Dissertation

A Distributed Platform for Integrated Modular Avionics

ausgeführt zum Zwecke der Erlangung des
akademischen Grades eines

Doktors der technischen Wissenschaften

unter der Leitung von

Univ.Doz. Dr Stefan Poledna
Institut für Technische Informatik 182/1

eingereicht an der

Technischen Universität Wien,
Fakultät für Informatik

durch

Roland Wolfig
MNr: 0327070
Urhausweg 1
2560 Neusiedl

Wien, im Mai 2008

Abstract

Integrated Modular Avionics (IMA), especially Distributed Integrated Modular Avionics (DIMA), and Modular Certification are widely discussed approaches in the aerospace community at the moment. IMA deals with the idea of sharing hardware resources and integrating several aircraft functions into one hardware unit, using a modular architectural approach, to reduce weight, space, cabling, power consumption and costs.

In difference to such, already established, IMA systems, DIMA architectures provide more flexibility to the used hardware, which does not necessarily have to be in a single box. It may be split-up into several smaller hardware units, distributed all over the aircraft connected by a safety-critical communication system.

Modular Certification uses the modularity, provided by IMA/DIMA systems to split their certification into several parts. In combination with efficient certification, based on optimized processes, development time and effort are reduced.

Based on these prerequisites, the concept of a distributed and integrated platform solution (DIPS) is introduced. This concept, based on a DIMA architecture, defines the constraints and services needed for the implementation of a modular certifiable and flexible platform, which, in combination with the hosted applications, is able to handle all safety-critical functions in an aircraft by providing data exchange between and encapsulation of different modules.

This thesis describes such a platform approach, discusses its attributes and certification demands, identifies its requirements and constraints and considers its business opportunities and future applications.

Kurzfassung

Integrierte modulare Luftfahrtsysteme (IMA), im speziellen verteilte integrierte modulare Luftfahrtsysteme (DIMA) und modulare Zertifizierung sind vielversprechende Themen, welche derzeit umfassend diskutiert werden. IMA behandelt die Idee gemeinsam genutzter Hardware Ressourcen und die Möglichkeit, mehrere verschiedene Funktionen auf einer Hardwareeinheit zu vereinen. Basierend auf einem modularen Architekturansatz können so Gewicht, Platz, Verkabelung, Leistung und Kosten optimiert werden.

Im Gegensatz zu diesen, bereits eingesetzten, IMA Systemen, ermöglichen DIMA Systeme größere Flexibilität, da die verwendeten Ressourcen nicht an der selben Stelle platziert werden müssen, sondern aufgespaltet und überall im Flugzeug verteilt werden können, wobei die einzelnen Einheiten mit einen sicherheitskritischen Kommunikationsnetzwerk verbunden werden.

Modulare Zertifizierung verwendet die Eigenschaften solcher Architekturen, um die Aufgabe in mehrere Teile zu spalten. Dieser Ansatz, kombiniert mit optimierter Zertifizierung, erlaubt auch in diesem Bereich die Kosten zu reduzieren.

Basierend auf diesen Voraussetzungen wird das Konzept einer verteilten und integrierten Plattform eingeführt, welche alle Voraussetzungen bietet, um gemeinsam mit den eigentlichen Anwendungen alle computergesteuerten Funktionen in einem Flugzeug auszuführen.

Diese Dissertation beschreibt solch eine Plattform, diskutiert ihre Eigenschaften und Zertifizierungserfordernisse, identifiziert ihre Anforderungen und Einschränkungen und behandelt ihre wirtschaftlichen Aspekte beziehungsweise ihre zukünftigen Anwendungsgebiete.

Danksagung

Diese Arbeit wurde nur durch die Unterstützung einiger Personen möglich, denen ich auf diesem Weg meine Dankbarkeit aussprechen möchte.

Zuerst möchte ich mich bei Prof. Hermann Kopetz bedanken, der es mir ermöglicht hat am Institut für Technische Informatik diese Dissertation zu schreiben. Desweiteren gilt mein besonderer Dank Dr. Stefan Poledna, der diese Arbeit betreut und mir mit Rat und Tat zur Seite gestanden ist. In weiterer Folge möchte ich mich auch bei der Firma TTTech bedanken in welcher ich während der Zeit in der diese Dissertation entstanden ist arbeiten durfte und dadurch nicht nur jede Menge Erfahrungen sammeln konnte, sondern mir auch der direkte und praktische Zugang zu diesem Themenbereich ermöglicht wurde. Im Speziellen möchte ich mich bei DI Martin Schwarz bedanken, der mir durch unsere Diskussionen einen tiefen Einblick in die Thematik ermöglicht hat.

Im privaten Bereich möchte ich mich bei DI(FH) Robin Ehfrank bedanken, der mir immer als Freund zur Seite gestanden ist und mir in unseren Gesprächen und Diskussion immer eine wertvolle Hilfe war.

Ganz besonderer Dank gilt Dr. Alexandra Thurner, die mich nicht nur mit ihrer Erfahrung und ihrem Wissen unterstützt hat, sondern mir auch immer eine moralische Stütze war und mir immer wieder neue Kraft gegeben hat.

Schließlich möchte ich meiner Familie meine tiefste Dankbarkeit dafür aussprechen, dass sie mir stets Rückhalt gegeben und mir dies alles erst ermöglicht hat.

Contents

List of Figures

List of Tables

Terminology

The following terms are used in the thesis and are essential for consistent understanding of the topic.

- Aircraft Function - A set of hardware and software modules, which together provide the wanted avionics functionality (e.g. flight control system, autopilot, power distribution, etc.).
- Application - Software with a defined set of interfaces that performs a function.
- Architecture - The architecture provides the theoretical environment of a platform. It considers services, interfaces, topologies, requirements, constraints, and integration and implementation details.
- Component - A self-contained hardware part, software part, database, or combination of them. A component does not provide an aircraft function by itself.
- Core Communication System - The core communication system is the central component of a distributed platform. It is a safety-critical, high-speed connection between nodes and has to provide several basic platform services.
- Core Software - The operating system and support software that manage resources to provide an environment in which applications can be executed. Core software is a necessary component of a platform and is typically comprised of one or more modules.
- Host (Processor/CPU) - The host is the processing element which executes the core software and the application. In connection with a communication interface, it is a node of a system.

- (D)IMA System - Consists of a platform and a defined set of hosted applications.
- Module - A component or collection of components that may be software, hardware, or a combination of hardware and software, which provides resources to the hosted applications. Modules may be distributed across the aircraft or may be co-located.
- Node - A node consists of the host CPU and the communication interface.
- Partitioning - An architectural technique to provide the necessary separation and independence of functions or applications to ensure that only intended coupling occurs.
- Platform - Module or group of modules, including core software, hardware and communication that manages resources to support at least one application. Platforms, by themselves, do not provide any aircraft functionality. The platform is the implementation of an architecture which establishes a computing environment, support services, and platform-related capabilities, such as health monitoring and fault management. The platform can be certified independently from hosted applications.
- Reusable - The design assurance data of previously accepted modules and applications may be used in a subsequent aircraft system design with reduced need for redesign or additional acceptance.
- Subsystem Communication System - Additional to the core communication system, low-cost subsystem communication is needed which also provides several platform services.

Where am I?

(Charles A. Lindbergh, upon arrival in Paris)

Chapter 1

Introduction

1.1 Motivation and Objectives

Cost, weight and safety are probably the most important key characteristics of aerospace development. Much effort is taken to reduce costs and weight by keeping the safety level or even increasing it. Furthermore electronic components and their features become more and more and the complexity of such systems increases too, which leads to development problems based on too complex designs.

In current planes, there are several different systems, like multimedia, power or control systems. Some of them are safety-critical and a lot of them need to share data with other systems. To be able to manage this complexity, for efficient development of such systems on the one hand, and to manage it in terms of the mental capability on the other hand, some sort of abstraction is needed. Therefore, such systems need to be separated into subsystems to reduce the overall complexity.

Several new concepts are currently under discussion to give the development of electronic systems in the aerospace domain a change. Among them, there is the concept of (Distributed) Integrated Modular Avionics (IMA/DIMA), which tries to increase efficiency by reducing hardware, and Modular Certification, which tries to cut development costs by reducing certification efforts. Both concepts use the approach of separating large systems into smaller subsystems.

1.1.1 (Distributed) Integrated Modular Avionics

The concept of (Distributed) Integrated Modular Avionics [106] deals with a modular structure of software and hardware components which are independently developed and certified. In difference to IMA, Distributed IMA [114][107] modules are spread all over the aircraft and connected by a communication system.

The composition of these distributed modules creates avionics functions which may interact with each other. This approach provides more flexibility to the system designer, reduces production costs and maintenance effort, and allows the reuse of already created modules but also creates a set of new problems in the design of aerospace applications.

One of the biggest problems is to guarantee that different aircraft functions are not able to disturb each other. To ensure this, an underlying distributed platform is needed, which provides the required services and handles the data exchange between these applications.

Another constraint is, that the integration of the pre-certified modules, which may be descended from different sources and have different levels of criticality, provides a system which can be certified according to airworthiness requirements. The Radio Technical Commission for Aeronautics (RTCA) and the European Organization for Civil Aviation Equipment (EUROCAE) addressed this demands with the SC-200/WG60 working group. This working group developed a new guideline for the use of IMA/DIMA systems, the DO-297 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations [99].

1.1.2 Modular Certification

The certification cost of a software project according to DO-178B [96] are doubled in contrast to a common software project [55]. The concept of Modular Certification approaches this problem by the possibility of splitting the certification in several parts.

In difference to the current approach, which approves only whole systems, Modular Certification verifies the correct development of single parts and their integration into the entire system. This provides a module based certification approach. The system is split up into single modules which are certified. The advantages of this approach are, that a module developer just has to care about his single module and its interfaces, but not about the whole system and its certification, on the one hand and that already developed modules, including its certification evidence, may get reused on the other hand.

Using the IMA concept named above, Modular Certification allows the independent development and certification of the software and hardware components the platform consists of. Only the integration has to be certified additionally but modules can be exchanged without the need of a complete re-certification of the system. This allows to reduce the certification costs after the second use of this module.

1.1.3 Conclusion

The objectives of this thesis are discussing the concepts of Distributed Integrated Modular Avionics and Modular Certification, illuminating their environment and creating a practical approach using these concepts. The ultimate goal is to establish theoretically a flexible, full certifiable, safety-critical platform which fulfills aerospace demands and is able to host aircraft functions of different safety classes. To achieve this idea, research regarding efficient certification is done and requirements and recommendations for the implementation of such a platform are developed.

According to this research, which is the main task of this thesis, several key factors are presented which allow to improve the development and certification process in future projects. In addition, the results are verified by a comparison to accomplished aerospace software projects. The second main task of this thesis is the question how to deal with the change of the avionics platform. Therefore, the parameters regarding the switch from current to future platforms is described and requirements and recommendations for such a problem are discussed.

1.2 Related Work

The basic constraints of establishing a safety-critical system are given by Kopetz [70]. He describes the properties of safety-relevant systems, especially for a safety-critical architecture. He continued his work by using this constraints to develop the Time-Triggered Architecture (TTA) [74].

Rushby [103] discusses Modular Certification and provides methods for formal verification of this concept. He identifies the key elements of the concept by showing how these elements can be achieved and verified. Furthermore, he discusses "Partitioning" [101], which is one of the key services of an Integrated Modular Avionics architecture.

The DO-297 [99] guideline describes the process of certification for Modular Certification. It is the result of the SC-200/WG60 working group and can be regarded the current guideline for the development of IMA systems. It gives guidance for development and discusses additional considerations.

Decos, which is an EU funded project, has the goal to develop a distributed execution platform using partitioning as a key service. The project provides an implementation of a distributed architecture using a safety-critical communication system.

This related work provides a baseline which will be supplemented during detailed consideration.

1.3 Structure of this Thesis

This thesis describes the requirements for a DIMA based platform and discusses several aspects of Modular Certification.

Chapter 2 addresses the architectural aspects of such a platform and provides a concluding overview regarding current concepts.

Chapter 3 discusses and compares several communication systems which can be used for platform communication, providing the base for distributed aircraft functions.

Chapter 4 defines requirements and constraints for the development of a safety-critical system architecture. It discusses the parts of such a system and gives an overview about already established architectures and new developments within this area.

Chapter 5 deals with the concept of Modular Certification. It describes current approaches for certification, gives insight in new standards and talks about processes and formal aspects. Furthermore, parameters for efficient certification are presented which are applicable for modular but also for common certification processes.

Chapter 6 combines the concepts and systems discussed before to present a so called distributed and integrated platform solution. This chapter contains requirements, recommendations and benefits regarding such an approach, which may change the way the development of aircraft functions is done.

Chapter 7 gives a prospectus about the way such a new technology can be used and describes its economical effects. Furthermore, it gives an outlook into the future and shows how the development of aerospace and also of several other domains might change.

Chapter 8 concludes this thesis with an overview about what has been achieved and what is still open for further investigation.

Trouble in the air is very rare. It is hitting the ground that causes it.

(*Amelia Earhart*)

Chapter 2

Avionics Architectures

2.1 Introduction

This section describes the way avionics systems are designed and evaluates new approaches for architecture design.

From the beginning of aircraft deployment, single functions got developed independently to provide the desired functionality. In case of electronic resources it is a widely used approach that each function has its own dedicated hardware and interfaces.

To illustrate this concept, let us assume this function is the autopilot. It has its own sensors, actuators, computational resources and displays and does not share data with any other function. The development of such a function is more or less independent and, based on the fact that no data is shared, it contains a natural fault-propagation barrier. This means that in case of faults no other function is influenced and the autopilot uses an internal fault-tolerance approach to provide the correct service.

The important point in this example is the fault-propagation barrier based on the dedicated hardware and interfaces. This approach has advantages on the one hand but also leads to a set of different computer systems in an aircraft which consume costs, weight and power.

Another concept which is already used in the aerospace domain is the model of Integrated Modular Avionics (IMA) systems. These systems use a different approach by sharing computational power for several functions. The challenge is to prevent fault-propagation between functions using the same hardware resources, which is called "partitioning". Partitioning may be considered as memory partitioning, to separate functions inside the host memory, and network partitioning, to separate modules in a network. The advantage is that less hardware is needed and the functions operate more integrated. This allows more convenient sharing of data which provides a reduction of interface modules.

A currently discussed concept, which is a combination of both architecture designs, is called Distributed Integrated Modular Avionics (DIMA). It describes a distributed but also integrated architecture which shares the advantages but also the disadvantages of both concepts. Based on the distribution, the fault-propagation barrier is achieved by physical separation but less hardware is needed by sharing the computational power and interfaces. On the other hand, it causes a lot of difficulties, like the communication between distributed nodes.

An advantage for both types of integrated architectures is the reduction of the certification effort caused by modularized certification evidence and the possibility of reusing these arguments.

2.2 Federated Avionics

Federated avionics [36][14] is currently a widely used concept in the aerospace domain was developed in the 1970s. This concept is based on the fact that every aircraft function was developed independently. These functions were developed even further and new functionalities were added which also used their own dedicated hardware.

Based on the increased complexity of aircraft functions, like fly-by-wire, communication between subsystems of aircraft functions was introduced with relatively little interaction between separate functions to reduce their influence on each other. This provides the advantage of function independence which ensures a natural fault-propagation barrier. A faulty function is not able to influence any other function and therefore the fault can not propagate and lead to a faulty behavior in several systems. A schematic illustration of a federated architecture design may be found in figure 2.1 on the following page.

Fault-tolerance is provided by active redundancy, which is a common approach

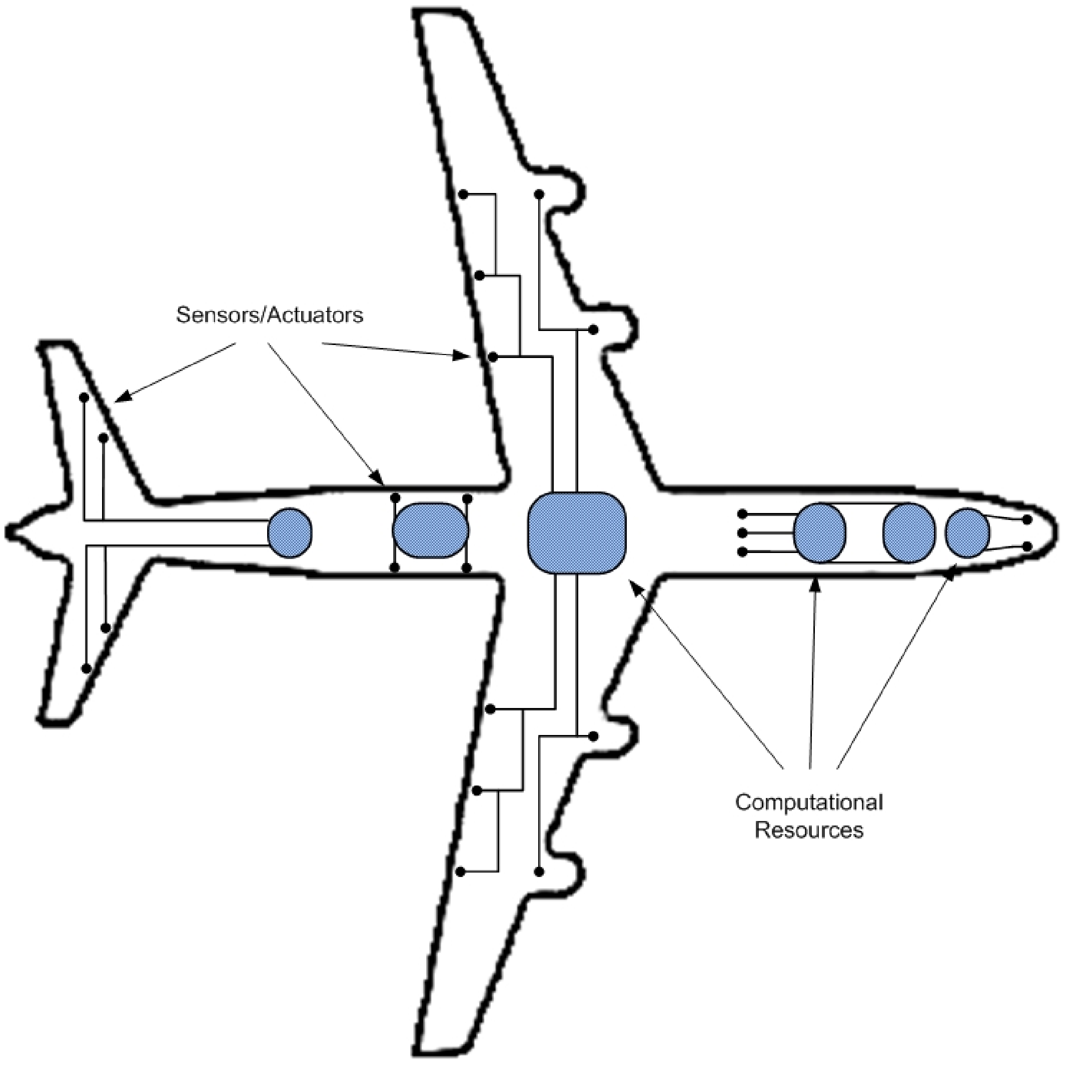

Figure 2.1: Schematic Illustration of Federated Architecture Design

for safety-critical systems, and demands dedicated hardware and, in case of distributed aircraft functions, dedicated communication channels.

A main point is that every system needs its own interfaces like sensors, actuators, displays and controls [121] which leads to a complex environment for the operator and can moreover lead to replicated hardware by several functions which are not necessary. This is a problem in the aerospace domain, where costs, space and weight are major design drivers. Therefore, federated avionics are an expensive way to map aircraft functions even if the control of complexity is easier, compared to other concepts, because different systems do no interact with each other.

A major disadvantage of this concept is a lack of flexibility. The missing interoperability between different aircraft functions reduce their efficiency. The interaction between such functions like the flight control system, the autopilot and the navigation system for example allow reducing the overall computing and interface resources and provide additional functionalities like global diagnosis, advanced flight controls and optimized fuel consumption.

Another disadvantage is the fact that small changes in a function or an upgrade, including new functionalities, may make redevelopment and recertification of large parts of an aircraft function necessary. Although, recertification in a federated approach is easier than in a modular approach, new developments and certification of single modules needs less effort if the system is already established. This is a major problem considering the operational life time of an airplane, which is about 30 years, and the fast evolution cycle in the electronics domain.

Because of these disadvantages, there was a need for new concepts which address these new requirements.

2.3 Integrated Modular Avionics

Based on the evolution of software and electronics technology, new functions are developed for aircraft. These functions provide new capabilities but also increase complexity. To be able to handle these increased requirements a new approach was necessary. Current aerospace developments, which head in this direction, are called More Electric Architecture (MEA) [57]. The goal of these approaches is to have a fully connected and modular avionics architecture which is called IMA (Integrated Modular Avionics). IMA [36][106] supports the use of high-performance computing platforms. These platforms are able to host multiple applications on a single processor or on distributed processors connected by a communication system.

A schematic illustration of a IMA architecture design can be found in figure 2.2 on the next page.

