# IMA Compliant Product Line

Dipl.-Ing. Marko Beutler

# 1 Introduction

This white paper gives an overview of an IMA compliant product line.

IMA systems currently represent the most advanced realizations of embedded Avionics. The key benefits are standardized interfaces and separation between application and system functionality.

Product lines are used to manage variations in software systems. There is a separation between product and domain engineering and its building blocks are designed for reuse.

Both approaches have their sources in the early '90s and have been applied in software systems in many cases. ADS developed a concept that combines the advantages. Its effectiveness has been proven by the ADS IMA demonstrator.

# 2 Scope

ADS already published a product line concept (see [SSPL]) and an architecture framework (see [CAPES$^2$]). This paper now describes the application of both concepts to an IMA system.

The prototype is based on the IMA standard [ARINC 653]. The concept is however flexible enough to be easily adaptable to another standard like [STANAG 4626].

Systems within scope are embedded control systems for Avionic equipment. They typically consist of a set of devices e.g. for communication or navigation, a user interface for input and output and a number of network connections. Those systems tend to have a high number of variants during lifecycle.

# 3 Concept

## 3.1 Design Process

In a product line, the features and their relations are described in a feature model. Likewise, there is a product model to describe product variants by selecting a set of features. Feature and Product Model are the output of the product engineering process.

Domain engineering encompasses the realization of features. In the ADS concept, this is done by plugins. Plugins consist of an interface description (ports, processes) and a corresponding implementation.

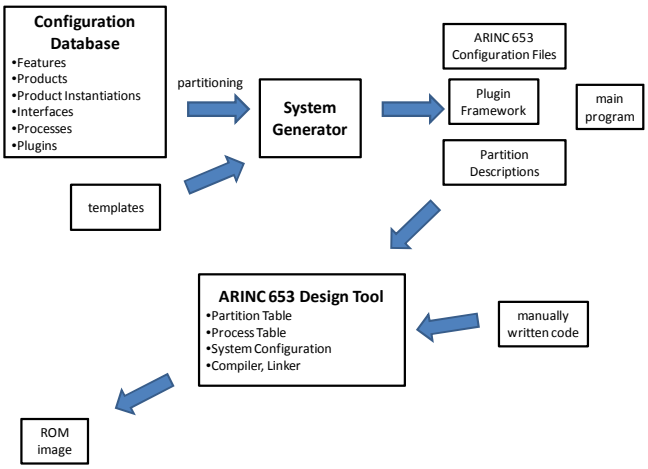


**Figure 1: Design Process**

When starting to create a new product, the interface descriptions of the contained plugins are extracted from the ADS database and the configuration on partitions is defined. The ADS system generator then produces the partition description in an exchangeable format. If the plugin implementations exist already, they are also extracted from the data base. Otherwise source code fragments for manual coding are generated.

Further processing depends on the customer selected IMA O/S and the corresponding design tools. After importing the partition descriptions, all additional partition definitions are made and the manually coded files are imported. Final output is a loadable ROM image file.

## 3.2 Software Architecture

The software architecture is based on a layers pattern.

The microkernel and hardware layers are O/S specific and not directly accessed by any plugin code. Plugins mainly use the APEX and the system software layers. The first one is specific for each partition according to the interface and process descriptions. Application partitions may only access the APEX layer, whereas system partitions can also access the system software layer.

In order to separate plugin code from system specific services, the ADS abstraction layer has been introduced. The abstraction layer is the communication and process interface for plugins.

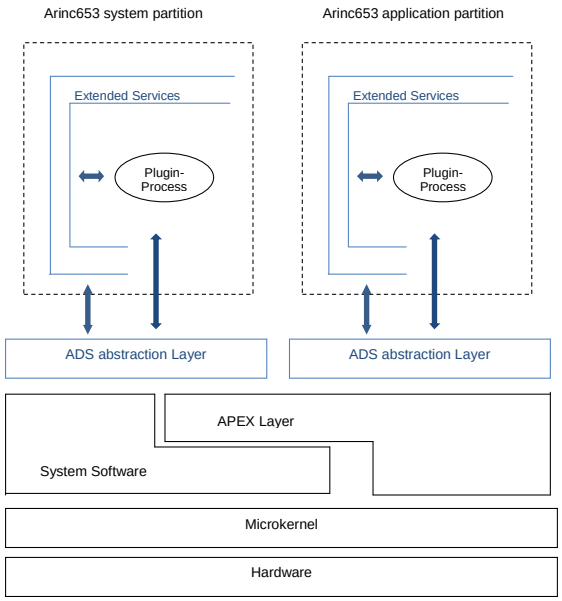


Figure 2: Software Architecture

**System Software**

The system software is specific for the underlying O/S. It is used by system partitions for all kinds of services that are not contained in the APEX layer. Among those are services are file access and networking which are typically used by device drivers.

**APEX Layer**

The APEX layer is generated by the ARINC 653 design tool. It contains access procedures for all kinds of APEX services, e.g. buffers, blackboards, queuing messages, sampling messages, events and process control. As those services are specific for each partition, each partition has its own APEX layer.

**ADS Abstraction Layer**

There are several responsibilities of the ADS abstraction layer. The most important one is to decouple plugin code from system specific services of the system software layer. This allows for easily exchanging the underlying O/S. In that case, only the manually written part of the abstraction layer needs to be adapted. The plugin code remains unchanged.

