAL:	Design Assurance Level
HM:	Health Monitor
IMA:	Integrated Modular Avionics
Partition:	A container for an application preventing interference with other applications an the Module OS of an ARINC 653 system
UML:	Unified Modelling Language
XML:	eXtensible Markup Language