The definition of IMA which is given by DO-297 [99] gives an overview about what IMA is and how it is used:

> *IMA is a shared set of flexible, reusable, and interoperable hardware and software resources that, when integrated, form a platform that provides services, designed and verified to a defined set of safety and performance requirements, to host applications performing aircraft functions.*

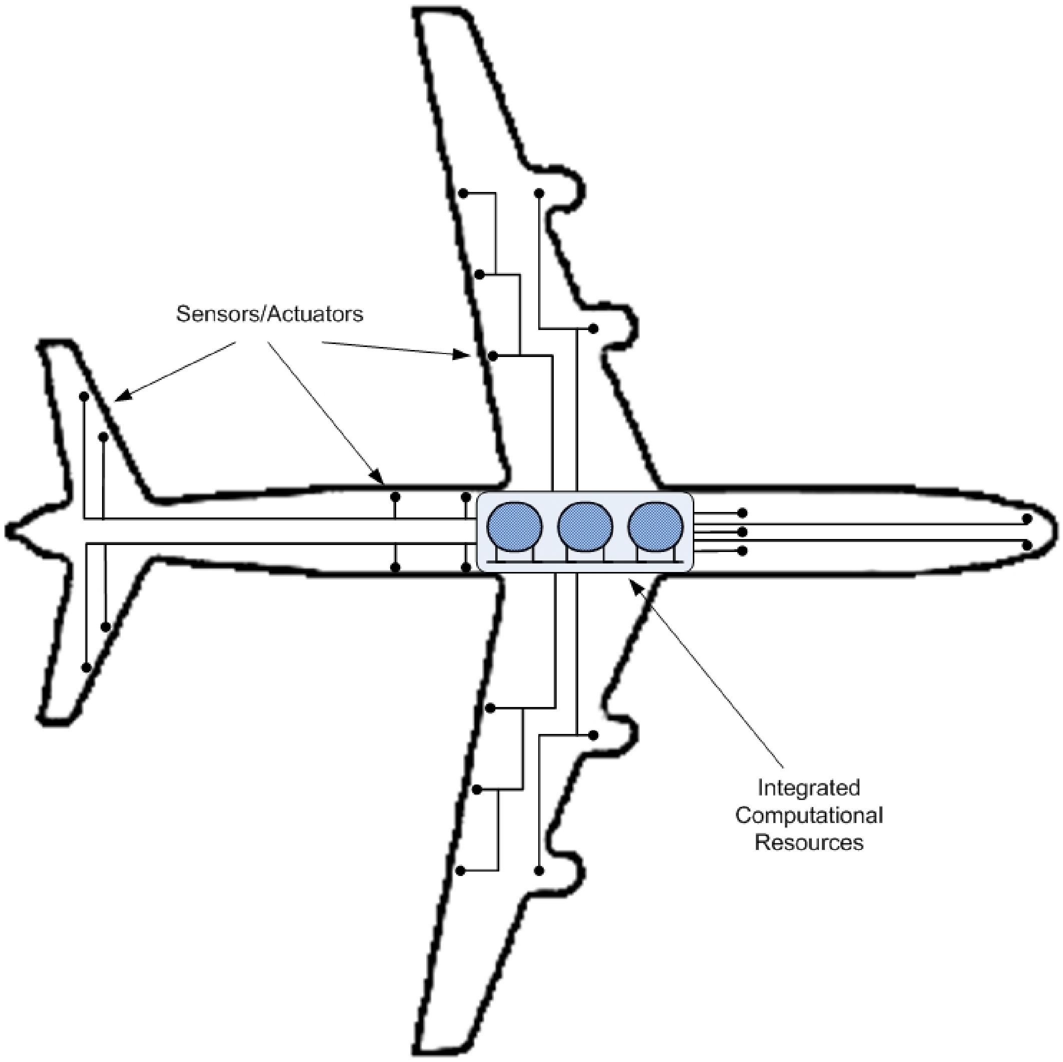

Figure 2.2: Schematic Illustration of IMA Architecture Design

The advantages of IMA are included in this definition:

- Flexibility: Flexibility allows to distribute functions over several computational resources. Therefore, it is possible to place the function in its naturally best position. This means that it may be next to its interfaces like sensors and actuators.

  Furthermore, based on the modular approach, a system may be composed of several modules which facilitates flexible products.

- Certification costs are a major part of the development costs of aircraft functions. IMA supports the reusability of modules and therefore certification evidence. Furthermore, the overall development effort is significantly reduced if modules are used more than once.

- Interoperability: The use of an architecture allows the sharing of computational resources. Furthermore, it handles the sharing of data between aircraft functions which do not need a direct connection to all of their system interfaces anymore. The benefit of this is the reduction of electronic systems per aircraft which cuts weight and volume of the systems and equipment. In addition, advanced diagnosis for all subsystems is possible.

Another important property is stated in the definition which is the capability of building a platform by integration of hardware and software. This platform has to provide a shared environment for multiple applications. Therefore, the platform needs to have protection mechanism like robust partitioning [101]. Furthermore, the platform has to consist of a fault-tolerant network to support safety-critical distributed functions.

Other aspects which have to be considered are the demands of the aerospace industry [85], where system integrator and application developer may be different parties. In this case, the intellectual property has to be protected without causing any problems for integration.

Even going a step further, economic factors for aerospace systems are dependability, maintenance and enhancement capabilities. These requirements ask for fast and cost-effective upgrade possibilities and ways to introduce new operational functionality which are also demands for the platform. Furthermore, continuing diagnosis is needed to support error detection and maintenance to prevent unscheduled downtime.

Based on the definition and the points discussed above, there are several requirements for an IMA platform which have different origins and provide a lot of challenges [93][41]. An approach called DIMA is currently under discussion in the aerospace community. It deals with aspects like safety-critical and secure communication, distributed integration, partitioning and physical distribution in addition to already described IMA properties like flexibility, reusability and interoperability and will be discussed in the upcoming section.

2.4 Distributed Integrated Modular Avionics

Distributed Integrated Modular Avionics (DIMA) [107] is an approach in the scope of IMA with an important amendment. It combines the advantages of the federated and the IMA concept [101] by physically distributing the integrated modules and connecting them with a fault-tolerant communication system.

A schematic illustration of a DIMA architecture design can be found in figure 2.3.

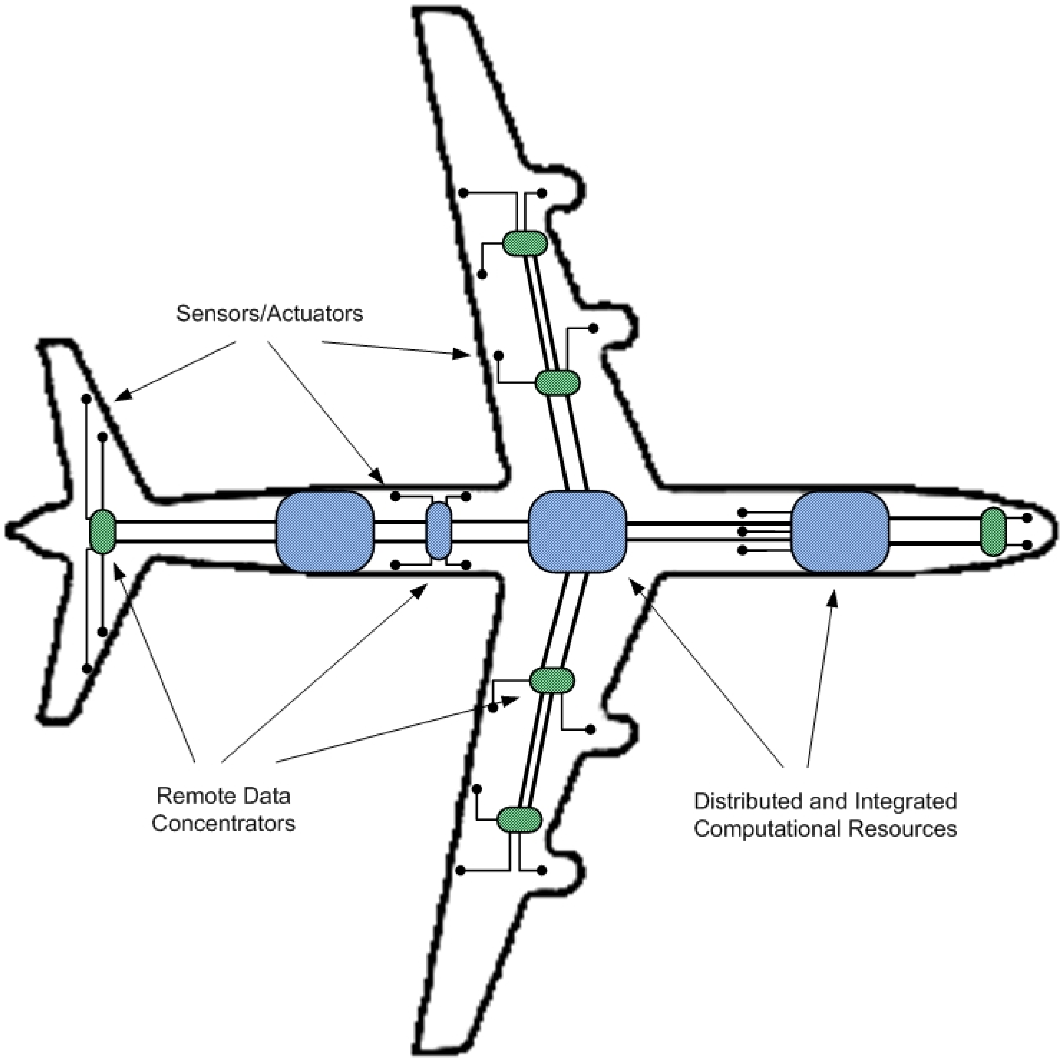
Sensors/Actuators
Remote Data Concentrators
Distributed and Integrated Computational Resources

Figure 2.3: Schematic Illustration of Distributed IMA Architecture Design

The main advantages of this concept are

- the natural fault propagation barrier and

- the physical positioning choices.

As already mentioned, federated avionics provide a natural fault-propagation barrier by the physical distribution of aircraft functions. DIMA uses this approach by distributing the functions over the aircraft but this does not solve the problem of fault-propagation completely. Considering the federated approach, the functions are not linked, in contrast to DIMA. Therefore, when using the DIMA concept, the fault-propagation barrier has to be provided at the interfaces between the modules of the function which means that the communication system is responsible for providing such a property.

The second main advantage is the possibility to position the functions near their in- and outputs [62]. Furthermore, remote data concentration units [108] may be used for exchange of input and output data with the next processing unit over a communication system. Because of the hierarchical structure of the architectural approach, cabling and system complexity may be reduced, because not every host in the system needs to be directly connected with each other. Furthermore, every processing unit in the system may be used for hosting functions because data may be shared with every other functions in the system. This reduces the overall needed processing power and therefore the number of needed hardware units [61].

Based on these advantages the DIMA architecture reduces weight, space and system complexity while demanding a fault-tolerant high-speed communication system to connect the modules. This communication system causes additional effort for the development of such a system.

2.5 Communication System

The communication system is an integral part of an avionics architecture. It connects the computational resources with each other and enables the sharing of data.

Based on this, there are several requirements for the communication system to be able to handle the communication in a correct manner. First of all, it has to be fault-tolerant. Therefore it needs to have a set of independent channels to be able to tolerate different types of faults.

This fault-tolerant approach also demands support for the replication of modules and data. The mechanism of replicated data handling should also be done on communication level instead of application level to increase the available processing power for the applications or to use cheaper processors for reducing costs.

Another requirement for the communication system is the distance between the modules. In IMA systems, it can only be a backplane bus, e.g. Safebus/Arinc 659 [58][15] which is used in the Boeing 777, which connects processing units within in a single box. In contrast, DIMA architectures need a communication system which is able to handle distances of up to 40 meters. Therefore, there are different physical requirements for such a communication system.

According to its relevance, a set of already existing or currently developed communication systems is discussed in detail in chapter 3 on page 19 and their suitability for the named requirements is compared.

2.6 From Federated to Integrated

This section discusses the way from a federated to an integrated concept [110] and describes its benefits. Kopetz, Obermaisser, Peti and Suri [51] identified five key obstacles that interlink economic and technical considerations regarding automotive architectures which are adaptable for avionics purposes:

- Electronic Hardware Costs: In federated systems, hardware costs increase with every additional function because of the dedicated hardware principal. Integrated systems allow adding functions without the need of further hardware.

- Diagnosis and Maintenance: Real-time diagnosis, which is able to detect faults in the system, handle them and maybe solve the problem, is a necessary approach for fault-tolerant systems. Especially if there is no safe state available like in the automotive domain.

  Furthermore maintenance is a major point in this domain because of ground time costs. Therefore a diagnosis system which detects faults and notifies the service responsible would enable reduced ground time and costs.

- Dependability: Fault-tolerant electronic systems have to fulfill avionics requirements. Because of the architectural approach, complex dependability requirements may be decomposed into several supportive functions which reduces system complexity.

- Development Costs: Using an architectural approach supports and even forces integration and reuse. This means that changes in a subsystem may not influence other subsystems in any unplanned manner.

  Another main issue is certification where reusability is a major factor. The development costs may be reduced significantly by using module based certification, where the certification evidence can be reused for previously developed and certified modules.

- Intellectual Property (IP) Protection: One of the advantages of the federated concept is intellectual property protection because every company develops their system by itself. Therefore, every competing concept has to provide mechanism to protect the knowledge of the developing company.

2.7 Federated vs. Integrated

This section lists the advantages of both concepts, gives a summary and a clear overview about the requirements of an optimal solution. Kopetz, Obermaisser, Peti and Suri [51] named the advantages which were used to develop the Decos [79] architecture concept.

2.7.1 Advantages of Federated Systems

- Fault containment
- Error containment
- Intellectual property protection
- Independent development
- Complexity control

2.7.2 Advantages of Integrated Systems

- Interoperability
- Hardware cost reduction
- Dependability improvements due to reductions of wiring and connectors

- Improved diagnostics
- Flexibility
- Fault-tolerance
- Quality of service

Based on these key aspects, an integrated approach consisting of distributed subsystems would provide the cornerstone for current and upcoming requirements.

Hammet [54] describes the future of an avionics system architecture as follows:

> *The ideal future system architecture would combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware benefits of an integrated system.*

Such a system which has not been used in commercial systems yet, combines the advantages of federated and integrated system design, which is, in my opinion, concluded in a DIMA platform concept.

2.8 Conclusion

In this chapter of the thesis, several requirements, advantages, disadvantages, features and examples of digital avionics architectures have been discussed.

The next step, which has not been taken by commercial systems yet, is to evolve the IMA concept and combine the advantages of federated and integrated system design into the DIMA approach.

The DIMA architecture will be used for avionics function development in the future. But, based on the life-cycle of currently developed aircraft systems, it is not clear when DIMA systems will replace them.

Aviate, Navigate,
Communicate.

(Unknown)

Chapter 3

Communication Systems

3.1 Introduction

In chapter 2 on page 7 several concepts for aircraft architectures were discussed. The concluded result is that an integrated but distributed solution would be able to use all advantages of common concepts.

In opposite to IMA systems where all functions are integrated into one rack and therefore close to each other, a DIMA system may have to span a communication network over wider distances. Therefore, it is not possible to connect the different parts of the system by a backplane or other short range buses but by a safety-critical communication system, which provides ranges between subsystems of several decades of meters.

The communication system is an integral part of a distributed architecture. It connects the distributed subsystems and provides several services which are needed by the applications. If such services move from the application level to the communication level, the application complexity and the needed processing power are decreased.

Several suitable communication systems are available which provide different attributes. This section discusses these communication systems which may be used for DIMA based systems. Some of these systems have already been used in the aerospace domain while others are still in development. At the end of the chapter, a comparison can be found which discusses the suitability of the described communication systems for different classes of application in DIMA based systems.

A comparison of bus systems for safety-critical systems was already done by Rushby [102]. It also describes some of the systems named below. This section wants to pursue this work in showing further developments of these systems but also wants to narrow on DIMA suitable communication systems.

3.2 Communication System Categories

3.2.1 Introduction

A distributed architecture needs at least two different categories of communication systems to be able to fulfill its requirements. Based on a hierarchical approach, a core communication system is needed which connects different subsystems. Furthermore, there has to be a subsystem communication system for inter-subsystem communication [115].

3.2.2 Core Communication System

The core communication system is the central part of a distributed architecture which connects different (sub)systems. It has to provide several attributes like fault-tolerance, high-speed and high-performance.

If a single architecture is used, these communication system can also connect systems from different suppliers and therefore be globally configurable. This means that its specification is the interface between several parties and therefore, it has to be configured by the OEM or system integrator.

3.2.3 Subsystem Communication System

The subsystem communication system is the central part of a distributed (sub)system. It connects several nodes of these subsystem to provide inter-subsystem communication. It is connected by a gateway to the core communication system and therefore also connected to every other system, subsystem or even node in the architecture. Its main attribute is low-cost, although it may have to provide fault-tolerance depending on the requirements of its subsystem.

3.3 TTP

3.3.1 Introduction

The Time-Triggered Protocol (TTP) [75] has been developed at the Technical University of Berlin [24] and the Vienna University of Technology [84] in the last 25 years. For commercial purposes, a company called TTTech [11] was founded which has continued the development of TTP.

The implementation of TTP which is discussed in this section is TTP [116], which is a safety-critical communication protocol which fulfills SAE (Society of Automotive Engineers) requirements. TTP has already been used in several domains like automotive, aerospace, railway and special vehicles.

3.3.2 Conceptual Attributes

TTP provides several services needed for safety-critical real-time communication. Two main concepts which were major design drivers in the development of TTP are:

- Fault-Tolerance: TTP is based on the fault-hypothesis that a single component may fail in an arbitrary failure mode. The likeliness of two concurrent independent component failures is so unlikely that it is considered as rare event. Furthermore TTP provides a Never-Give-Up (NGU) strategy to handle this rare events.

- Composability: TTP ensures that distributed subsystems are composable to a larger system. This leads to several advantages like the reduction of complexity, based on the separation of large systems into smaller ones, reduced interface mismatches, based on a detailed interface specification and the protection of intellectual property, according to separation at the interfaces.

Several services, which are described in detail in the upcoming sections, were introduced on the protocol level to support these concepts named above.

3.3.2.1 Communication Services

TTP uses a TDMA (Time-Division Multiple Access) scheme for communication. This means that every node in the cluster has a given amount of time, its sending slot, to transmit data. Based on this, every node in the cluster can have such a slot, one after the other. A set of such slots is called a round. After the last slot in the cluster is finished, the next TDMA round starts with the first slot again.

Furthermore, TTP is a broadcasting protocol which means that the data in the sending slot is transmitted to all other nodes.

The information about the sending slot and every other information, which is necessary for communication is stored in the Message Descriptor List (MEDL). This configuration information is defined before runtime which ensures a consistent view of the communication for every node in the cluster.

Cluster Startup Cluster startup is a service which is performed at power-on or a reset of the whole cluster. This service changes the state of the cluster from an unsynchronized to a synchronized one.

A node queries for synchronization information from other nodes and if received, it uses this information to synchronize with other nodes in the cluster. If no synchronization information is received after a certain period of time, a node performs a "cold start". This means that the node starts to send synchronization messages by itself to provide other nodes in the cluster with the necessary startup information.

Another concept which is used at startup is called "big bang". This mechanism ensures that in case of a startup collision between two cold starting node, no one will integrate on any of the collided frames.

Integration The process of synchronization to a running cluster is called integration. The controller state information, which is sent by other nodes in the cluster, is used to get the necessary synchronization information and based on this, the node tries to synchronize to the network and acquire a sending slot.

Data Transport Data transport is the service of distribution of data. This service only consists of application data and does not include protocol information overhead and data provided by other services like global time base.

The host communicates with the TTP controller via the CNI (Communication Network Interface). This is a dual ported memory where either the host or the TTP controller can read or store data. For data transport, the host stores the data into the CNI and these data will be sent in the nodes transmission slot.

Based on the broadcasting functionality, every node in the cluster receives the data and stores it in its CNI to provide it for the application running on the host. Furthermore, an acknowledgment algorithm is implemented which provides information about the validity of the transmissions in the cluster.

TTP supports replicated and non-replicated data transport. A controller is able to send different data on different channels which may be lost in case of a single fault. In case of safety-critical information, it has to be sent on both channels to prevent this scenario.

Clock Synchronization - Global Time Base The global time used in the TTP protocol is based on a sparse time base [69][42]. This means that the time is separated into statically designated intervals. This approach reduces flexibility but also reduces complexity and simplifies clock synchronization and the determination of concurrency.

To determine a global time over a cluster, distributed clock synchronization has to be done. To achieve this, an algorithm is used which calculates a correction term for the own local clock based on several local times from different nodes in the cluster to bring the clock into better agreement with the ensemble. The algorithm which is used by TTP is called fault-tolerant average (FTA) algorithm [73].

Noise Tolerance TTP is able to tolerate one (permanent) noisy channel during startup and normal operation. This is important to fulfill the single point of failure fault-hypothesis.

Acknowledgment Acknowledgment, which is a service based on the membership mechanism (see section 3.3.2.2), informs a sender whether the receivers of a message have consistently and timely got the new information.

Based on this, all nodes in the cluster have a consistent view about the transmissions and may decide about transmission errors or faulty nodes.

3.3.2.2 Safety Services

Safety for communication systems means, that the receiver gets all the data correctly and timely. Even in the case of errors, the receiver should get the data. To be able to handle this, consistent communication is needed which is enabled by a global view of the system for all nodes in the cluster.

Membership Membership is one of the main services of TTP and is formally verified [23]. This service informs all nodes of a cluster about the operational state of each node within a latency of about one TDMA round.