The ADS abstraction layer is also used to separate plugins from the APEX interface. It can be used to decouple plugins from ARINC 653 and thus to replace this API by STANAG 4626 APOS. Or it can be used to decouple plugins from different evolution stages of the same standard or different realizations of the extended ARINC 653 services. The latter aspect is again a system specific behavior that should not be reflected in the plugin code.

A third responsibility of the abstraction layer is to provide an extended interface for testing activities. For testing on module level, the APEX interface is used without system software and O/S. In that case, a data element and the corresponding counterpart for each service are required. If the partition uses e.g. a queuing message in send mode, the abstraction layer for module test would also contain the queue itself and a service to receive the message. This service would be used by the test driver.

Operation calls (cyclic and acyclic) from the O/S are forwarded by the abstraction layer. This can be used for variant handling. In the design tool, only a single operation with a certain period is registered, namely the one of the abstraction interface. The abstraction interface then forwards this cyclic call to a variant dependent set of plugin operations.

The APEX dependent part of the abstraction layer, the extensions for module tests and the invoker for operation calls can be generated from the partition descriptions. The other parts are manually written.

**Extended Services**

Extended services cover recurring activities. Typical examples are libraries for queues and numeric conversion or timer functionality.

## 3.3 IMA Architecture

The following model shows the relations between features and the main components of an IMA system.

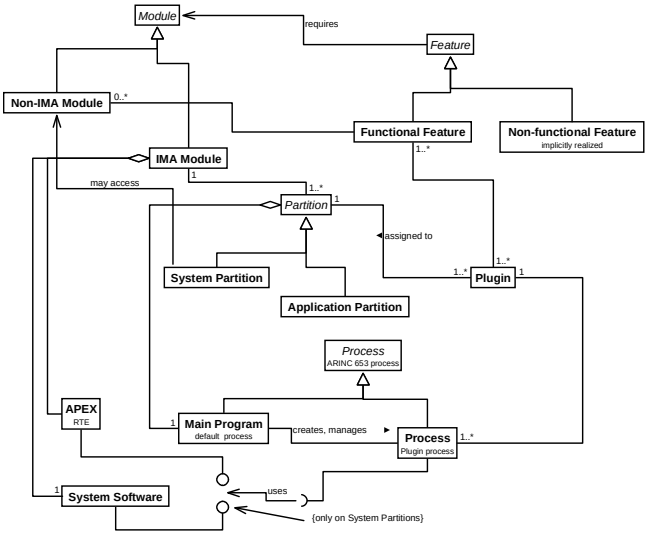


Figure 3: IMA Architecture

The system functionality is described by features. Features can be subdivided into functional and non-functional ones. Non-functional features describe e.g. quality aspects like performance or maintainability and are realized implicitly by the system components. Functional features are realized by plugins and may

have a relation to a non-IMA module. Radio functionality could e.g. be realized by two plugins (a radio OPF and a device driver) and the radio device itself which is a non-IMA module.

Plugins are assigned to partitions. Each partition can host one or more plugins. Partitions can represent application or system partitions. Application partitions are used for operational functionality whereas system partitions are used for device drivers that require access to the system software.

Partitions are allocated to IMA modules. Each IMA module can host one or more partitions, depending on the available resources and the needs of the partitions. Partitions are isolated from each other in space and time.

Plugins consist of one or more processes. Each partition also has one additional process, the main program. The main program is the only process that is directly created by the O/S and run at system startup. The main program then creates all other APEX processes and starts them.

Processes may use the APEX communication interface and system software services. The latter possibility is however only given within system partitions. Communication to non-IMA modules is also realized by system software services.

# 4 Evaluation

The described concept is very flexible as varying aspects are separated from each other by design:

- Product engineering separated from domain engineering
- Independent from specific IMA O/S
- Adaptable to different IMA standard and evolutions
- Separation of operational and device specific functionality
- Easily extendable data model and (asset) database
- Decoupling of plugin interface and implementation
- Support of different modes (operational/test mode).

The software architecture and the product line are highly configurable on different levels of product instantiation.

The system functionality is realized by small building blocks – plugins – that are easily manageable and rather independent from each other. High cohesion with plugins and low coupling between them is one prerequisite for scalability. The other prerequisite is the usage of an appropriate data model and a high degree of automation.

Partitions are independent from each other and can be verified and validated separately. This opens the path for advanced qualification concepts like incremental qualification.

# 5 Summary

The white paper describes an IMA compliant system that realizes the [SSPL] product line concept based on the [CAPES$^2$] architecture. Varying system aspects are separated from each other by design. Modelling is done based on an appropriate data model and automation techniques like code generation are applied. The system is highly flexible, configurable and scalable.

The feasibility of the concept has been proven by the ADS IMA demonstrator.

# 6 References

[SSPL]
ADS-Whitepaper “System Software Product Line”

[CAPES$^2$]
Configurable Avionic Platform for Embedded System & Software

[ARINC 653] AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE PART 1 – 3

[STANAG 4626] NATO MAS STANAG 4626 (DRAFT 1) – MODULAR AND OPEN AVIONICS ARCHITECTURES, PART I – VI