Every node maintains a membership vector, including information about all other nodes in the cluster, which is broadcasted with every data transmission. Due to this service, every node in the system receives this information and compares it with its own membership vector, which ensures a consistent view about the operational state of each node in the system.

Clique Detection To avoid clique formation and to detect inconsistencies in the cluster, the number of nodes that agree on the current controller state is monitored.

Host/Controller Life-sign The host life-sign is used to indicate the TTP controller that the host application is alive.

If the host life sign is not updated periodically, the TTP controller does not send any data and switches into a passive mode. It is also a pre-condition for correct controller startup.

The controller also provides a life-sign to the host to indicate a correct operational state.

Bus Guardian The bus guardian is an autonomous subsystem, which may be implemented locally (bus topology) or centrally (star topology) and protects the communication channels from temporal transmission failures. Additionally, it can be used to prevent "slightly-off-specification" faults.

The bus guardian has knowledge about the communication schedule and the time. Based on this information, it opens a transmission window for the sending node during the time of its sending slot. Nodes which are not allowed to send at the moment, are prevented from transmitting data and therefore from corrupting the current transmission.

3.3.2.3 Higher Level Services

These services provide extra functionality to the host application but are not necessary for basic TTP communication.

Cluster Modes Based on the fact that a system can be used under different circumstances, TTP is able to handle different modes. These modes can have different schedule parameters but the TDMA slot sequence has to be the same.

TTP needs at least one cluster mode, the startup mode, which is used for the startup of the cluster. Usually after startup, a mode change is proposed to change from startup mode to a normal operation mode.

External Clock Synchronization This service can be used to synchronize a cluster with an external time source.

A time gateway node which is connected with an external time source on the one hand and part of the synchronized cluster on the other, is able to calculate the external rate correction value, combine it with the clock synchronization value and broadcast it to the other nodes to synchronize them with the external time source.

3.3.3 Implementation Attributes

3.3.3.1 Network Topology

TTP provides two network topologies:

- Bus Topology: In the bus topology, every node in the system is connected to both communication channels. To protect the bus from faulty nodes, local bus guardians are used. To support the single-fault hypothesis, two TTP channels are needed.

- Star Topology: In the star topology, several nodes are connected by so called star-couplers. These star-couplers also provide a centralized bus guardian service and of course they are replicated to support the single-point of failure hypothesis.

TTP supports up to 64 nodes in a system and a mixture of both topologies is possible, too.

3.3.3.2 Physical Layer

TTP does not rely on a particular physical medium or bus coding scheme. But there are constraints which are:

- two independent physical channels,

- a shared broadcast medium and
- a known propagation delay.

Available physical layers support communication speeds of up to 25 Mbit/s for synchronous and 5 Mbit/s for asynchronous transmission.

According to physical distribution, cable lengths of up to 100 meters were successfully tested.

3.3.4 Application Attributes

3.3.4.1 Flexibility

Flexibility is a major disadvantage of TTP. This is based on the fact that the attention, in the trade-off between flexibility and safety, had always been on safety. Therefore, it is appropriate for safety-critical applications, especially demanded in the aerospace domain, but also narrows the system designer, especially in lower criticality cases.

3.3.4.2 Suitability

TTP is suitable for aerospace applications and has already been used several times in this domain. The TTA (see section 4.7.1 on page 69), which is the architecture behind this communication system, also provides constraints and services which are needed for designing a DIMA based platform.

A disadvantage of the communication system is the limited flexibility and the bandwidth which is whether high nor expandable enough to fulfill the requirements for a core communication system in a DIMA architecture. Considering subsystem communication, TTP is already used and may also be a possibility for the future.

3.3.5 Conclusion

The communication systems which were described in this section provide several services which are needed for safety-critical aircraft functions.

Based on this, TTP is suitable for the use in aircraft functions but also has some weaknesses which keep it from being the core communication system. On the other hand it provides a safety-proven solution for other fields of application, even in the aerospace domain.

3.4 FlexRay

3.4.1 Introduction

FlexRay [31][34] is a communication protocol designed for the automotive domain to meet the requirements of communication bandwidth, determinism, reliability and scalability that are hard or impossible to meet with existing technologies. It addresses high-speed, high-throughput applications as well as distributed control functions with high accuracy requirements.

The FlexRay communication schedule consists of two main parts:

- The Static Segment: Comparable to TTP communication.
- The Dynamic Segment: Comparable to ByteFlight [64] communication.

Based on these segments, FlexRay tries to support deterministic, without the loss of flexible communication for smaller applications.

3.4.2 Conceptual Attributes

3.4.2.1 Communication Services

Communication Format The time-triggered and therefore fully deterministic "static segment" of FlexRay communication supports protection against communication controller faults and the clock synchronization mechanism offers a fault-tolerant distributed time base built up from low-cost hardware components. Services based on this segment are ideally suited for control algorithms and functions with strict timing and response time requirements due to the highly deterministic behavior of communication in this segment, even in very complex systems.

The event-triggered "dynamic segment" of FlexRay supports periodic and a-periodic data transmission based on a collision-free minislotting arbitration scheme. The advantage of this segment is that the communication bandwidth, reserved for dynamic transmissions, does not need to be statically scheduled and is therefore well suited for highly dynamic services such as diagnosis.

Clock Synchronization Like every time-triggered communication protocol, FlexRay needs a global time base. Therefore FlexRay provides a fault-tolerant synchronization algorithm which ensures that all local clocks are synchronized.

Startup and Integration FlexRay provides startup and integration algorithms which allow a distributed startup of all nodes. After the startup, a node tries to establish the communication, which allows other nodes to join. If there is an interrupt during the startup procedure, the starting node stops trying to establish the communication and listens to the bus if other nodes are trying to establish communication too.

If a node resets itself, it can start up again and join communication. Using an integration algorithm, nodes who are not part of the communication anymore are able to join the communication again.

3.4.3 Implementation Attributes

3.4.3.1 Network Topology

FlexRay provides two network topologies, or a combination of both of them:

- Bus Topology: Every node in the system is connected to one or both communication channels using a bus topology. A local bus guardian ensures passive communication which means that a node only transmits when it is allowed to.

- Star Topology: Using the star topology, several nodes are connected by one or both communication channels to so called star-couplers. These star-couplers are central points which actively forward the received data to all communication nodes.

3.4.3.2 Physical Layer

Based on the physical layer specification [33][32], FlexRay supports communication speeds of 10MBit/s only.

3.4.4 Application Attributes

3.4.4.1 Flexibility

Based on the communication services of FlexRay, many existing functions, which are already implemented, can be reused and integrated into the communication network. Furthermore, new functions can be added to the same network instead of adding further communication systems to the already complex network architecture by using the static communication for regular and the dynamic communication for occasional data exchange.

3.4.4.2 Suitability

New functionalities which demand safety-critical communication are currently not addressed by the technology. This means that FlexRay is currently intended as a faster and more deterministic communication technology in the automotive domain but not as an enabler for distributed safety-critical applications.

At the design phase, such specific safety functions were intended to be implemented on higher levels. Using FlexRay for safety-related functions, which require more than basic FlexRay services, these specific safety services have to be provided by those higher layers as part of an additional hardware or software layer.

3.4.5 Conclusion

The advantages of FlexRay are that it is highly used in the automotive domain and therefore already integrated in several different microcontrollers as standard communication system. Furthermore, different vendors are available which provide development and production diversity which is an important issue in the aerospace domain.

On the other hand, the safety services which are also needed are not implemented innately and need additional resources. Therefore, to be able to widely use FlexRay in the aerospace domain, an approach is needed which supports the use of low-cost standard components but also introduces advanced safety services to FlexRay.

3.5 Layered-TTP and Layered-FlexRay

3.5.1 Introduction

Layered-TTP (L-TTP) and Layered-FlexRay (L-FlexRay) [20][80][124] provide conceptual extensions on top of the standard communication systems TTP and FlexRay. Therefore, the implementation attributes are the same like for the standard systems which were already discussed before.

These extensions which have been developed in the DECOS project (see section 4.7.3 on page 76) introduce several new services to the communication systems. Using this layered approach, both communication systems provide advanced services to fully support aerospace demands.

3.5.2 Conceptual Attributes

L-TTP and L-FlexRay provide a set of extended services for TTP respectively FlexRay. The extensions include advanced services for integrated systems. Based on this, TTP and FlexRay become more suitable for the use in DIMA architectures.

The name of L-TTP and L-FlexRay originates from the layered approach. It consists of two layers, the synchronization and communication layer (SCL) and the advanced services layer (ASL). Both of these layers and their services are described in the upcoming sections.

3.5.2.1 Synchronization/Communication Layer (SCL)

The SCL provides the basic communication services with some additional functionalities in comparison with TTP or FlexRay. Only these additional services are described below.

Fault-Tolerant Startup Using a central bus guardian, any single fault during startup is tolerated. This service guarantees cluster startup within a known time.

Never-give-up Integration Strategy If the synchronization is lost, the controllers try to integrate or to coldstart. This never-give-up strategy supports fault-tolerance.

Flexible Cluster Round Schedule Using this service, a controller may have more than one sending slot per round. In opposite to TTP where every node has at most one slot per round, a task which needs high bandwidth is able to provide more data.

Dynamic Slots This service allows a task to allocate an additional dynamic slot if needed. This slot is assigned during runtime according to the priority of the task. A major advantage in opposite to standard FlexRay is, that this slot is also protected by the bus guardian. This service and the flexible cluster round schedule increase the flexibility for the cluster designer.

3.5.2.2 Advanced Services Layer (ASL)

The ASL provides some higher-level services which may be used by the host application. These services, which are described below, are optional and may be deactivated.

ASL/SCL Independence An important constraint which is supported by this approach is the independence of the layers. This ensures that membership or clique failures do not have influence on synchronization and communication.

Partitioned Membership Partitioned membership allows to define different membership groups which do not influence each other. This supports the possibility of having applications with different safety-levels in the same cluster. This algorithm has also been formally verified [87].

Another service is called passive membership which allows to agree on the membership of a slot which does not belong to the same membership group. Furthermore, it is possible to have membership free slot and nodes.

3.5.3 Application Attributes

3.5.3.1 Flexibility

Increased flexibility is one of the major advantages of L-TTP in opposite to TTP and was a major design driver. Based on this lack in TTP and according to new requirements in the aerospace but also in the automotive domain, L-TTP is an approach to fulfill these requirements and increase flexibility without a major reduction of safety.

In terms of L-FlexRay, flexibility is provided by the underlying FlexRay communication system. The layered services on top of it provide advanced safety services without a significant reduction of flexibility.

3.5.3.2 Suitability

Like TTP and FlexRay, their layered upgrades are suitable as safety-critical subsystem communication systems. According to their extended services, they have advantages compared to the standard systems but the performance is still too slow for the use as core communication system.

An interesting possibility is that the additional layers can be included into the central switches of a star topology which provides the extended services to the communication system by simultaneously using cheap commercial-off-the-shelf (COTS) components at the end nodes.

3.5.4 Conclusion

The layered approach for TTP and FlexRay provides the advantage that both communication systems become more suitable for integrated architecture communication, according to their extended services. Furthermore, it reduces the disadvantages of both systems regarding the trade-off between safety and flexibility. On the one hand, TTP, which is designed to be highly reliable and safety-critical, gains more flexibility. On the other hand, FlexRay, which has no safety related services innately, conceives these safety-services on communication system level.

3.6 AFDX

3.6.1 Introduction

Avionics Full Duplex Switched Ethernet (AFDX) is described in the ARINC (Aeronautical Radio, Inc) specification 664 [19]. The ARINC664 specification deals with communication systems for aerospace applications.

During the 90s, Airbus performed several technology programs to evaluate new technologies in various domains. Regarding data bus communication, the goal was to find a better solution then ARINC429 [18] in terms of costs, performance, flexibility and applicability.

The first consideration was the ARINC629 [17] specification which is based on another avionics communication system. But later on, technology from the telecommunications domain was evaluated, especially Ethernet. Based on the maturity, the standardization aspect and the hardware costs, Ethernet was the preferred choice and Full Duplex Switched Ethernet was adopted to fulfill civil aerospace requirements [44].

Airbus specified the AFDX protocol which is used in the Airbus A380 the first time. Based on this communication service, IMA functionality is contributed using distributed communication resources and multi-function real-time computers.

3.6.2 Conceptual Attributes

3.6.2.1 Virtual Links

Packet routing mechanism in AFDX are called virtual links. In traditional Ethernet, the destination address is used by the switch to route incoming frames to the correct output links. In AFDX, a value called the virtual link ID is used to route the frames within the network.

The switches in an AFDX network are "configured" to route an incoming frame to one or more outgoing links. An important property of AFDX is, that a frame has exactly one originating end system. Based on the virtual link ID, the switches are configured to deliver the frames to a predetermined set of end systems.

The virtual links concept also provides partitioning at the network layer and a flow control mechanism, which regulates the flow of data between the switch and the end systems. Partitioning is done by the virtual links, which should not be shared by two or more source applications or source partitions.

3.6.2.2 Service Guarantee

AFDX provides a set of guaranteed services, like the bandwidth and the maximum end-to-end latency of a virtual link. But there is no guarantee that a packet is delivered. Acknowledgments for transmission and retransmission have to be handled at application level.

Bandwidth The communication system provides a bandwidth control mechanism. The Bandwidth Allocation Gap (BAG), which defines the minimum time interval between two successive frames assuming zero jitter, is used to transmit the given data. The system integrator has to define the BAG value and thus the allocated bandwidth for each virtual link, according to the application and equipment requirements. These configuration values are stored in the appropriate end systems and the configuration tables of the switches.

Jitter Jitter is introduced by the transmission of frames for a virtual link. It is defined as the interval between the start of the BAG interval and the first sent bit of the frame.

An end system might have to send multiple virtual links which can delay a frame up to the maximum allowed jitter value, to limit the instantaneous frame rate of the end system and thus accommodate frames from other virtual links.

Latency The maximum system latency is not defined by the specification but each supplier has to specify the upper limit for any delivered system. According to these limits, the maximum latency of the whole network can be evaluated.

3.6.2.3 Real-Time Control

Based on the guaranteed services, an accurate time-stamping logic and the latency control in form of the maximum network transit delay control, real-time performance is achieved for special end systems.

3.6.2.4 Redundancy Management

In an AFDX network, there are always two independent, physically separated paths between each end system. Furthermore, there are redundant switches to support fault-tolerance.

The default way is the transmission of the same frame on both networks. The receiving end system accepts the first valid frame and passes it to the application. For every received frame, an integrity check is done. If a frame is valid, any other frame with the same sequence number is discarded.

Based on the configurable redundancy option it has to be decided for every frame whether it is sent either via one or both separated paths.

3.6.2.5 Application Services

From the application point of view, AFDX provides three different services:

- Sampling: A simple connectionless implementation which does not support acknowledgment.
- Queuing: Like sampling, a simple connectionless implementation without acknowledgment.
- File Transfer: The Trivial File Transfer Protocol (TFTP) is used for file transfer.

3.6.3 Implementation Attributes

3.6.3.1 Network Topology

AFDX uses a star topology with a special switch as central connection point. The end systems are connected by a point-to-point connection to the switch which is responsible for the routing of the data frames.

A network consists of a maximum of 24 end systems but may be cascaded to construct larger networks.

3.6.3.2 Physical Layer

Based on the fact that AFDX is an Ethernet based communication system, the communication speeds are 10MBit/s and 100MBit/s. Furthermore, it supports cable lengths of up to 100 meters.

3.6.4 Application Attributes

3.6.4.1 Flexibility

Flexibility was a major design driver during the development of Ethernet. Therefore, AFDX provides a high level of flexibility, too. But according to the requirements in the aerospace domain, flexibility was decreased to increase determinism and predictability.

These limitations are reflected in the maximum number of end systems in one network and the virtual links including its predefined data handling. Despite these restrictions, AFDX is one of the most flexible solutions in the field of avionics.

3.6.4.2 Suitability

AFDX is already used in several major airplanes and supports IMA based systems. Due to its flexibility and high performance, it is absolutely suitable for the use in aircraft functions and as DIMA core communication system in special.

There is only one major disadvantage which is the absence of several system services on communication level. More precisely, most of the required safety services have to be implemented on application level which reduces the efficiency of the host computers and therefore the computational power available for the aircraft functions. Furthermore, the complexity of the application is increased.

A minor disadvantage is the absence of a tool chain for the configuration of the network and its end systems. But this problem will be addressed in near future.

3.6.5 Conclusion

AFDX is a relatively new but already used communication system for aerospace systems. It is based on Full Duplex Switched Ethernet including additional services like guaranteed latency, guaranteed bandwidth and virtual links to fulfill aerospace requirements.

The biggest advantages are its high performance, its mature functionality and the high degree of flexibility. The only disadvantage can be seen in the absence of advanced system services which requires their implementation on application level.

In the future, AFDX should be extended to provide a set of system and safety services on communication level to be able to fully support DIMA architectures.

3.7 TT-Ethernet

3.7.1 Introduction

TT-Ethernet [53][52] combines the advantages of common Ethernet and TTP. It allows the coexistence of TDMA and CSMA (Carrier Sense Multiple Access) schedules by using a star topology with central switches. These redundant switches do not only provide message forwarding but also include central bus guardians, which verify the communication and assure determinism and a priori known latency according to the differentiation of time-triggered and event-triggered messages.

3.7.2 Conceptual Attributes

3.7.2.1 Communication Services

The central switches, which ensure the communication between the communication nodes, are the main parts in TT-Ethernet. If one of the central switches gets a message, it verifies if this is a time-triggered (TDMA based) or an event-triggered (CSMA) one. The time-triggered message always has priority because this ensures a deterministic communication and an a priori known latency.

There is no possibility that two time-triggered messages arrive at the switch at the same time because these messages have to be properly scheduled before. If two event-triggered messages arrive at the same time, one will be delayed until the other one is transmitted.

3.7.3 Implementation Attributes

3.7.3.1 Network Topology

TT-Ethernet only supports a star topology because the centralized switch [65] is the major communication point. The central switch provides the safety services and handles the distinction between time-triggered and event-triggered messages.

3.7.3.2 Physical Layer

TT-Ethernet uses standard Ethernet physical layers which allow communication speeds of 100MBit/s, 1GBit/s and 10GBit/s. Based on the Ethernet approach, either electrical or optical physical layers are available. The decision which one to use depends on the requirements of the application and its environment.

3.7.4 Application Attributes

3.7.4.1 Flexibility

TT-Ethernet is compatible to Ethernet and AFDX innately and supports a central gateway approach for TTP and FlexRay. Therefore, it is able to communicate with several different communication systems and to connect subsystems with different classes of criticality.

3.7.4.2 Suitability

Based on the fact that TT-Ethernet is compatible to AFDX, the current standard for high performance communication systems, it is suitable as DIMA core communication system. TT-Ethernet provides high dependability, high performance, fault-tolerance and support safety-critical communication by extending AFDX with its missing services.

3.7.5 Conclusion

According to the fact, that TT-Ethernet combines Ethernet and TTP, all important communication and safety services of TTP are implemented in TT-Ethernet too. Consequently, it allows deterministic, fault-tolerant and safety-critical communication with advanced diagnosis features.

As a result, it is perfectly suitable as high-performance core communication system for DIMA architectures.

3.8 Spider - Robus

3.8.1 Introduction

The Scalable Processor-Independent Design for Extended Reliability (SPIDER) project is a research project of the NASA [10] formal methods group in Langley [81]. The goal of the project is the formal verification of safety-critical communication systems and architectures as well as creating an environment for the use of formal verification for certification purposes.

SPIDER is an architecture which provides partitioning for the application and a safety-critical communication system between these partitions, respectively applications. The communication system, which is used, is called Reliable Optical Bus (ROBUS). ROBUS, which is based on a fault-tolerant TDMA bus, and its services are fully verified by formal methods.

SPIDER, including its communication system ROBUS, is still a research project, which is prototyped in conjunction with Derivation Systems [111] but does not get designed commercially. Since this chapter discusses communication systems, the section below deals with ROBUS and its services. For detailed information about the SPIDER architecture, please refer to section 4.7.2 on page 74.

3.8.2 Conceptual Attributes

3.8.2.1 Distributed Coordination

ROBUS uses a set of two main protocols:

- Synchronization Protocols: The synchronization protocols are based on event-triggered communication which is used to synchronize all nodes in a system. This is done at startup or during a restart phase.

- Synchronous Protocols: The synchronous protocols use time-triggered communication, which is started after synchronization to provide the operational communication.

Clock synchronization is used to coordinate the local clocks of the nodes and to provide a global time. The fault-tolerance attributes allow clock synchronization even in the occurrence of faults.

Based on the synchronous state of the node and a determined execution scheme, ROBUS provides a highly deterministic behavior.

3.8.2.2 Redundancy Management

Fault Containment and Diagnostics The communication system provides redundancy management on communication level. Furthermore, it establishes fault-containment-regions (FCRs) which isolate faults and prevent their propagation.

ROBUS provides a distributed diagnostic system which is divided into two layers:

- The Local Layer: The nodes monitor the communication and diagnose the bus and each individual node separately.
- The Collective Layer: The nodes exchange their local diagnostic information to extend their local assessments.

Every node performs several diagnostic functions like error detection, node assessment and bus assessment.

Cliques Groups of BIUs and RMUs which are working together are called cliques in ROBUS. If the service of a clique is in accordance with the specification, it is considered trustworthy. Based on the diagnostic assessments, a clique membership is provided which indicates the trusted nodes.

Error Detection and Containment Based on the FCRs, the only path of error propagation between nodes is through their interfaces. Therefore, barriers are placed at both ends of the interfaces. If a local failure is detected or a bus failure is considered, the interfaces are disabled. Furthermore, input-error detection, in-line checks and dynamic voting are used to mask undetected errors from trusted sources.

3.8.2.3 Operational Modes

BIUs and RMUs use the same state transitions in the operational mode. After enabling the node or after a local or a bus failure, the node starts with the self-test. If this test is passed, the node searches for existing cliques which are already in the clique preservation mode.

If a working clique is found, the node tries to join it in the corresponding mode. If no clique is found, the node tries to establish a new one in the clique initialization mode.

If the node joined an existing clique or set a new clique up, the node switches to the clique preservation mode.

3.8.2.4 Point-to-Point Communication

The point-to-point communication supports three different communication modes:

- synchronous: It is used with the synchronous protocols which uses a time-triggered communication scheme based on the local time.
- fixed-delay: It is also used with the synchronous protocols. Based on event-triggered events, the data is buffered for a predetermined time till it is processed. This communication mode is used for synchronization.
- asynchronous-monitoring: It is used by a recovering node to observe the bus in order to be able to synchronize the local time source.

3.8.3 Implementation Attributes

3.8.3.1 Network Topology

The network topology used in the ROBUS communication system is an active star topology. Based on the concept, it consists of two major parts:

- Bus Interface Units (BIUs): These units are used for the network access.
- Redundancy Management Units (RMUs): These units are network hubs which provide the connection between the BIUs.

The BIUs and the RMUs are connected using point-to-point links.

3.8.3.2 Physical Layer

The design of the communication system is independent from the physical implementation of the point-to-point links, which are used as connection between the BIUs and RMUs. It is suitable for use with point-to-point optical data links.

3.8.4 Application Attributes

3.8.4.1 Flexibility

The focus in the design of ROBUS, in the trade-off between safety and flexibility, is clearly on safety. In particular, the fully verified services and the predetermined communication schedule state this fact.

To increase flexibility, it is possible to introduce a new schedule during runtime using an agreement protocol. If all nodes in the clique agree on this schedule, it will be used after the agreement process. If there is no agreement on the schedule, a default schedule is used.

Based on this, flexibility is increased a bit but is still very static compared to the Ethernet based protocols.

3.8.4.2 Suitability

The architectural approach based on a communication system which provides a set of safety and system services, ROBUS is theoretically perfectly suitable for aircraft systems and DIMA architectures.

But due to the fact that ROBUS is a research project which is not offered commercially it can not be used at the moment.

3.8.5 Conclusion

ROBUS is one of the most advanced safety-critical communication systems which has been designed yet and it provides important research in the area of formal verification.

But it is nevertheless a research project which does not allow the practical use in an aircraft. Based on this, it is a standard, for every other communication system which is developed, in terms of formal verification and theoretical investigation but cannot play a major role in current or closely upcoming aerospace platforms.

3.9 Comparison

From a theoretical point of view, the ROBUS communication system is the most advanced one, according to the fact that every service is formally verified. Furthermore, it is able to tolerate several possible failure scenarios, including Byzantine failures. The main problem of ROBUS is that it is commercially not available.

TTP, FlexRay, and their layered versions L-TTP and L-FlexRay, provide a set of services which support the architectural approach but their performance is too low to be suitable for a DIMA core communication system. But for subsystem communication, some of these communication systems are already used and all of them are perfectly suitable.

AFDX is already in use in aerospace applications and provides a high degree of performance but does not include services which therefore have to be implemented on application level.

Therefore, TT-Ethernet might be the best choice for a communication system in a DIMA based platform. It has good performance which is comparable to AFDX, supports legacy systems and provides additional safety and system services at communication level.

3.10 Conclusion

This section shows that there are several different communication systems available which can be used for aircraft applications or architectures. According to their different constraints and attributes, they can be used for different classes of communication systems needed by a DIMA architecture.

Regarding commercial aircraft, AFDX has already been used several times and is therefore considered to be the standard. Upcoming projects will clarify if TT-Ethernet will be able to compete. In terms of subsystem communication, where TTP has already been used several times, there is an interest to use FlexRay because its communication controller is already integrated into several microcontrollers.

Chapter 4

System Architecture

The major difference between a thing that might go wrong and a thing that cannot possibly go wrong is that when a thing that cannot possibly go wrong goes wrong it usually turns out to be impossible to get at or repair.

(Douglas Adams)

4.1 Introduction

The system architecture, including the communication system, defines the attributes, services and constraints of DIMA architectural based platforms. Good architecture design reduces the work of the aircraft function developer. Most of the network management and communication services are done hidden from the application. Therefore, complexity is reduced by increased safety.

The system architecture consists of the communication system, the system services, the operating system, the hardware, including its drivers, and a software tool chain for configuration.

This chapter discusses the requirements that have to be considered in the design phase of an architecture, talks about operating systems and their services, describes the structure of the hardware, defines demands for a tool chain and shows some examples from already used or still developed architectures.

4.2 Design

4.2.1 Introduction

During the design of an architecture, several constraints and requirements have to be considered. This section wants to discuss such key factors which are needed by an architecture which is suitable for the use in the aerospace environment, especially in DIMA systems.

Most of this is also valid for architectures in several other domains like the automotive's, special vehicles' or other safety-critical systems.

4.2.2 Composability

Composability [113] means that a system is composable into subsystems which provide the functionality of the whole system. An architecture which ensures composability must adhere to following four principles: independent development of nodes, stability of prior services, constructive integration of the nodes to generate the emerging services and replica determinism as discussed by Poledna [88].

Composability provides several advantages for platform design and implementation like:

- Reduction of Glue Code: If the system has well defined interfaces, which is a requirement for composability, glue code, which is needed to connect the subsystems at the interfaces, is minimized. Since the interfaces are defined in the value and the time domain and they are considered during (sub)system design, no glue code should be needed at all.
- Reduction of Complexity: If the system is decomposed into smaller pieces, the mental complexity of the subsystems is reduced [100]. This provides an easier understanding of the subsystems and therefore the whole system, which leads to higher quality and faster development.
- Advanced Testability: Decomposed subsystems can be tested independently which allows the start of the testing process in a much earlier state of development. Furthermore, this focused testing allows to find design and implementation faults much easier than by testing the whole system at once.
- Specialization and Focusing: Based on the separation into subsystems, every development team can focus on their core competence. Furthermore, the integration can be done by a specialist which forces productive work with increased quality and reduced instruction effort.
- Distribution of Work: Different teams can develop different subsystems in parallel without having a direct dependency between them. Even testing and integration can be distributed which allows efficient development.

- Protection of Intellectual Property: Based on the fact that outsourcing is a popular strategy nowadays, several subsystems can be outsourced without the risk of losing intellectual property. Furthermore, the system integrator is able to handle all system design decisions regardless the design or implementation considerations of the subsystem provider. The interfaces of the architecture are the only point of connection which needs to be defined clearly.

Based on this, the straightforwardness will be increased and this allows an efficient development of aircraft functions. Therefore, composability is a major design driver for an architecture in the aerospace domain.

4.2.3 Scalability

Scalability means the partitioning and abstraction of large systems into subsystems that are easier to handle. This is supported by the communication system which encapsulates functions and makes only properties available, which are needed for the correct operation of the function.

4.2.4 Extendability

Extendability describes how an architecture can be extended for future requirements. The development of an architecture is an evolving process, where new requirements can come up after the basic design has already been specified. But not only in the development phase, also during the use of a system, innovations in established technologies can force the extension of a system.

A scalable architecture is designed to support extensions and enhancements. A distributed architecture provides the possibility of extending the system by adding or changing nodes. This allows to add further functionalities or processing power to the system. Furthermore, gateway nodes can be used to connect several clusters to a system which provide additional resources and the possibility of hierarchical and therefore scalable architectures.

4.2.5 Complexity

A main question during the development of an aircraft function is how to manage the complexity. According to current highly integrated developments and by increasing size of the system, the complexity [37] and therefore the effort to understand the system increases too.

Using subsystems, the behavior of functions can be encapsulated behind simple interfaces to reduce complexity. Kopetz [70] describes this as follows: *"The partitioning of a system into subsystems, the encapsulation of the subsystem, the preservation of the abstractions in case of faults, and the most importantly, a strict control over the interaction patterns among the subsystems, are thus the key mechanisms for controlling the complexity of a large system."*

4.2.6 Dependability

Dependability is one of the most important claims in the aerospace domain. Based on the fact that no safe state is available during a flight, the safety requirements are important design drivers.

To provide at least a minimum of service during the occurrence of faults, a system has to provide following requirements to be dependable:

- Fault-Containment: Fault-containment [74][66] tries to limit the impact of a fault to a defined region, the fault-containment region (FCR). In a distributed architecture, one node can be considered to be one FCR.
- Error-Containment: Error-containment [74][66] tries to assure that an error, the consequence of a fault, may not propagate to other components and corrupt their state. In a safety-critical computer system an error-containment region (ECR) requires at least two fault-containment regions.
- Fault-Tolerance: Fault-tolerance assures that the system works even if faults affect parts of the system. Further information can be found in section 4.3.2 on page 52.

4.2.7 Partitioning

The term partitioning [101] is tightly coupled with the terms IMA/DIMA and Modular Certification because it is the baseline which is absolutely necessary for the development and certification of such systems. Partitioning is used to provide fault-containment. A fault in one partition must not influence any other partition neither in the spatial nor in the time domain.

There are two different types of partitioning which can be achieved in an architecture:

- On a Single Processor
- Across a Distributed System

Based on the fact that partitioning is used as the means of protection for the computer resources in an IMA system, the certification of the partition and their interfaces has to be considered during development [8]. The certification of such systems is discussed in detail in chapter 5 on page 83.

Using partitioning is no decision between the single-processor or the distributed system approaches, rather both approaches should be used to provide a consistent environment.

4.2.7.1 Partitioning on a Single Processor

Partitioning on a single node deals with the possibility of running several partitions on one host processor by sharing computational resources, interfaces and peripherals.

Advantages Aircraft functions with different levels of criticality can be hosted by the same processor. This allows sharing computational power and therefore reduces the number of needed hardware units in an aircraft, which furthermore decreases cost, weight, space and power consumption.

Implementation Partitioning has to be implemented in connection with the operating system. There are two ways where the functionality can be located:

- Above the Operating System (see part A of figure 4.1 on the next page): It provides the same structure as standard operating systems but has the disadvantage that the operating system has to handle a lot of functionality.
- Above a minimal Kernel, below the operating system (see part B of figure 4.1 on the following page): This approach can be considered as a "virtual machine" and has the advantage that partitioning depends only on the kernel and the hardware.

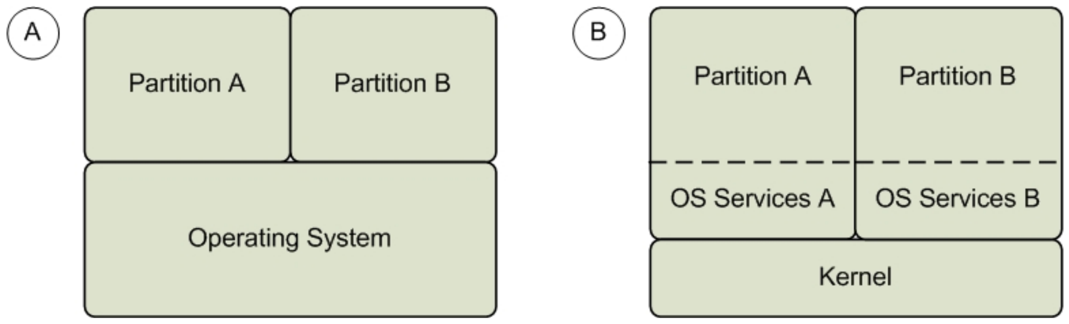

Figure 4.1: Implementation Possibilities for Partitioning

Additional information about operating systems and some examples can be found in section 4.5 on page 58.

Design Considerations Based on the fact that several partitions use the same processor, there are two major challenges for the development of this service:

- Spatial Partitioning: The memory of a partition can not be overwritten by any other partition, even if the other one is faulty. Furthermore, it is important that a partition does not have single access to the interfaces and the peripherals of the host system.

  Current processors provide memory management units which are able to handle different partitions in the memory and guard them against unauthorized changes.

  Another main point to consider is the communication between partitions because it can influence the state of the receiving partition. Therefore, only predefined communication should be possible, using buffers which are handled by the kernel.

- Temporal Partitioning: Based on the safety requirements of aircraft functions, which are mostly handled by real-time functions, the state of the system also depends on the time. If a partition has uninterruptible access to the processor, a single fault can corrupt the whole host. Therefore, deterministic schedules should be used to guarantee correct functionality, even if one partition is corrupted.

  Furthermore, such predefined static schedules, although they decrease flexibility, simplify the operating system and increase the safety because the state of every partition is deterministic.

4.2.7.2 Partitioning Across a Distributed System

Partitioning across a distributed system is similar to partitioning on a single processor, using the assumption that the whole system including the communication is comparable to a single processor.

Of course, there is no processing power which needs to be shared and memory can not be overwritten directly, but regarding the complexity, replication, use of different criticality levels, fault propagation and the communication between partitions the same problems but also advantages appear.

Advantages Large systems can be decomposed in several smaller components with different levels of criticality and therefore different requirements for certification. Furthermore, different functions are able to share nodes and in the distributed approach their replicated partitions are distributed over the system.

Implementation In a distributed system, partitioning has to be implemented in the communication system. The interface between the host processor and the communication system has to be the fault-propagation barrier which assures that only correct data may pass.

Design Considerations Like partitioning on a single processor, the separation between spatial and temporal partitioning has to be done in a distributed system, too:

- Spatial Partitioning: Spatial partitioning over a distributed system is easier than on a single processor because the partitions are already physically separated like in the federated avionics concept.

  Therefore, the most important point is the communication between the partitions. Like in the single processor case, predefined communication assures that only conscious data may influence the state of the partition.

- Temporal Partitioning: Temporal partitioning is also comparable to the single processor case. A static schedule of the communication system, based on a global time, provides determinism and therefore a high degree of predictability.

4.2.8 Layering

Layering is the possibility to abstract and reduce functionality to an adequate amount for the hosted application. This is very important for the design of an architecture to reduce complexity. Furthermore, using layers makes it possible to split the work according to the interfaces between these layers.

An architecture can be layered into the communication layer, the hardware, the operating system and the application. Each of these layers consists of several layers itself. The most important thing in a layered concept is that the interfaces between these layers are simple and well defined to prevent wasteful effort during the composition of the layers.

4.2.9 Time

In the context of real-time systems, time plays a major role. The validity of a value depends not only on its value but also on the current time.

4.2.9.1 Global Time

Several services which support fault-tolerance need to have a consistent view of the system and therefore a global time base.

Current communication systems provide a global time. This global time is established during synchronization of the nodes and is synchronized continuously to prevent drifts of the local clocks.

Based on this global time, the host is able to know the order of events, check deadlines, handle time-triggered communication and even schedule its tasks.

4.2.9.2 Correctness in the Time Domain

As already mentioned, preventing fault-propagation needs correct data in the value and in the time domain. Real-time systems need the correct data at the correct point in time in order to be able to react in time.

4.2.10 Conclusion

This section described a set of constraints and requirements which have to be considered during the design of an architecture, especially a safety-critical, distributed and integrated one.

Most of these points consider directly or at least touch the design and definition of different interfaces. Interface design is the most important but also most critical part during the design of such an architecture. If it is done well, it can solve a lot of problems at the start, if not, it can cause a lot of problems in the future.

4.3 Implementation

4.3.1 Introduction

This section considers the implementation details of the design requirements discussed in the previous section. Several mechanisms which are used to provide the claimed requirements are discussed.

4.3.2 Fault-Tolerance

Fault-tolerance is needed for safety-critical systems to assure that faults are not able to have catastrophic effects. Therefore, every safety-critical system should define how it will handle faults and how many faults will be tolerated.

4.3.2.1 Fault Hypothesis

The fault hypothesis [71] specifies how many and what types of faults are tolerated by the system. Therefore, it is the baseline for the development of fault-tolerance mechanism which have to be validated at the end of the project.

4.3.2.2 Never-Give-Up Strategy

If a failure arises which lies beyond the fault-hypothesis, there has to be a never-give-up (NGU) strategy in addition which should keep the system in a defined state. This can be the restart of a permanent faulty node or even the whole system, if necessary. If possible it should try to lead the system into a safe state.

4.3.2.3 Error Detection

Error detection provides two major functionalities. Based on error detection, strategies can be initiated which try to resolve the problem and bring it back to a correct state. In the worst case this has to be the NGU strategy.

Furthermore it provides important information for diagnosis which helps to debug and maintain the system.

4.3.2.4 Fault-Containment and Error-Containment

As already mentioned, fault-containment and error-containment tries to prevent the propagation of faults. To ensure this, the direction of control has to be unidirectional to reduce the dependency between different nodes and the interfaces have to obtain a strict specification.

Furthermore, fault-containment regions (FCR) and error-containement region (FCR) have to be considered in the design phase, to ensure functional and physical independence between FCRs which form an ECR. A single ECR has to consist of at least two FCRs.

4.3.2.5 Fail-Silence

Fail-silence means that a node either produces correct results in the value and the time domain or does not produce any results at all. If a node is fail-silent, it supports fault-tolerance even if it fails because it does not propagate any faults. Therefore, a set of critical failure types, which increase the complexity of the fault-hypothesis, do not need to be considered.

4.3.2.6 Critical Failures

This section describes two critical failure types which need to be considered in the fault hypothesis.

Babbling Idiot A major problem regarding the communication are nodes which are faulty and send data when they are not allowed to. This would corrupt any other communication and therefore affect the whole system. Based on a fault-hypothesis which depends on a single point of failure, this behavior has to be prevented. Current time-triggered communication systems use guards which allow a node to send in his specific time slot only. Therefore, it may corrupt its own slot but not the communication of any other node.

Byzantine Byzantine faults are derived from the Byzantine Generals problem [76]. It is based on the problem that a faulty node agrees every information it receives, even if different nodes send different information. This is difficult to resolve because it needs at least $3k+1$ different nodes which are connected to each other, whereas k means the number of Byzantine faults which have to be tolerated.

4.3.3 Redundancy

Redundancy is one of the widest used mechanism for fault-tolerance in the aerospace domain. In a federated architecture, every safety-critical aircraft function has at least one replicated node which is able to take over the provided functionality in case of a fault in the primary system. Highly critical functions, like the primary flight control, typically use quad-redundant hardware or even a higher degree of redundancy. Redundancy in IMA systems provides the advantage that a function is not bound to a specific hardware. Therefore, the plane can use the normal mode of operation, even in the case of faults, as long as the number of non-faulty processors is sufficient to provide the defined level of replication and therefore safety.

The approach that one system replaces the other is called redundancy. The possibility that several instances of the same function can run redundantly in the network needs consistent communication for all nodes in the system. If it is guaranteed that all instances of the redundant function implementations always run synchronously and always receive consistent input data, they will always produce the same output as well and thus one function can replace the other one in case of a fault.

Current communication systems are designed to support active redundancy. Replica determinism [88] guarantees that if one host receives a certain data item at a certain time, all others will receive either the same value or will be excluded from the list of valid hosts.

4.3.4 Diagnosis

Diagnosis has to be integrated directly into an architecture to allow the effective detection, identification and classification of experienced errors. This allows every component in the system to have a consistent and holistic view on the state of the distributed system.

Based on this advanced diagnostic information, debugging and maintenance are simplified and fault-tolerance is supported because recovery actions can be taken.

4.3.5 Certification

Every system that is used in the aerospace domain has to be certified before it is used in normal service. Therefore, certification has to be considered at the design phase as well [30].

Chapter 5 on page 83 discusses the certification for aerospace systems in detail and focuses on modular certification which is a rather new approach, appropriate for integrated architectures.

4.3.6 Conclusion

This section described a set of key points which have to be considered during the implementation of an architecture. All of these points regard safety which has to be the ultimate goal for aerospace or any other type of safety-critical systems.

4.4 Hardware Considerations

4.4.1 Introduction

The considerations regarding hardware in an integrated approach are different from those in the federated one [7]. In the federated approach, every hardware is developed for its own, dedicated purpose. Therefore, every node can be designed for its special functionality.

In an integrated approach, the hardware should be as general as possible to support all needed functionalities. Furthermore, a single processor should host several different applications or subsystems and provide access to its peripherals to everyone. Therefore, the organization of such an integrated node is much more difficult than in the federated approach.

4.4.2 Types of Hardware

An interesting point which arises when comparing different available operating systems, which may be found in section 4.5.4 on page 62, is the wide range of supported hardware. Some operating systems support hardware for special functionalities which equals dedicated hardware systems in the federated approach, others support general purpose hardware, like x86-compatibles, per default.

Within this wide range of different computer systems, this thesis wants to concentrate on multiple purpose embedded systems because they are widely used for safety-critical aircraft functions.

4.4.3 Node Design

From the viewpoint of the system designer, the hardware of a node consists of following major parts:

- Host Processor: The host processor is the central part which handles the access to the interfaces and the communication, and the execution of the software.

- Communication Controller: The communication controller handles the interaction with other nodes in the system. It can be included into the host processor or located on a separate chip. The advantage of the extern solution is that the processor and the communication controller may fail independently of each other. On the other hand, an integrated controller saves costs and space.

  If the node provides gateway functionalities, it is possible that it contains several communication controllers to shift information from one network to another.

- Environmental Interfaces: The environmental interfaces are the inputs and outputs of a node. This can be sensors, actuators, displays or any other interface to the environment. These interfaces can be the major property of the node, if it provides a transition to the environment, but can also be nonexistent, if it is a general purpose processing unit.

4.4.4 Interfaces and Peripherals

The interfaces and peripherals of a common aircraft function are sensors, actuators, displays and switches. A lot of them pertain the interface to the users which are the pilots or the maintenance staff.

New approaches for small aircraft try to standardize this interface using touch-screens and bigger displays to provide all relevant data. Even in commercial airplanes a standard environment is used to reduce the training costs.

Based on this experience, the systems in the aircraft will consist of processing units which can be located anywhere in the plane and the input and output nodes. These nodes are part of the architecture connected by a communication system. The information of these inputs and outputs is shared for all aircraft functions. The access is handled by the architecture, in detail the communication system and the operating system of the nodes.

Another advantage of this approach is the health monitoring functionality which is done on architecture level by logging all relevant data and providing it to the corresponding maintenance staff for detailed verification.

4.4.5 Commercial-Off-The-Shelf

The use of Commercial-Off-The-Shelf (COTS) hardware [67][46] allows to reduce the development of aircraft functions to software development and certification. The development of hardware is only necessary if a special functionality is needed. Based on this, the development costs can be reduced. Furthermore, using COTS hardware increases the safety because the hardware has been used and tested several times and faults in design or implementation can be found much earlier.

4.4.6 Integration of Communication

The development of special hardware which is not COTS has to be considered. Using a communication chip which is easy to integrate in the developed hardware provides the interface to the architecture. This chip should also support the upgrade from other communication systems to support the reuse of legacy systems.

4.4.7 Conclusion

Considerations regarding the choice of the hardware should include the aspects named in this section. Based on tight boundaries for costs and weight, the use of integrated solutions will increase and dedicated hardware will only be developed for special purposes.

4.5 Operating System

4.5.1 Introduction

This section discusses the requirements of an operating system which is suitable for the use in an integrated architecture [6]. In addition to already mentioned aspects in association with partitioning, there are several other properties which need to be respected.

An important factor is, for example, is the compliance to the Arinc653 standard and the possibility of certification for the operating system. At the end, this section shows several examples of operating systems, which may be suitable, and compares their attributes.

4.5.2 Design Considerations

4.5.2.1 Layering and Partitioning

Both points were already mentioned in the context of design considerations for architectures. Layering and partitioning [120] in terms of the operating system are related to the one already discussed but provide additional insight into the topic.

The operating system can be divided into several layers [86]:

- Communication Layer: The communication layer which provides the support for the respective communication system.
- Core Services: The main part of the operating system are the core services, which consist of the partition handling including their scheduling, the inter partition communication and the error handling system.
- Partition System Interface: The partition system interface provides the interface between a partition and the core services. It is responsible for the resource handling of the partition and the communication to the core services.
- Partition Level OS: The partition level OS provides operating system services to the partition like task scheduling and intra partition resource management.
- Application Layer: The application layer is on top of the partitioned OS. Based on the abstraction of the services of several layers, the implementation

details are hidden for the application and it can use the resources like in the federated approach where only a single function is hosted by one processor.

An important feature which supports these hidden implementation details is the virtual network service. This service provides different network interfaces to the application by hiding their implementation. For example, time-triggered and event-triggered communication services are supported [92][82][91]. Furthermore, it handles inter-partition and cluster communication which means that the application does not have to care about the difference between internal or external communication.

Based on the core services including the virtual network layer, it is possible to have different partitions with different criticality levels and different communication requirements on the same host processor.

4.5.2.2 Scheduling

Scheduling for safety-critical real-time applications is always a trade-off between safety and flexibility. There are two different ways to realize it:

- Dynamic Scheduling
- Static Scheduling

Dynamic Scheduling Dynamic scheduling provides more flexibility to the application developer but also needs more resources on the other hand. A dynamic scheduler is able to handle tasks in a changing environment and does not need to know every constraint in advance. But this is also a problem because it is not predictable and therefore the behavior under critical conditions is not defined.

Static Scheduling Static scheduling provides more safety to the system because it is absolutely predictable and therefore even critical conditions are known in advance and solved before runtime. The major disadvantage is that it is very inflexible and minor changes in the design need a complete recalculation and reconfiguration of the system.

4.5.3 Arinc 653

The Arinc 653 specification [16] describes the interfaces between an operating system and an application for the use in IMA systems. The so called APEX

(Application/Executive) is the main interface which is described. Furthermore, operating system services like communication and scheduling are described.

This specification builds the baseline for operating systems which are supposed to be used in an IMA environment. It consists of the core operating system, the APEX and several partitions.

4.5.3.1 Core OS

The core OS is the central part of the operating system which is called core services in section 4.5.2.1 on page 58. The already discussed idea of partitioning is a main concept in the Arinc 653 specification, too. Consequently, partitioning has to be an integral part of the core OS. Partitioning has to be robust in time and space to be able to host several different partitions. The scheduling of these partitions, which is handled by the core OS, has to be defined statically by the system integrator.

4.5.3.2 APEX

APEX is the interface layer between the core OS and the partitions. It allows independent applications to run on the same processor and use the same peripherals. APEX is the layer which is called the "Partition System Interface" in section 4.5.2.1 on page 58.

APEX provides the following benefits:

- Portability: Based on the fact that APEX is a static interface, it offers the same services and constraints for every processing environment. Therefore, every application which is based on this interface can easily be ported to other targets.

- Reusability: According to portability, the reuse of applications in other environments is also supported for applications based on the APEX interface.

- Modularity: Based on the decomposition of the application into partitions and the reduced hardware and software dependencies according to the APEX interface, modularized applications are supported.

- Integration of Application with Different Criticality Levels: Using robust partitioning, APEX supports applications of different criticality levels to run on a single processor.

4.5.3.3 Partitions

The partition level OS is responsible for managing the processes in a partition and the communication with the APEX interface. Each partition supports several processes which can be scheduled periodically or aperiodically. This means that dynamic scheduling as well as static scheduling is supported at partition level. For dynamic scheduling, task priorities are intended.

4.5.3.4 Time Management

IMA systems are often real-time systems and therefore a time management is needed. Time in the Arinc 653 specification is unique and independent of the partition execution in a node. Several services are defined, based on the unique time like scheduling, deadline monitoring, periodicity and communication.

4.5.3.5 Inter Partition Communication

APEX supports communication between partitions in a cluster, even if those partitions are not on the same node. The design of the communication has to be independent from the physical destination of the sender and the receiver. The communication uses messages for data transport which support time stamping for determination. Several times of different messages are supported like:

- Fixed or Variable Length
- Periodic or Aperiodic
- Direct or Broadcast Messages
- Acknowledged or Unacknowledged

4.5.3.6 Health Monitoring

The health monitor is a set of maintenance functions which reports hardware, application and OS software faults and failures. It helps to detect and isolate faults, and prevents their propagation.

Furthermore, it supports maintenance, by creating an error log, and supports recovery, by stopping and/or restarting single processes or partitions to resolve the failure or to obtain a safe state.

4.5.3.7 Configuration

Based on the goal of the specification to define an environment which supports full portability and reusability of the application, the configuration of the environment is an important decision. Therefore, the system integrator is responsible for the integration of the partitions into the nodes and their access to the required external data.

According to that the integrator needs a set of requirements for every partition:

- Memory Requirements
- Period
- Duration
- Incoming Messages
- Outgoing Messages

Based on this data and other information regarding the operating system, the configuration tables are created which configure the operating system and its services.

4.5.4 Examples

4.5.4.1 VxWorks 653 Platform

The VxWorks 653 Platform [112] is an operating system by Wind River Systems for aerospace and defense applications. The operating system provides full API conformance to Arinc 653 for the supported programming languages C and C++. Ada support is also possible but not directly included.

The partitions support software which is written for VxWorks, which is the real-time operating system of Wind River, or legacy operating systems which need to be adapted. Furthermore, a configuration tool suite and a certification package according to DO-178B Level A is available.

Following target processors and COTS boards are supported:

- Target Processors:
  - IBM 750GX
  - PowerPC 7xx, 74xx

- COTS Boards:
  - MVME5110
  - wrSbc7447, wrSbc7457
  - wrSbc750GX

4.5.4.2 Integrity

Integrity of Green Hills is a partitioned operating system for safety-critical systems. It supports the Arinc 653 APEX API and provides the possibility to run Ada, C and embedded C++ partitions on the same node.

A DO-178B Level A compliant certification package is available and it has already been used in several projects.

Several hardware targets are supported by performing the certification effort on the customer hardware.

4.5.4.3 LynxOS

LynxOS [78] is an Arinc 653 compliant operating system for safety-critical real-time systems. It supports the APEX API and the POSIX standard for operating systems.

A certification package according to DO-178B Level A is available as well as a development tool suite. Furthermore, training support is provided.

Following systems are supported by board support packages:

- AMCC 440EP
- Apple PowerPC G5
- Extreme Engineering XPedite6032
- Thales VMPC6a, VMPC6c, VMPC6d
- PC-AT/x86-compatibles

4.5.4.4 CsLEOS

BAE Systems' operating system is called CsLEOS [21]. There is already the second generation available which provides an Arinc 653 interface and support for OpenGL graphics for display control.

Furthermore, a development tool suite and a Level A certification package are obtainable.

Following board support packages are available for hardware support:

- RAD750
- BAE Systems Modular Control 555, Modular Control 750/7400
- Dy 4 Systems SCP/DCP-119, SVME/DMV-179, SVME/DMV-181
- SBS Technologies RL4, VG4
- Motorola Sandpoint PPCEVAL-SP3-755

4.5.4.5 Decos - EEE

The Encapsulated Execution Environment (EEE) [40][39] is developed for the Decos [79] research project. For inter-partition communication, event-triggered as well as time-triggered messages are supported which ensures the reuse of legacy systems.

Furthermore, a strong isolation between safety-critical and non safety-critical subsystems provides guaranteed safety (for safety-critical subsystems) by increased flexibility (for non safety-critical subsystems).

4.5.5 Conclusion

This section shows that an operating system which is suitable for an integrated architecture has to fulfill a set of requirements. Especially partitioning requires extensive attention in the design phase.

The Arinc 653 specification provides the baseline for this kind of operating systems [122]. Based on this specification there are several operating systems already available which fulfill most of the needed functionality. For academic purposes, the Decos project also develops an partitioned operating system which provides interesting approaches.

The choice of the operating system has to be taken carefully because it has effects on the whole platform. The main criteria include the availability of the certification package, the supported hardware platforms and the reusability possibilities of already developed systems.

4.6 Development Environment

4.6.1 Introduction

The development environment is a set of software tools for the configuration and initialization of the architecture. The most important tools are needed to create the schedule configuration of the communication system and the task respectively partition schedule of the operating system, if a static schedule is used.

Other tools which simplify the work with the architecture are download, diagnosis and debugging solutions.

4.6.2 Design Approach

The design of the used software configuration is based on the system which is chosen. As already mentioned, event-triggered communication and dynamic task or partition scheduling demands lower configuration data but a higher degree of processing power. Furthermore, they are less predictable which is difficult in terms of safety-critical systems.

The more complexity in the configuration, the less complexity has to be in the running system and therefore more safety and a better performance is achieved. If everything is predefined and predictable, critical situations can be simulated in advance but this also means that the system is not very flexible anymore. This is again a trade-off between safety and flexibility which forces the choice of the design.

As already mentioned, the schedules of the different parts of the architecture are the most critical parts in the system. Dynamic scheduling allows to reconfigure the system during execution without the need of a complete recalculation. Probably a mixture of dynamic and static parts is the best solution.

The configuration tools for scheduling, no matter if they are dynamic or static, often use different approaches to create and define the needed messages, tasks or partitions. Databases are often used to define this items but they are different to handle for embedded applications.

Therefore, simpler representations are often chosen for the used configuration, like structures which have the advantage that they can be accessed directly in the source code but have the disadvantage that they need a complete recompile if changed. Another possibility are tables which can also be accessed easily and do not depend directly on the source code.

Obviously even the design of the configuration tools consume a lot of mental work. Therefore, this thesis wants to discuss the possibilities for several tools which might be useful.

4.6.3 Communication

The type of the communication system for safety-critical systems is a highly discussed topic. There are several approaches which result in the well known trade-off between predictability and therefore safety on the one, and flexibility on the other side.

For example TTP is a fully predictable system in opposite to AFDX which is completely event-triggered. The solution of this problem or just the best approach might be hybrid systems like FlexRay which introduce flexible messages to a static schedule or TT-Ethernet which adds time-triggered real-time messages to an Ethernet based and therefore event-triggered system.

4.6.3.1 Schedule

Based on the type of system, either event-triggered or time-triggered, the communication system has a dynamic or static schedule. In case of a dynamic schedule, the scheduling process is done during runtime. Therefore, only the messages and their sizes have to be defined as configuration data and given to the application. The effort for a configuration tool is comparatively small.

In case of a static schedule, the scheduling process is done in advance. The scheduling tool needs a set of data regarding the messages, their deadlines and even the tasks which send or receive these messages. Furthermore, it provides the possibility to optimize the schedule and to simulate it. The effort for such a tool is even higher because the complexity of the scheduling is handled in this tool.

The decision of the type of communication system is a design question which is based on the application and its requirements.

4.6.3.2 Driver

The application needs an interface to handle the communication. Therefore, the communication system should provide an Advanced Programming Interface (API) for the control of the communication system. This API should be generic to support the use in several applications, no matter if it is an operating system or not.

4.6.4 Operating System

The operating system and its role in safety-critical systems has already been discussed. The scheduling problem is comparable to the one concerning the communication systems with the difference that a hybrid solution is more difficult to implement.

4.6.4.1 Task Schedule

Considering a partitioned operating system, the task schedule is handled for the partitions. Based on the fact that the partitions are absolutely independent it would be possible to implement dynamic schedules for non safety-critical partitions and static schedules for the safety-critical ones.

The disadvantage is that the system has to be predefined but also needs a scheduling algorithm in the operating system which doubles the effort. At least for systems which profit from a high degree of flexibility, the task scheduling should be dynamic.

4.6.4.2 Partition Schedule

The scheduling of the partitions have to be strictly deterministic in accordance to Arinc 653. This is suitable for safety-critical systems to ensure that the partitions always keep in shape and safety-critical ones are never disturbed by non safety-critical ones.

4.6.5 Download

Download regarding embedded systems consists of two main parts:

- Download of Executables

- Download of Data

These two features differ in several ways because the first one has to be done before runtime, the second one can be done during runtime of the system. Furthermore, the executables are downloaded using external resource whereas the data are loaded by the application itself.

4.6.5.1 Download of Executables

Ideally, the download of the executables is done once and will never have to be done again for a system. By experience it has to be done at least for upgrades of the system. Therefore the requirements for a system should include an easy update solution which should be able to update every single part of the system.

The interface for such an update solution can be shared with the maintenance interface which is used for diagnosis.

4.6.5.2 Download of Data

The download of data can be an important functionality for several systems which need mission specific data (e.g. maps or flight routes).

Furthermore, not only download has to be recognized, but upload should also be supported to be able to handle log files or other diagnosis data.

4.6.6 Diagnosis and Debugging

Diagnosis and debugging are related but different approaches for error detection. Debugging has to be done during the development phase whereas diagnosis has to be executed during operational conditions.

The important point is that there is a feedback loop, where diagnosis information from the maintenance is used for improvements in the system development again.

4.6.6.1 Debugging

The verification and debugging of a system should always be done on a hardware which is exactly the same like the one in the runtime environment. This provides additional safety, making sure that hardware dependencies can not influence the behavior of the system.

The tools that are needed for debugging are communication and source code analyzers. They enable the development team to investigate every single anomaly to ensure the correctness of the system in every situation.

Furthermore, problems in the schedule, like critical situations or peak loads, can be uncovered in an early state of the development and countermeasures can be taken.

4.6.6.2 Maintenance

The interface for maintenance has to be comprehensive but easy. Every anomaly has to be logged, stored and provided for the corresponding person to evaluate the problem. This ensures a continuous improvement of the whole system.

Additional tools may be suitable to optimize the maintenance process and to simplify the error detection.

4.6.7 Conclusion

This section discussed different approaches for development tools and described a set of tools which are suitable for the development and operation of aircraft functions.

The tool chain is an important development instrument which is also useful and needed during the operational phase. The existence of such tools increases the efficiency of the development and provides additional safety for the final system.

4.7 Examples

4.7.1 TTA - The Time Triggered Architecture

4.7.1.1 Introduction

The TTA provides a computing infrastructure for the design and implementation of dependable distributed embedded systems [74]. It has been developed over a period of 25 years at the Vienna University of Technology [84] under the aegis of Prof. Kopetz.

The TTA infrastructure enables the implementation of applications and provides constraints to partition them into nearly autonomous subsystems. To be able to control the complexity, partitioning will be applied along small and well-defined interfaces which provide composability.

The subsystems are connected by a time-triggered, fault-tolerant communication system like TTP, FlexRay or TT-Ethernet. The communication system has to provide a set of services. The most important is a fault-tolerant global time base of known precision at every node. This global time base is used for communication, error detection and the timeliness of the real-time applications.

4.7.1.2 Architecture Model

As mentioned above, the TTA provides constraints for the development of distributed applications:

- Model of Time: The TTA uses the model of a sparse time base, which divides the time into durations of activity and silence. All events that occur happen in a duration of activity. From a global point of view the events which happen in the same duration of activity even on different nodes are considered simultaneous. Events that happen on different durations of activity, which means there is a duration of silence in between, have a consistent temporal order.

- Time and State: The sparse time base provides a system wide notion of time which furthermore supports a consistent state of the distributed system. This consistent view of time and state is important if fault-tolerance is provided by replication.

- Structure of the TTA: The TTA consists of nodes, clusters and the communication system. A node contains a processor with memory, an input-output system, a time-triggered communication controller, optional an operating system and the application software.

  The nodes which are connected over two replicated communication channels form a cluster. The communication system is based on a TDMA (time-division multiple access) schedule which periodically sends messages. It reads a state message from the CNI (Communication Network Interface) of the sending node and sends it to all other CNIs in the cluster and replaces the last version of these message. Every node has a global view on the communication schedule included in a schedule table which is called MEDL (message descriptor list).

- Interconnection Topology: The network topology of TTA can be a bus or a star. Using the bus topology, every node has a bus guardian, which is an independent unit that monitors the temporal behavior of the node. The star topology uses two central bus guardians which monitor the temporal behavior of all the connected nodes.

4.7.1.3 Design Principles

A main purpose of the TTA is to provide a consistent view of the system for all correct nodes in the system. To achieve this, a membership service is necessary. If the consistent view of the system is not possible, the fault-hypothesis is violated and the application is informed about the situation. Then it can decide how to proceed to reestablishing consistency as soon as possible.

The most important interface in the TTA is the CNI, which is the interface between the communication controller and the host computer. The control flow of this interface is unidirectional and no control signals ever cross the CNI. Therefore, the application can never be interrupted by the communication system which ensures that the propagation of control errors is prohibited by design. Furthermore, the CNI, which provides a well defined interface in the spatial but also in the temporal domain, supports composability and scalability.

Safety-critical real-time applications need a transparent implementation of fault tolerance. Active redundancy and voting is a common approach. This is supported by the fault tolerance layer which is embedded in the CNI and remains transparent to the application in the TTA.

4.7.1.4 Communication

A communication system has to provide the following services to be suitable for the TTA:

- Fault-Tolerant Message Transport
- Global Time Base with Fault-Tolerant Clock Synchronization
- Membership Service
- Clique Avoidance

4.7.1.5 Fault Tolerance

The fault hypothesis describes the types and numbers of faults the system should be able to tolerate. If the fault hypothesis is violated, the TTA activates an application specific never-give-up strategy.

Fault-tolerance is supported at design level in the TTA. The architecture provides active redundancy, fault-isolation and error detection independently from the application.

4.7.1.6 TTA Design Methodology

TTA provides a two-level design approach [117] which supports composability and the reuse of nodes and applications. The TTA distinguishes between the cluster design and the component design level.

- At the cluster level, the system integrator designs the physical cluster layout, the subsystems, and the interactions between the subsystems.
- At the component level, the suppliers design and implement the subsystems on the basis of the subsystem interactions, precisely specified in the cluster design.

In the cluster design the system is decomposed into clusters and nodes and the interfaces (CNI) are defined in the value and time domain. During the component design phase, the application software for the host computers is developed with respect to the CNI specification. The integration and validation is supported by this design methodology because of the well defined interfaces.

Two-Level Design using the V-Model The V-Model in figure 4.2 on the following page illustrates the different levels of system design. The big line indicates the separation of the cluster level and the component level.

The important fact regarding this approach is that the system integrator, which develops the system specification, does not need to provide his suppliers with this information. Based on the component interface specification, each supplier is able to develop a subsystem and provide it to the system integrator without passing proprietary information. Due to composability, which is an important aspect of the TTA, the system integrator composes the different subsystems, based on a system specification, to the entire system.

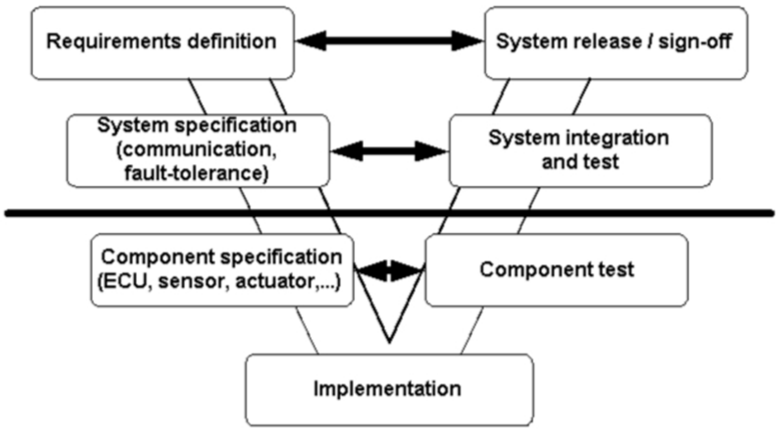
Requirements definition
System release / sign-off
System specification (communication, fault-tolerance)
System integration and test
Component specification (ECU, sensor, actuator,...)
Component test
Implementation

Figure 4.2: Two-Level Design using the V-Model

Reusability A crucial aspect for the software development for aircraft functions is reusability which leads to the use of off-the-shelf products. The main concept for the use of such software subsystems is composability which needs development methods that guarantee the stability of their interfaces. As already mentioned, the TTA uses the CNI with its defined messages to provide this stability. Based on the strict specification of the interfaces, these subsystems can be updated and reused according to their message definition.

4.7.1.7 TTA - Platform Components

The TTA provides solutions for dependable hard real-time requirements which consist of:

- Time-Triggered Architecture TTA / Time-Triggered Protocol: The design specification of the TTA and communication protocol TTP build the basis for the development.

- Communication Controller: The TTP-controller allows fault-tolerant time-triggered communication with membership service and distributed clock synchronization. The controller is certified according to aerospace standards DO-178B [96] and DO-254 [97].

- Operating System and Middleware: Certified software supports the development of TTA based products.

- Development Tools: The TTP-Tools are an integrated TTP software development suite which allow a seamless integration with other design and development systems.

4.7.1.8 Conclusion

The TTA is an established architecture for safety-critical systems. It has already been used in several automotive and aerospace applications, provides certified software solutions and its algorithms are mostly verified by formal methods.

For the use as IMA platform [126], some services are not flexible enough, which was addressed by the L-TTP/L-FlexRay approach (section 3.5 on page 29). Partitioning on operating system level is not innately supported and the reuse of legacy systems needs additional effort.

Decos, which is based on TTA, tries to address these points and to improve their architectural constraints. Decos is discussed in section 4.7.3 on page 76 in detail.

4.7.2 Spider - Scalable Processor-Independent Design for Extended Reliability

4.7.2.1 Introduction

Spider is a general purpose fault-tolerant architecture which has been developed under the lead of Paul Miner at the NASA [10] Langley Research center [81].

The most important and also most advanced feature of Spider and its communication system Robus, which is discussed in section 3.8 on page 38, is that all used algorithms are formally verified. This approach guarantees a predictable behavior in every possible state.

Based on this work, it is shown that formal methods are suitable in this field and therefore they will become more important for safety-critical systems and maybe support or even be a condition for certification in the future.

4.7.2.2 Architecture Model

In Spider, every Processing Element (PE) is connected with the Robus communication system. As already mentioned, Robus is a TDMA based broadcast bus with a time-triggered schedule.

An advantage of Spider in opposite to the TTA is its fault assumption. Spider is able to tolerate several combinations of simultaneous faults, whereas TTA only tolerates a single fault at a time.

Based on a global view of the system which is provided by the health information of every PE, diagnosis information for the whole system is available.

4.7.2.3 Design Principles

Safety and formal correctness were the major design driver of Spider. The communication is based on interactive consistency which is also known as "Byzantine Agreement":

- Agreement: For any message, all non-faulty receiving nodes will agree on the value of the message.
- Validity: If the originator of the message is non-faulty, good receivers will receive the message sent.

Based on this agreement and a global time, every non-faulty node receives the same messages and agrees on their validity. This provides a global state of the system which is a prerequisite for active redundancy.

4.7.2.4 Communication

The communication system Robus provides the basic services which are used for the fault-tolerance of the architecture. Based on the specification, each PE is connected with every other PE in the system with a direct link. This leads to a high degree of needed resources but guarantees the availability of the service even if several faults arise at the same time.

In such critical situations, Spider allows a dynamic reconfiguration of the communication schedule, which eliminates less important functions to ensure the correct operation of critical functions.

4.7.2.5 Fault Tolerance

As already mentioned, Spider including Robus provides the highest degree of fault-tolerance compared to other architectures and communication systems discussed. Furthermore, Spider prevents the propagation of faults using fault-containment regions and is able to detect communication cliques.

4.7.2.6 Services

Spider provides a set of services to support inter-dependent applications with different classes of criticality:

- Robust Partitioning on Communication Level
- Fault-Tolerant Clock Synchronization
- Clique Detection
- Consistent System Diagnosis
- Fault-Containment Regions

4.7.2.7 Conclusion

Spider is the most advanced architecture in terms of fault-tolerance and safety that is currently available. It was designed for the use in space missions with a long mission time under difficult circumstances.

This leads to the disadvantage that it is too expensive for the use in aerospace applications but this research project creates new standards for the development of safety-critical architectures in terms of formal verification, used algorithms and services, and numbers and types of tolerated faults.

4.7.3 Decos - Dependable Embedded Components and Systems

4.7.3.1 Introduction

The European research project Dependable Embedded Components and Systems [79] (DECOS) develops an integrated architecture for safety-critical systems.

The key features are:

- Hardware Cost Reduction: Based on an integrated architecture, COTS hardware is used which hosts several applications on the same processor.
- Reduction of Wiring and Connectors: By reducing the hardware, also wiring and connectors are removed which reduces connector problems as well.

- Improved Diagnostics: Diagnosis is also integrated into this approach. Using data from every part of the system enables a global view and an effective way to detect, identify and classify faults.

- Flexibility: The possibility of dynamic reconfiguration of subsystems in the architecture provides a high degree of flexibility. Furthermore, subsystems can freely be arranged on any processor in the architecture.

- Fault-Tolerance: Based on dynamic reconfiguration and free arrangement of subsystems, safety-critical subsystems can be executed on every processor in the system. Therefore hardware faults can be tolerated while a minimum of operational processors is available.

  Furthermore, active redundancy is supported by the partitioning approach because replicated partitions can be located everywhere in the system.

- Quality of Service: Based on the communication between every application in the system, a tactic coordination of tightly coupled control activities is possible.

According to these requirements, Decos wants to develop an integrated architecture for the use in the aerospace and automotive domain, based on the services of the TTA.

4.7.3.2 Architecture Model

Decos is based on Distributed Application Subsystems (DAS). These are a set of nearly-independent subsystems which, connected by a communication system, provide the overall functionality of the application together. In terms which are used in this thesis, a DAS is a subsystem of an aircraft function, like a single partition including its application.

The structure of a node in Decos can be seen in figure 4.3 on the following page. Comparable to the Arinc 653 partitioned OS, Decos provides a partitioned operation system called Encapsulated Execution Environment (EEE) [39][40]. The subsystems are divided into partitions providing one difference to the Arinc standard that partitions are divided into safety-critical and non-safety critical ones using the architectural high-level services [83].

Below these layers, the core services are located which are independent of the DASs providing fault-tolerant services like:

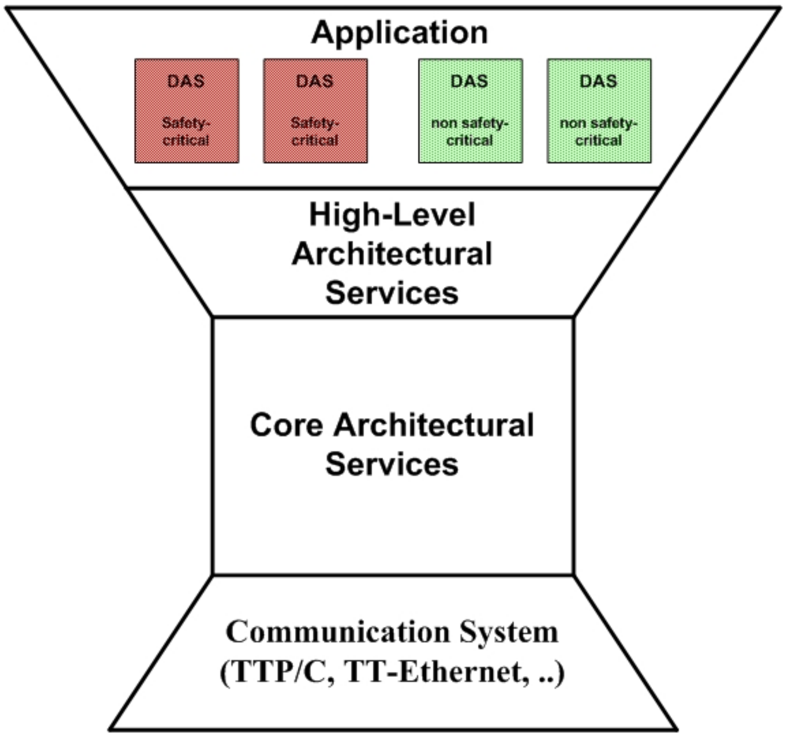
Application
DAS
Safety-critical
DAS
Safety-critical
DAS
non safety-critical
DAS
non safety-critical
High-Level Architectural Services
Core Architectural Services
Communication System (TTP/C, TT-Ethernet, ..)

Figure 4.3: Structure of a Decos Node

- Deterministic and Timely Message Transport
- Fault-Tolerant Clock Synchronization
- Strong Fault Isolation
- Consistent Diagnosis of Failing Nodes

These core services have to be provided by the underlying communication system which is independent from the layers above.

The architecture consists of such nodes, connected by the communication system. The focus is on independence of the layers to provide a high degree of flexibility in the choice of the parts of the system.

4.7.3.3 Design Principles

The design principles for the Decos hardware are the current and future requirements of the automotive and aerospace industry. Based on this, an architecture is developed which focuses on the reduction of costs and weight by increased flexibility and safety.

Based on the core services of the communication system, an architectural approach is created which tries to address requirements from the named domains. Important issues are the support of legacy systems, reduced hardware and wiring, improved diagnostic and maintenance and extended services.

The trade-off between safety and flexibility is tried to be disarmed by increasing flexibility in the communication and the operating system without reducing the level of safety.

4.7.3.4 Communication

The only requirement for the communication system is to provide the core services. For research aims, TTP, FlexRay, respectively L-TTP/L-FlexRay, or TT-Ethernet are considered as a communication system because they provide the needed services and are well known in the field of safety-critical systems. In general, every communication system which provides these services can be used.

Decos also considers a virtual network service which is a network layer on top of the time-triggered communication layer providing the core services. This layer abstracts the communication service for the application from the physical implementation. A connector unit provides time-triggered and event-triggered communication channels to the application. This supports the reuse of legacy systems.

The physical communication, based on the inter-node communication and the core services, is handled by this communication layer. This abstraction frees the application from interdependence with the communication system which increases flexibility and reduces complexity for the application designer.

4.7.3.5 Fault Tolerance

The system is designed to support active redundancy using the core services and the encapsulated execution environment. Furthermore, fault-containment regions, for strong fault isolation, are provided on partition level by the operating system.

The integrated approach provides improved diagnostics supporting a global view of the architecture. Due to universally available resources, fault-tolerance is supported by better replication strategies, like dynamic reconfiguration of safety-critical subsystems.

4.7.3.6 Services

Based on its core services, the TTA is appropriate for the implementation of a distributed and integrated platform solution which is able to host aircraft functions. Figure 4.4 shows such a platform which provides services for the subsystems

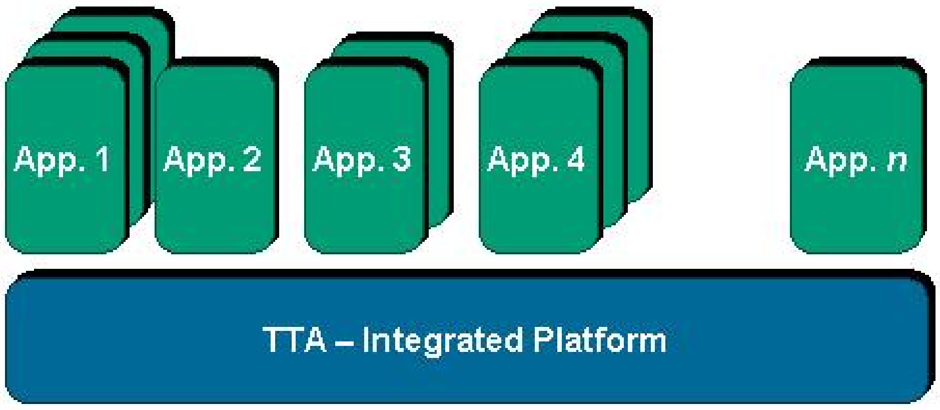

Figure 4.4: Integrated Platform Solution

on top. This system platform distinguish between safety-critical and non safety-critical subsystems which have differences in their failure modes and needed services. Minimal services for safety-critical subsystems force predictability, dependability and minimize certification efforts. On the other hand, non safety-critical subsystems with extended services increase efficiency and flexibility.

4.7.3.7 Conclusion

Decos is a research project at the moment but provides several interesting approaches which will influence upcoming projects in this domain. The outputs of this project should be used for the development of a distributed and integrated platform for aircraft functions.

In addition, Decos also provides ideas regarding upcoming platforms in the automotive domain.

4.8 Conclusion

The system architecture provides the theoretic environment for the platform. Based on the maturity of this environment, the complexity of the implementation can be significantly reduced. In case of an integrated architecture, the developed platform contains most of the complexity in contrast to the federated architecture, where all of these issues have to be addressed by the application designer himself.

According to the possibility of using a given platform including a communication system, support for COTS hardware, a development tool suite and the appropriate certification package, the major complexity and therefore the focus remains on the aircraft function. This reduces the effort and the development time and therefore the costs by increasing the dependability and the safety.

Chapter 5

Modular Certification

If you were born on an airliner in the US in this decade and never got off you would encounter your first fatal accident when you were 2300 years of age and you would still have a 29% chance of being one of the survivors.

(Les Lautman)

5.1 Introduction

Certification is the way the authorities, like the FAA [2] in the United States and the EASA [12] in Europe, verify the development of software and hardware which will be used in aerospace applications. The authorities demand standards for development processes which have to comply to Federal Airworthiness Requirements FAR 23 [4] and FAR 25 [3].

The authorities monitor these processes by review meetings which are conducted by Designated Engineering Representatives (DERs). DERs are experienced engineers designated by the authorities to approve engineering data used for certification.

After the DER has checked and accepted the different steps of the process, the generated artifacts are shipped to the authority. The authority has to review the whole aircraft certification data and award the product certification if everything is approved.

5.2 Software Certification in the Aerospace Domain

5.2.1 Introduction

Software quality is a widely discussed topic, especially in the aerospace domain where a fault in the software can lead to catastrophic behavior. Therefore, several approaches have been developed to ensure quality for aerospace software.

The most common approach is certification according to DO-178B [96], which defines guidelines for different phases of software development. The main idea behind such process definitions is to have a controlled way for the planning, the design, the implementation and the verification of software where all steps are reasoned by at least two persons, the executing one and the reviewing one. This effort provides confidence in understanding the system behavior under several conditions. The ultimate goal of certification is that the developed software is fully predictable. This means that it is known in advance what the application does in every situation, even in critical ones.

DO-178B is a means of compliance with airworthiness requirements which is acceptable for the aviation authorities and therefore is treated like a standard. Every process that has been chosen to provide evidence has to ensure compliance with the objectives of DO-178B.

A rather new guideline for the certification of integrated modular avionics is DO-297 [99], which considers the certification of integrated systems and architectures. This approach, called modular certification, allows the independent development and certification of aircraft functions and their corresponding integrated architecture which hosts these safety-relevant applications [43].

A major problem for such software projects, which are going to be certified according to any standard, is the effort that is much higher than for common software projects. Based on this and the fact that an authority has to check the system and the used processes, too, the costs for such a project increase with every level of criticality. Especially for Level A certification, which is the highest criticality class in DO-178B, the effort is more than two times higher than for the same software development without certification [55].

According to the high amount of certification costs and effort, it is interesting to know which parts are the most critical ones in the development and certification process and which effort can be saved through an efficient process. This would allow to improve the planning, reduce design faults in the beginning and minimize the risks of such a project.

5.2.2 DO-178B

Based on the DIMA approach discussed in this thesis, software certification according to DO-178B [96][105] is one of the needed development standards. Considering the architectural approach, DO-297, which is described in section 5.3 on page 88, is essential for the development and certification of the platform. In terms of certification, DO-297 is used to verify the integration of the different modules and components, but their individual certification is very similar to DO-178B.

5.2.2.1 Overview

Radio Technical Commission for Aeronautics (RTCA) DO-178B is the certification standard which is used for commercial aerospace software. It defines guidelines for the processes which have to be used for the development of safety-relevant software.

Based on the objectives defined in the standard, a process has to be established which ensures that the software is sufficiently safe and understood. Sufficiency is defined according to software levels which are defined by the circumstances and the failure conditions of the software [50]. The software levels reach from Level A for catastrophic to Level D for minor failure conditions.

For a detailed description of the failure conditions and the corresponding process objectives [95], please refer to table 5.1.

Failure Conditions	**Software Level**	**Process Objectives**
Catastrophic	A	66
Hazardous	B	65
Major	C	57
Minor	D	28

Table 5.1: Software Levels and Objectives

According to these software levels and their objectives, the development process is defined. Therefore, the following stages of the project have to be considered in advance:

- Planning Process
- Development Process

  - Requirements Process
  - Design Process
  - Implementation Process

- Verification Process
- Configuration Management Process
- Quality Assurance Process
- Customer and Certification Liaison

Especially the requirements and the design phase are of decisive importance and should not be switched into the implementation phase too early. The requirements and the design should clarify the most likely problems and provide a solution, that is clear and easy to understand. Every finding in the beginning reduces the effort for verification and rework, which can, in the worst case, strike the customer.

Figure 5.1 shows how the costs for a detected fault in the system increase if it is found later in the process or even in the field [25][89].

Another major aspect to enhance quality is the feedback at the end of the project.

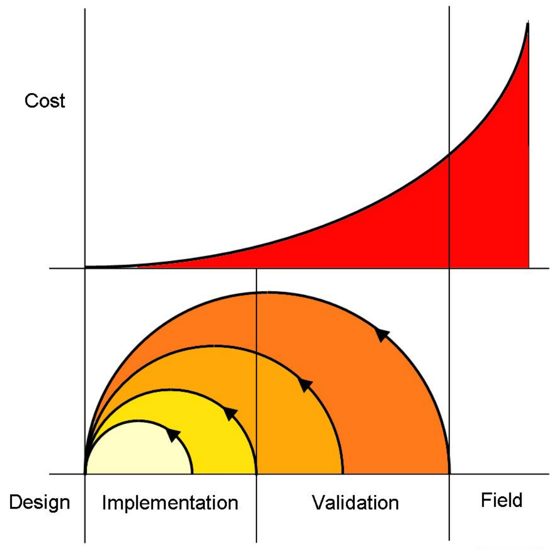

Figure 5.1: Costs for Recognized Faults

It is not directly part of the standard but helps to improve planning estimations and to customize the process to fully comply with the individual work flow and project structure.

5.2.2.2 Reuse of Previously Developed Software

The reuse of already developed software and the corresponding certification package is a solution which is granted by the authority. According to Advisory Circular AC20-148 [9], it is possible to certify software as reusable software component [94][1].

This approach provides the advantage that the certification effort for projects are minimized after the completion of the first project and the corresponding approval. If this approval has been successful, the authority provides a letter of certification credit to the applicant which either reduces or eliminates the certification effort for the following projects, using the same piece of software.

The disadvantage of this approach is that the initial effort for certification of a reusable software is higher than for common software certification [68]. Furthermore, it is possible to formally or informally reuse parts of the already developed software life-cycle data for follow-up projects including the same source code. Therefore, a certification according to AC20-148 is rarely done.

The additional effort for certification of a reusable software component is based on additional coordination, extended reviews and analysis by the developer, integration effort with corresponding applications, reusable software components change and post certification issues, document retention and increased effort in tracking compliance.

Certification for a reusable software component only makes sense if the software does never change and is used very often, like control algorithms for example. It has to be kept in mind that the reduction provided by this approach starts at the second time the software and the certification is used.

COTS-Components: Commercial-Off-The-Shelf (COTS) software [5] is a special form of previously developed software, as it can be developed independently from the corresponding application. Operating systems, for example, can be considered COTS, providing certification credit like service history and a verified development life-cycle.

As already mentioned, COTS software needs to have the same software level as the application it supports. Furthermore, service history provides additional credit but does not preclude the presence of unintended functionality and therefore does not fully satisfy certification objectives. Based on these constraints, certification evidence is needed for COTS components, too.

5.2.3 DO-178C

According to rapid changes in the aerospace domain, the working group RTCA SC205/EUROCAE WG-71 is currently discussing updates of the DO-178B standard. Apart from minor changes several major issue are under discussion like:

- Conflation of DO-178 and DO-278 [98]
- Increased use of COTS Software
- Use of Formal Methods
- Use of Object-Oriented Technology
- Use of Model based Design and Verification
- Update of Tool Qualification

5.2.4 Conclusion

This section describes the current standard used for software certification in the aerospace domain. DO-178B, which is this standard, defines development processes to ensure high quality and therefore safety for every artifact developed.

Although there are new standards, like DO-297, DO178B or its successor DO-178C are still suitable and needed for aerospace software development. DO-178C will provide new approaches in different areas of certification but the main idea regarding multiple validation of every artifact will be kept.

5.3 DO-297 - Integrated Modular Avionics Development Guidance and Certification Considerations

5.3.1 Introduction

Integrated Modular Avionics based systems are already in use and are rapidly expanding. Due to this, it is necessary to have a standard for an IMA based development process. DO-297 [99] was developed to address this problem and to define a process for the development and certification of modular software and hardware for avionics functions.

The main problem of modular certification is to prove that a set of modules or subsystems does not affect each other. This is equal to the problems arise by partitioning at operating system level. Rushby [103][104] identified this problem and discussed its solutions.

5.3.2 Overview

Modular certification according to DO-297 is a rather new approach based on the need of certification for integrated modular avionics and the corresponding system architectures [22].

DO-297 breaks down the whole system into the following levels to map the modular approach:

Module Acceptance: A module is a component or a collection of components which can be software, hardware or a combination of both which provides resources to the application and/or the system platform.

Application Acceptance: An application is based on modules and performs a function.

System-level Acceptance: The system-level consists of one or several platforms which provide a computing environment, managing resources for at least one application. Furthermore, it establishes support services and platform-related capabilities like health monitoring and fault management.

Aircraft-level Acceptance: The aircraft-level considers the integration of the system into the aircraft and its systems.

Furthermore, it defines criteria for the change and the reuse of parts of the system.

5.3.3 Architectural Considerations

DO-297 defines an architectural approach which enables the certification of reusable modules and applications. The needed functionality is established by connecting the single parts of the distributed application using a communication system.

The integrated architecture provides constraints to partition the aircraft function into nearly autonomous subsystems which can be certified independently. To be able to control the complexity and to support the independent certification, partitioning will be applied along small and well-defined interfaces which have to provide composability [74]. The main concept behind this is: "architecture design is interface design".

Based on this distributed approach, some requirements are of major importance [70]. These requirements are:

Partitioning:[101][8] Robust partitioning has to be provided on system level to ensure that different applications are not able to interfere with each other. Using partitioning, the platform shall establish a computing infrastructure for every application like in a federated architecture.

Data and Control Coupling: Another aspect of partitioning for distributed systems is data and control coupling [26][48]. If several modules use the same data sources, dependencies have to be analyzed and verified. Furthermore, different modules can not influence the execution of each other.

Composability: Composability means that an application can be decomposed into small, distributed subsystems which provide the wanted functionality together. A system which supports composability must adhere to following four principles: Independent development of modules, stability of prior services, constructive integration of the modules to generate the emerging services and replica determinism [88], which deals with active replication of subsystems.

Interface Definitions: Well-defined interfaces are a condition to support partitioning, composability and data and control coupling are needed for every distributed approach. Necessary glue code reduces the performance and the safety of a system and can be avoided through correct interface definitions.

Using an architectural approach forces the reuse of legacy systems and provides the possibility of modular platforms. Therefore, the certification activities have to consider the certification of modules and especially their integration into the platform.

The certification of single modules in this approach is fairly similar to the certification effort needed for a certification package according to DO-178B [96] for SW or

DO-254 [97] for hardware modules. Therefore, the reduction of certification effort is firstly given at the second use. Furthermore, the communication system which connects the modules needs to become fully approved.

5.3.4 Integral Processes

5.3.4.1 Safety Assessment

Safety Assessment defines standards which have to be used for the creation of the Safety Assessment Analysis. Recommended standards are ARP4754/ED-79 [50] and ARP4761 [28]. Furthermore, the responsibilities of the different parties and their activities are defined.

5.3.4.2 System Development Assurance

This process establishes the development guidelines for hardware and software modules in IMA systems. Furthermore it discusses the interrelationship of different parties during the development of modules. Shared design, integration and testing in accordance with the given environment are described.

5.3.4.3 Validation

The validation process ensures that the requirements are correct and complete. It identifies the validation activities for every level of system development (see section 5.3.2 on page 89) and the artifacts which need to be validated.

5.3.4.4 Verification

The verification process ensures that the implementations of specified requirements have been met. It identifies the verification activities for every single part of the acceptance steps (see section 5.3.2 on page 89) and the artifacts which need to be verified.

5.3.4.5 Configuration Management

This process addresses the configuration management activities for the development of the IMA system and the modules it consists of.

5.3.4.6 Quality Assurance

This process addresses the quality assurance activities for the development of the IMA system and the modules it consists of. Quality assurance needs to pay increased attention that all levels of system development (see section 5.3.2 on page 89) and their integration provide compliance to the needed safety level.

5.3.4.7 Certification Liaison

This process defines the interaction between the applicant and the certification authority. Furthermore, it describes typical development life-cycle data which is needed for acceptance of the developed system.

5.3.5 Conclusion

The modular certification approach is an important step to support the development needs for future aerospace applications. It provides the baseline for the use of (distributed) integrated modular avionics and ensures their modular structure.

Furthermore, essential concepts like partitioning, reuse of modules, modularity and integration are defined. Therefore, it will become a commonly used standard in the future.

5.4 Parameters for Efficient Certification

5.4.1 Introduction

This section is based on a survey regarding efficient certification [125], supported by about 40 experts in this field. The main focus is on software but the results are also suitable for hardware certification according to DO-254 [97]. Based on the fact that certification of software or hardware systems is fairly similar to module certification according to DO-297, this results should also be considered for modular certification.

5.4.2 Efforts and Savings

Based on the results of the survey and valuable data from several certified software projects [49][56] the efforts for the software development have to be divided according to figure 5.2 including validation steps for each of the phases.

The key phases, where most effort can be saved, are the requirements and the

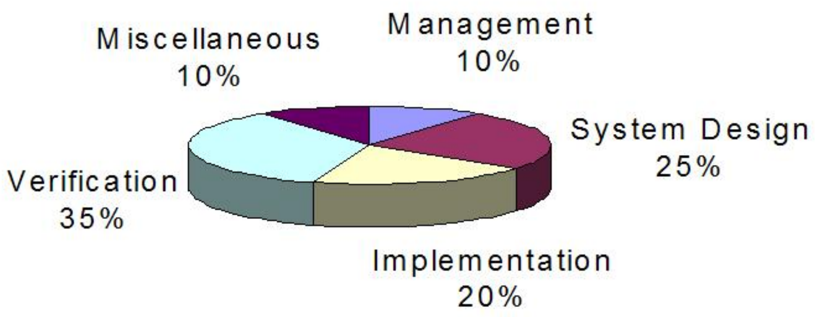

Figure 5.2: Efforts of Software Development Process

verification phase.

The efforts which are saved in the requirements and design phase can be achieved by increased attention, which means that the implementation is not started too early, and tool support, which provides a structured environment for the creation and validation of the system design.

In the verification phase most of the effort can be saved by using an automated test suite. According to the costs of such a suite an appropriate solution has to be considered which is suitable for the current but also for upcoming projects.

5.4.3 Requirements, Design and Traceability

5.4.3.1 Introduction

The requirements and design phases at the beginning are the most important parts in the software life-cycle process. The requirements define the outcome and therefore need to be clear and easy to understand. The design is derived from the requirements and describes how they should be implemented. Every fault or ambiguity in this phases returns later on with highly increased effort.

Requirements are the building blocks of the system. Therefore, the quality of the system depends on the quality of every single requirement. The design consists of detailed requirements, which have to be traceable to the high level requirements mostly, and design components which describe complex algorithms and data structures to support the understanding. Another major point is the traceability from the requirements to the design and further on to the test cases, down to the source code. This ensures that nothing is missing and everything has a reason for its existence. To ensure a constant quality level for the requirements and guarantee traceability through the process, some basic points have to be considered.

5.4.3.2 Tool Support

A database centric requirements management tool provides a lot of advantages to the development and certification process. Firstly, several process steps are already included in the tool and therefore the formal handling is simplified. Furthermore the waterfall based top-down life cycle process can be split up which allows to move forward from requirements to implementation and verification without the need of showing consideration for other parts of the system. Furthermore such a tool checks that all relevant traceability information is available. Additionally, some of these tools provide the possibility of creating an evidence media which contains all necessary life-cycle and traceability information in an easy to review form. According to this efficient way to deal with the process and based on the experts judgment, the effort for these steps can be optimized by between 2% and up to 20% with respect to the process used before.

5.4.3.3 Standardized Requirements Definitions

There should be standards for requirement definitions which provide guidelines for requirements engineering to ensure their quality. Furthermore, each requirement has to be self contained because this supports the verification of each requirement.

5.4.3.4 Design Components

If the requirement describes complex functionality, the developer should add definitions, figures and information which support the understanding. This encourages the demand for self contained requirements and helps to comprehend the whole system.

5.4.3.5 Testability

The requirement has to be checked for testability. This must be done by the requirement developer and especially by the reviewer. The easiest way to handle this is to write functional test cases in parallel to the requirements to find testability problems at an early stage of the requirements process. If this is not possible, the developer should at least give advice regarding what to test to the verification staff.

5.4.3.6 Conclusion

I conclude that, based on experts judgment as well as own experience and research, considering these points in the requirements and design phase can definitely reduce the effort and the costs for the overall project by up to 30% because a clear system design decreases the effort for implementation and verification.

5.4.4 Verification and Validation

5.4.4.1 Introduction

The verification phase is commonly the phase of the software life cycle where most work has to be invested. Based on estimations, the effort for this part of the software life cycle is up to 50% of the overall project outlay.

Independent from the used tool, the verification effort can be minimized if the requirements, the design and the implementation are kept simple and clear (see also 5.4.3 on page 93). In this case, every function can be tested by its own low-level and additional high-level tests, which verify the required functionality to complete the test of the whole system. Additionally, some robustness test cases are needed, where boundary values are checked.

Another very important part is the validation phase. These reviews ensure the quality of every generated artifact and provide the transition to the next phase. As already mentioned, especially increased effort for requirements and design review pays off later on.

According to these facts, there are several points to consider which help to carry out these steps efficiently.

5.4.4.2 Test Suite

A major concern regarding the verification process is the use of a test suite. The advantage of such a tool is the possibility of automatic verification of the test cases and their structural coverage. Considering that the tool qualification package, that has to be provided to the authority, is already available, which is most common for suitable tools, the verification effort can be optimized between by 5% and up to 20% based on the process used before. These numbers are the outcome of the research survey.

5.4.4.3 Verification Tools

The use of verification tools is an interesting aspect of DO-178B. It provides the possibility of having complex algorithms, like schedulers, easily certified. The verification tools must verify the results of these algorithms to prove their safe and deterministic behavior. Furthermore, a tool qualification package is needed for the verification tool suite, which provides confidence about the tool. The verification tool and its tool qualification package are mostly less expensive if the verification for correctness has to be done several times, compared to verifying the algorithm itself.

5.4.4.4 Review Criteria and Checklists

The criteria for reviews should be clear, suitable, applicable, detailed and have to be stated on the corresponding checklist. Furthermore, it is sometimes useful to have detailed checklists, which name every single review item and every corresponding review criteria, including additional information about what and how to review the items. This increases the quality by enhanced efficiency.

5.4.4.5 Informal Reviews

Informal reviews of the life-cycle artifacts disclose major problems and increase the quality in an early stage. The overhead of a formal review process for immature artifacts is minimized and necessary evidence is provided in a formal, second stage. This approach provides the same quality like doing every review in a formal manner, but more efficiently.

5.4.4.6 Conclusion

According to these points, I conclude that the reduction of effort and therefore costs is up to 15% of the overall project if the verification and validation is done in an optimized way.

5.4.5 Optimized Processes and Continuous Process Improvement

5.4.5.1 Introduction

A well set-up process and its continuous improvement are key tasks for the development of high-quality software. There are several points to consider which help to optimize the process and therefore increase the quality and efficiency [47].

5.4.5.2 Quality Culture

Establishing a quality culture is a major issue considering safety-critical software. It takes some time to train, introduce and establish the processes and customize the work flow but finally it is an opportunity for the whole software development. This cultural approach increases the quality and efficiency considerably if everybody who is involved feels responsible for producing high quality artifacts. The survey's outcome is that, once established, a quality culture decreases the effort of the whole software development process by 5% to 20%.

5.4.5.3 Continuous Process Improvement

The defined and used process should not be set in stone. It has to be customized if a better approach is found or new tools are used. After the end of the project, a feedback loop has to be installed to verify the changes for their positive or negative influence. According to expert's judgment, if continuous process improvement has strong management support, possible improvements can be up to 20% in three years, depending on the maturity of the currently used process.

5.4.5.4 Training

Training ensures that every person involved in the process fulfills his role and helps to complete the project with the needed quality. Ensuring constant high quality, every person involved should receive customized training according to its special needs. Based on my research, appropriate training increases the quality and therefore the efficiency by 2% to 10%, depending on the experience of the staff.

5.4.5.5 Conclusion

Concluding the points described in this section, I am of the opinion that optimized processes do not only save effort up to 15% in a single project, but also increase the quality in a sustainable manner on the long term.

5.4.6 Conclusion

There are several possibilities for efficient software certification by modifying and optimizing the process, considering several criteria described. These criteria name the phases of the process, where most of the effort is needed and especially where most of the effort can be saved.

The key aspect is that additional effort in the requirements and design phase, which is spent at the beginning, pays off for the software development process and the whole product lifetime. Furthermore, it is possible to save a lot of effort in the verification phase using appropriate tools.

All parameters that are presented in this section are based on research and assume the saved effort comparing a new process to a well established one. Achieving these goals might take some time but it will increase efficiency of development and the quality of the product.

5.5 Comparison of Survey Assumptions and Actual Results

This section compares the assumptions from the survey with actual results from real software development and certification projects. According to this comparison, the assumptions are validated. Two types of projects were chosen which have comparable size. There first comparison is based on two small sized projects with less than 1000 effective lines of code (eLoc) and the second source are two medium sized projects with about 4000 eLoc.

The major difference between the comparable projects is that one of them was the first approach of development according to aerospace requirements and the other one is developed with 2 to 3 years of experience in this area. Based to these projects and its results in several phases the possible gain of efforts and costs, based on advanced processes, increased knowledge and additional tool support, is evaluated and the assumptions are validated or discarded.

5.5.1 Requirements

According to the survey, the requirements phase can be optimized with different approaches. The one which have been selected for evaluation are those which are considered to provide the biggest enhancements. These approaches are:

- The Introduction of a Requirements Management Tool
- Several minor enhancements like:
  - Standardized Requirements Definition
  - Testability of Requirements
  - Design Components

The evaluation, based on the compared projects, provided interesting results. The effort lost, based on bad developed requirements in terms of no standardized requirements, requirements not reviewed for testability and the absence of design components is significant higher than expected.

The use of a requirements management tool supports the process and reduces the effort but not as significant than the problems named before.

In summary, good requirements reduce the effort of the whole project up to 40% compared to a project with bad requirements.

5.5.2 Verification and Validation

The verification and validation phase is the most extensive one in the software development and certification process. Therefore, it is necessary to find the key aspects which have the biggest influence to this phase:

- The Introduction of an Automated Test Suite
- Several minor enhancements like:
  - The development of Verification Tools instead of tool certification.
  - Review Criteria and Checklists
  - Informal Reviews

Although this phased turned out to be the most extensive one, the results of the survey could not get confirmed. According to the high amount of needed effort, the process can be improved by using automatic verification tools and the other points described before but it does not reduce the effort to the expected results. Based on the comparison of the actual projects, a reduction of about 4% was observed.

This result includes the introduction of a requirements management tool which reduced the gained effort because a new review process had been introduced. If this process is well established, a reduction of up to 10% compared to the first project is conceivable.

5.5.3 Process

The process itself defines the development and certification environment. Therefore, it is from critical importance to review this area and evaluate its criteria. The major points which can be influenced are:

- The Introduction of a Quality Culture
- Continuous Process Improvement
- Training

The adaption and continuous improvement of the process are important factors in such development and certification projects. Based on the fact that the improvements may not directly be observed it is difficult to provide this information.

Based on the comparison of the projects and the efforts needed for them it can be concluded that the results of the survey are correct and the gained effort after three years is about 15%. The important fact regarding continous process improvement is the increase of efficiency and quality on the long term.

5.5.4 Conclusion

According to the results presented above, it is shown that there are several aspects which have major influence on the development and certification approach. Therefore it is essential to fit the process to the development team and improve the process continuously because this ensures that the efforts are reduced and the quality is increased on the long term.

The comparison of the actual project with the survey shows that the experts judgment has been correct in terms of the requirements process. Bad developed requirements lead to problems which turn up through the whole software life cycle process. Therefore, the approaches presented in 5.4 on page 92 lead not only to a significant improvement of the efforts needed for the project but also to increased overall quality.

On the other hand, deviations between the survey and the actual results can be observed in terms of verification and validation. Although it is possible to reduce efforts, the reduction does not reflect the assumption.

In conclusion, the total gain of efforts between the comparable, actual projects, based on process enhancements and advanced tool support is 50% which is a major part of such a project.

5.6 Conclusion

Certification is a common approach to ensure the safety of developed hardware and software. Applying the presented standards, a modular architecture and the software on top of it can be developed, certified and approved by the authorities.

An important possibility, according to high costs of certification, is to accomplish development and certification efficiently. There are several ways for efficient certification that were shown in this section. Using the described parameters allows to reduce the effort by at least keeping the level of quality the same. Therefore, it makes sense to use these parameters to adjust the development and certification process.

Another possibility to reduce the certification effort is an architectural approach according to DO-297. This modular approach deals with integrated architectures, enables a simplified development process for distributed systems and therefore allows a significant reduction of the certification effort for single applications.

Furthermore, all parameters for efficient software development and certification are also suitable for this modular approach, because the development process of a single component is very similar to the process defined in DO-178B.

It only takes five years to go from rumor to standard operating procedure.

(Dick Markgraf)

Chapter 6

A Distributed and Integrated Platform Solution

6.1 Introduction

According to the chapters 3 on page 19 where communication systems are described, 4 on page 44 where the system architecture, including hardware, middleware and application software and tool support, is discussed and 5 on page 83 where the certification process is explained, all prerequisites for a avionics computer platform are provided.

This chapter discusses now the system level requirements, attributes and benefits of such a platform.

6.2 Requirements and Recommendations

This section provides requirements and recommendations regarding the change of the system architecture. In addition to certification considerations and presented requirements, a change in the culture of the development teams has to be applied in terms of a holistic approach to ensure an optimal outcome [27].

6.2.1 Change of System Architecture

The transition from federated to integrated systems started in the late eighties. Further evolution toward physically distributed, but fully integrated avionics and control systems continues in new aircraft programs as a result of the needed reduction of lifecycle costs and weight.

Another advantage is the opportunity to optimize or change classical aircraft system design, as distributed functions can be fully integrated independent from placement and distances between subsystems. Due to the fact that open and standardized platforms [63] can reduce the effort for aircraft function development and provide new tools for advanced system integration, the system architecture and its properties become the source of competitive advantage for suppliers and system integrators and reduce the investment barriers for market entry for new aircraft function developers.

A major question regarding integrated systems concerns the costs for changing the system architecture. The initial costs for the development of such an architecture are very high. Furthermore, there are costs to integrate the function into the new environment for using a legacy system on the new platform. Therefore, the step of changing the system architecture has to be done very thoughtfully [27], by considering all possible costs and savings.

Another approach is to obtain the platform from a supplier. This has the advantage that the whole development and certification considerations are part of the purchased platform and the system developer does not have to care about it. The development concentrates mostly on the application instead of the overall system and therefore the needed effort for development and certification and thus the time-to-market are significantly reduced.

Advanced services like partitioning, communication and control flow are handled by the platform and their services. Therefore, costs for the development of an integrated function are the same or less than for a comparable federated one because additional hardware layers (e.g. OS, Drivers,..) have to be considered in the federated approach. In the integrated approach, this is already part of the platform and except for the interface specification it does not need to be considered at all. But additional costs for the development or purchase of such a platform is the other point that has to be kept in mind.

6.2.2 Reuse of Legacy Systems

A crucial aspect of aerospace software development is reusability [94][123] which leads to the use of off-the-shelf products [5]. The main concept for the use of such software subsystems is composability. Therefore, development methods are needed that guarantee the stability of the interfaces between these subsystems, which have to be provided by the system architecture.

Based on these partitioned resources, full reusability of legacy systems independent from their original core hardware and the interfaces can be supported. Considering the operating system, the communication between partitions and the communication between distributed hosts, abstraction layers have already been developed which provide several aspects of application environment without the need of major rework.

An important fact to support legacy systems is the need to support different interfaces. Therefore, an abstraction layer in the operating system and/or in the communication systems has to provide interfaces for several communication systems like AFDX/ARINC664 [19], Safebus/ARINC659 [58][15], ARINC629 [17] and ARINC429 [18]. This allows reuse of existing applications without major redevelopment and recertification.

Considering the reuse of a system, the effort for additional certification issues corresponds with the factor of software changes. Based on the fact that most of the certification data of the federated system can be reused with little modification, the only additional effort is the integration of the system into the new environment. Nevertheless, these costs for integration and updated certification are not negligible and have to be evaluated in advance.

6.2.3 Initial Efforts and Long Term Advantages

As already mentioned, the initial effort for the development or purchase of an integrated architecture is very high. Furthermore, already developed aircraft functions have to be updated and integrated into the new environment. Therefore, the initial efforts and costs for switching the architecture are several times higher than keeping the federated system.

On the other hand, there are several advantages of the integrated approach like modularity which allows the change or update of every single module without the need of a full system re-certification. Furthermore, these modules can be reused in other systems with the possibility of reusing most of the certification credit.

Because of these advantages, a more efficient airplane is possible which furthermore provides quite new design possibilities without a loss of safety.

6.2.4 Communication Infrastructure

The communication system for an integrated architecture or especially for a distributed integrated architecture has to fulfill several requirements. As already mentioned, network partitioning, data- and control coupling, network composability and interface design have to be solved on communication level.

Another important property for communication is security. Based on current problems and especially due to the fact that functions with different levels of criticality are hosted on a common platform, the secure transport of information is necessary. Therefore, a distributed communication system for open avionics platforms has to provide safe and secure [118] exchange of information. The next property which has to be considered are the types of communication systems needed to create a whole platform. On the one hand, a high performance system bus is needed which connects all subsystems. This also provides the advantage of increased diagnosis by evaluating the information shared by different subsystems. On the other hand, there has to be a low cost bus for subsystem communication. Currently automotive systems, which are cheap because of their high number of pieces, are evaluated.

Such a common communication infrastructure provides several advantages like lower costs, less interface problems and interface control by the aircraft manufacturer. Interface problems between different suppliers are currently a huge problem. According to this communication approach the aircraft manufacturer defines the interface specification which guarantees easier integration of every single aircraft function.

Furthermore it is connected by the low-cost bus to the system bus and therefore a reduction of wiring is possible because the interface data can be shared with every other application in the system.

6.2.5 Conclusion

This section describes the requirements and recommendations for changing the system architecture from a federated to an integrated one. The key criteria are the set up of an open architecture which allows to develop and certify applications according to the open and well defined interfaces and reuse already developed legacy systems. A key role plays the communication system which has to provide such open interface too, to be able to support not only several applications but also several types of subsystems.

6.3 Platform Attributes

Following attributes have to be considered during the development of a distributed avionics platform. Furthermore, these attributes support the evaluation process regarding a possible change from a federated to an integrated system.

6.3.1 Integrated vs. Distributed

An interesting aspect of such a platform is the possibility of distribution in addition to integration. As already discussed in the DIMA architecture, it is possible to have integrated systems which are physically distributed. This means that from the point of view of the application there is no difference if the inputs and outputs are directly connected or at another place in the aircraft. Therefore it is no contradiction to talking about an integrated and distributed platform.

6.3.2 Aircraft Function Development

The aircraft function development process in an integrated architecture contrasts to that in a federated one. In the federated approach, the subsystem, which provides the aircraft function, consists of the software application and the underlying middleware software and hardware, which includes directly connected interfaces to the environment. In the integrated approach, several applications share the underlying middleware software and hardware and the interfaces can be provided by another subsystem which shares it.

Therefore, the aircraft function developer primarily needs to consider the interfaces but does not need to care about the whole subsystem because all in- and outputs are provided by the underlying architecture. Based on this approach, the development process mainly consists of the interface specification, which has to be provided by the system integrator, and the application development. This allows a significant reduction of the development and certification process and ensures easier integration into the global aircraft system.

6.3.3 Platform Communication

The platform communication using a DIMA approach should consist of a high-performance system bus which connects all subsystems and a low-cost field bus for the communication within the subsystems. The advantage of such an approach is that standard components can be used all over the aircraft which reduces the development and production costs.

Furthermore, global diagnosis and easier integration are supported if all of the subsystem are based on the same communication system. This ensures efficient maintenance in terms of fault detection and replacement of defect systems.

6.3.4 Modular Design

The modular design approach, used for the development of integrated systems [126], enables the separate development of system architecture and subsystems design. Furthermore, it allows the strict control of key system interfaces and the separation of functional/logical from temporal behavior facilitates the reuse and seamless integration of electronic subsystems provided by different suppliers.

6.3.5 Development Control and Intellectual Property

An interesting aspect of such an integrated platform is that the system integrator (e.g. aircraft manufacturer, tier-one supplier), who is responsible for the platform communication system, has development control over all subsystems. Based on the interface specification (communication schedule), every supplier is able to develop its own subsystem. Therefore, the system integrator has full control but is also able to integrate each subsystem easily based on the well-defined interface specification.

Another important fact in an architectural approach is that the module interfaces always provide a gate for information. This means that the system integrator does not have to provide the supplier with all relevant information about his or other subsystems because the supplier is able to develop his own aircraft function based on the interface specification.

6.3.6 Reusability and Composability

Reusability and composability are major design drivers for an integrated platform. Based on the modular structure of the architecture, modules can easily be changed, reused and integrated. To achieve this, the operating system, which handles the interaction of modules inside the host, and the communication system, which handles the interaction of modules inside the network, have to provide several services which support this approach.

6.3.7 COTS

Commercial-off-the-shelf (COTS) hardware and software are a highly discussed topic in the aerospace domain. The advantages are a reduction of development and certification effort, especially if the system has already been used several times.

But there is also resistance which argues that a system has to be fully certified to comply to airworthiness requirements. Currently, the DO-178C working group SC205/WG-71 discusses this topic and the result will be very interesting for aircraft function developers.

6.3.8 Certification

If the certification evidence for a modular platform is accepted once, the effort and costs for changes are low compared to changes in non-modular developed systems. Of course the effort for this first certification is very high, but the certification evidence can be used with minor changes several times.

Assuming that certification has already been done on platform level including all levels below the application, the certification effort for new aircraft functions is reduced to the acceptance of the application and its integration into the platform.

6.3.9 Conclusion

A distributed and integrated platform solution has to provide a set of attributes. This section describes these attributes and discusses their requirements and opportunities.

The key aspect is modular construction which supports independent development and certification. Furthermore, the interface between these modules can be used to save and control intellectual property and support reusability and composability. Another important argument is the use of COTS components because this reduces costs and certification efforts.

6.4 Benefits of an Integrated Platform Approach

The main benefits of the integrated architecture listed below lead to economic benefits, higher quality and better serviceability which are main criteria in the aerospace domain.

6.4.1 Reduction of Complexity

The integrated architecture allows designers to develop subsystems as in a federated system. Based on the fact that every computing resource in the system may be used to host replicated applications, fewer computer systems are needed.

Another point is the integrated diagnosis support which facilitates the understanding of the systems behavior in the presence of faults. Furthermore, the integrated architecture provides rules for structuring the overall application functionality into a set of subsystems.

6.4.2 Reduction of Space, Weight and Power Consumption

Space and weight are major design drivers in the aerospace domain. According to a distributed and integrated platform, common computing units are able to host several different aircraft functions which reduces the units and therefore space, weight and power consumption [35]. Furthermore, interface units can be placed directly at their physical in- and outputs which decreases needed cabling.

6.4.3 Independent Development

The independent development [90] of different aircraft functions allows parallelizing this work. Furthermore, different vendors can develop their functions independently based on the open interface specification of the integrated architecture, which also provides seamless system integration in combination with the encapsulation services of the architecture.

Another aspect is the reuse of components which is also exploited by the modular development of the components and the integration philosophy of the architecture. This speeds up the development process and reduces the time-to-market.

In addition, aircraft function development is reduced to application development because system requirements like replication handling, communication and interaction with the environment are handled by the platform and its services. The aircraft function is just an application which is hosted by the platform.

6.4.4 Simplified Certification

Certification for ultra-dependable systems is a significant cost factor. The integrated architecture offers modular certification by separating the certification of architectural services from applications and by supporting independent safety arguments for different subsystems. Combined with the reuse of components, this strategy considerably reduces the effort for certification and verification activities for the second time the platform is used because only new applications and their integration have to be reviewed.

6.4.5 Increased Flexibility

The integrated architecture provides flexibility in terms of the development process and product customization. In the development process it is possible to experimentally evaluate different configurations and communication topologies without changing the physical structure of the system. Furthermore, it provides mass customization to allow the modification of a subsystem without the need of readapting other subsystems.

6.4.6 Increased Maintainability

Maintenance costs are of major importance for aircraft. Therefore, the computer system has to support this process. Based on such a modular platform, single modules can easily be replaced or updated. Furthermore it is possible to update whole functions or even subsystems faster.

In addition, based on advanced diagnosis, faults can be detected and recovered much faster which reduces the ground times, too.

6.4.7 Conclusion

The switch of a system architecture creates a lot of costs because it has to be developed from scratch. Therefore, there have to be benefits which justify this decision. The key aspects are space, weight and power consumption but also complexity, flexibility and maintainability a from major importance. This section discusses these benefits and shows that a change of the system architecture in the aerospace domain has to be done.

6.5 Conclusion

A distributed and integrated platform solution is the future core system for aircraft electronics. Based on such an open platform, every aircraft function is hosted by standard hardware and middleware software and the function development is reduced to the interface specification and the application development. This reduces the market entry barriers for new function developers and therefore ensures a higher degree of competition which leads to cheaper and better products.

A key advantage of this approach is that every function is connected with each other and information is shared between them. This decreases the number of needed computational resources and in- and outputs, and enhances interaction between different aircraft functions and allows global diagnosis.

In terms of commercial advantages, key aspects for aerospace systems like space, weight and power consumption can be reduced by increased maintainability. In addition, the platform provides technological advantages like reduced complexity, modular design, increased flexibility and simplified development and certification. As a result, such an approach ensures safety and efficiency by reducing development, production and operational costs.

Chapter 7

Prospectus

In the future, airplanes will be flown by a dog and a pilot. And the dog's job will be to make sure that if the pilot tries to touch any of the buttons, the dog bites him.

(Scott Adams)

7.1 Introduction

This section wants to give an overview about current and upcoming trends regarding integrated platforms and their underlying architecture and discuss their effects in different domains. As already considered, the change in aerospace system design is already noticeable but in future systems, it will gain even more influence with respect to technical but also commercial considerations.

7.2 Architectural Evolution

Aircraft architectures will evolve continuously. In my opinion, one of the most promising architectural approaches is the "Web Architecture". This concept is a special kind of a DIMA architecture which consists of several layered, redundant webs. The biggest difference from a common DIMA architecture is that the communication services are initiated and monitored by intelligent, centralized switches.

This star topology allows fault-tolerance and advanced communication services without the need to have advanced services supported by every communication controller. All of these services are handled by the centralized switches. A schematic overview about a web architecture can be found in figure 7.1 on the next page.

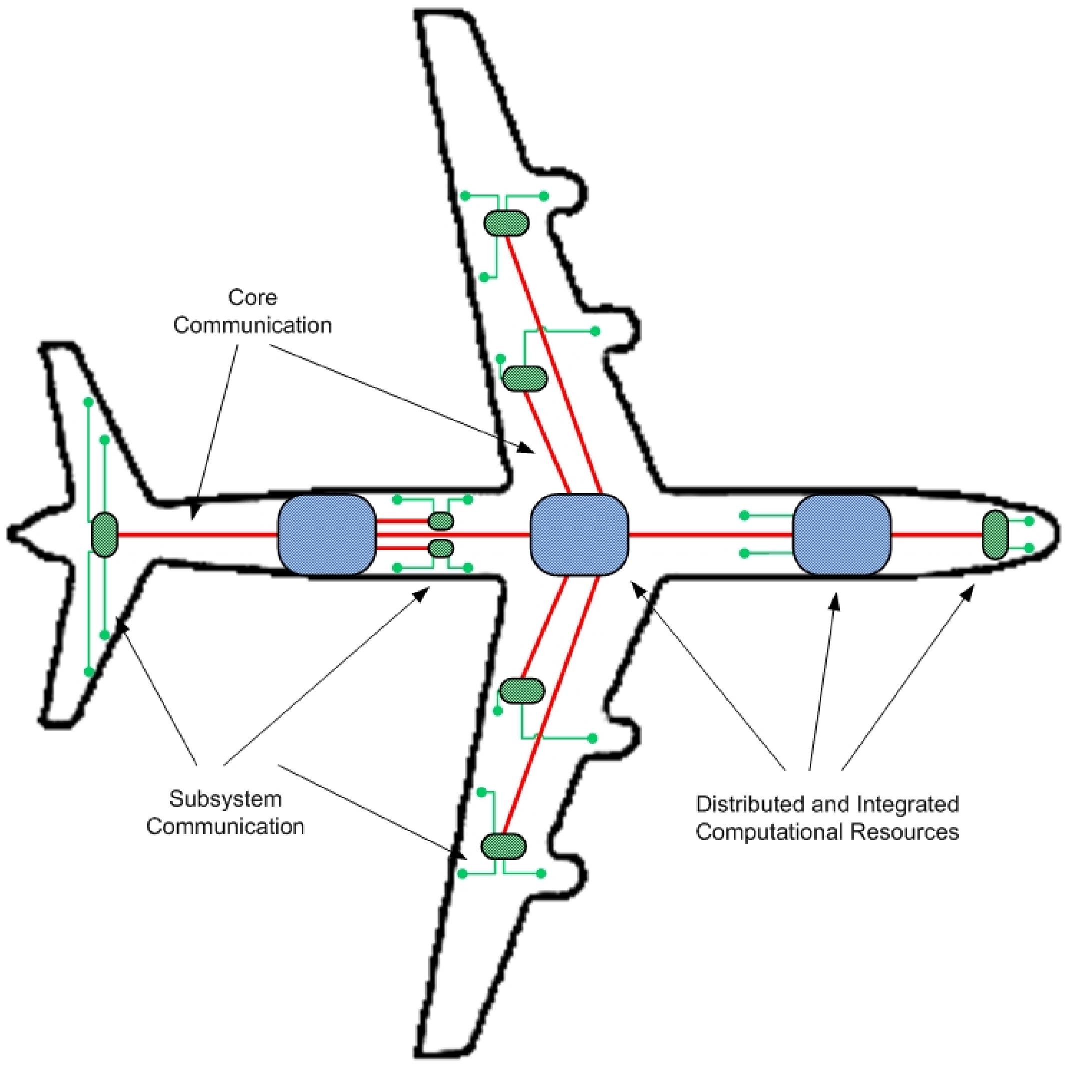

Figure 7.1: Schematic Illustration of Web Architecture Design

This approach also provides the possibility of a hierarchical setup by using a high-performance system bus, the core communication system, which connects gateway modules. Such gateway modules connect the core communication system and a low-cost communication system, where this gateway module act as centralized switch. This allows to create a hierarchical platform which integrates modules with different levels of criticality which are able to communicate with each other and share application and diagnosis data.

A major advantage is that the interface specification of the high-performance core web is in control of the OEM or integrator. The subsystem communication can also be defined by the OEM to reduce the number of different communication systems in the aircraft and therefore minimize maintenance costs in terms of spares, test equipment, production costs and complexity.

Furthermore this architecture provides all DIMA functionalities and increases flexibility, modularity, upgradeability, maintainability, advanced and centralized diagnosis, easy integration and composability. This is because based on the hierarchical approach not only single modules but also groups of modules, which have their own, independent communication, may be replicated, reused, changed, rearranged, upgraded and provide mutual diagnosis.

7.3 Aerospace Domain

7.3.1 Orion - CEV

The decision of the system architecture in the NASA Orion [119] spacecraft might initiate a change for aircrafts, too. Therefore the used system architecture like IMA or DIMA but also the used communication system can effect future trends in the aerospace domain.

The decision for the high-speed communication system has already been taken by using a special development of TT-Ethernet called TT-GBE. This approach allows to send time-triggered and event-triggered data on the same medium. Furthermore it supports a DIMA architecture which demands a high-performance and fault-tolerant communication network.

The choice for the low-cost subsystem network has not been taken yet but a time-triggered approach, like TTP, is still in consideration for this class of communication system.

The final choice of the communication system provides an outlook of the used architecture and can be the first step into a new and forward-looking direction.

7.3.2 Commercial Aircraft

In the Airbus A380 [13] and the Boeing 787 [29] aircraft, there are still several different communication systems in use. But the possibility of cutting costs will also lead to a reduction of the number of different systems. Therefore, a common platform for all aircraft functions connected by a high-performance communication system will be the result.

The switch to such a platform will probably evolve in upcoming projects like the Airbus A350 or the update of the Boeing 737 aircraft. But according to my experience, the final step to a common platform will need some more time for commercial aircraft over 100 seats.

The competition in new developments of regional aircraft (100 seats class) between Airbus, Boeing, Bombardier [60], Embraer [38], Mitsubishi, [77], Suchoi [109] and the China Aviation Industry Corporation I [59] might lead to faster development cycles and the need for advanced technology to provide a reduction of costs, weight by increased efficiency, maintainability, safety and comfort. Therefore, the use of an advanced computer platform which supports these demands will increase. This competition will also rise in the domain of bigger aircraft if the Chinese and Russian companies start their plans to develop airplanes in this class, too.

7.3.3 Small Aircraft

The demand for integrated platforms in smaller aircraft is currently not significant. But the market for small aircraft, especially business jets, is currently increasing and will rise even more in the future. Therefore, it is possible that the competition between different manufacturers will increase like in the regional market segment and requirements for higher safety, more comfort and increased efficiency needs to be addressed. In this case, there will be a market for advanced computer systems for small aircraft, too.

In my opinion this switch will be forced by suppliers who want to use their modular aircraft applications without major changes in several types [85] of aircraft and therefore force the use of integrated platforms. But the change will start at commercial and proceed to small aircraft.

7.4 Automotive Domain

The switch to integrated approaches can not only be found in the aerospace domain. Key phrases like X-by-wire and communication architecture are also discussed in the automotive area at the moment. According to their demands, several automotive companies joined to a consortium called Automotive Open System Architecture (AUTOSAR) to develop an architectural approach.

7.4.1 AUTOSAR

The AUTOSAR initiative [45] is a development partnership of leading automotive manufacturers and suppliers. The goal of the initiative is to develop and standardize an open software architecture for automotive electronic control units (ECUs).

This approach is based on a layered software architecture which should be defined and specified. The layers consist of:

- the hardware abstraction layer (HAL)
- the basic software layer
- the runtime environment
- the application layer

According to these layers, the runtime environment and the basic software layer provide a clearly defined and standardized infrastructure for the development of application software. Based on this, the application can be developed fully independently from the used hardware.

Furthermore, the use of a standardized software architecture allows to reduce development costs and time significantly. The major objectives of the AUTOSAR approach are the possibility of easy integration and transferability of functions, flexible maintenance, scalable functionality, a high standard of system reliability and the already mentioned independence of software and hardware.

7.4.2 Communication System

The standard communication system in the automotive domain is FlexRay which replaces CAN. According to the latest specification, FlexRay [34] does not support safety-critical communication on communication level. This means that it is currently not suitable for safety related functions.

This might lead to problems for the development of a safety-critical architecture on top of FlexRay because all safety-services either have to be solved in the software layers above the communication system, which reduces the efficiency of the host CPU, or have to use the Layered FlexRay approach, described in section 3.5 on page 29. Nevertheless, FlexRay is the standard and will be used to provide the communication services for automotive architectures.

Apart from this disadvantage regarding safety, FlexRay will be used in every future car and therefore a lot of microcontrollers already support it innately. According to this availability and the low costs, FlexRay is also an interesting approach for aerospace systems.

7.5 Economical Effects

The change to integrated platforms provides several economic effects which will change current development processes.

7.5.1 Reusability

Based on composability and a modular system approach, including hardware and software, the reuse of several modules, applications, or whole subsystems is supported even for different domains. For example, flight control systems can be used for commercial but also for regional and small aircraft. Maybe the functionality will be limited to fit the demands of smaller aircraft but the development and certification effort will be minimized based on modularity, composability and reusability.

7.5.2 Faster Development

The factors named in the last section and the platform approach, where a developer only needs to care about the application, allow faster development cycles. If most of the modules can be reused, the development time is decreased even more. Furthermore, if the core system stays the same, different product families can be developed faster, too.

7.5.3 Faster Product Changes

Based on faster development and a reduced time-to-market, modularity and increased competition in the aerospace domain, products will also change faster. Current systems are developed for a product lifetime of at least 20 years. For an integrated platform it is possible to update or change subsystems easier and faster. Therefore, products like flight hardware or software can be changed at a major maintenance procedure.

7.5.4 Cheaper Development and Production

Based on faster and more efficient development, a reduction of development cost is possible. Furthermore, the certification effort is reduced which decreases the development costs, too. Using a platform solution reduces cabling which is also a cost factor for production.

7.6 Future Trends

7.6.1 Development and Integration

Future platforms will be fully designed to support easy development and integration [72]. Advanced development methods, including model-driven ones, allow the developer to focus on aircraft function development without the necessity to consider the underlying platform. Furthermore, the integration of these functions is fully handled by the platform whereby composability problems cease to exist.

7.6.2 Diagnosis and Maintenance

According to advanced and integrated services [72] of future platforms, diagnosis and maintenance are improved but simplified. The whole state of a system, including every minor flaw, will be globally observable and therefore input for recovery or maintenance. According to this, the availability of the overall system is increased and maintenance down-time is minimized.

7.6.3 Weight and Costs

As already mentioned, the use of integrated computer systems allows a reduction of weight, power consumption and, therefore, costs. The economic effects like a decrease in fuel consumption and, therefore, of air pollution is possible.

7.7 Conclusion

This sections describes the future effects of using computer platforms based on an integrated architecture. Increased efficiency and the fact that the competition becomes even higher, aerospace systems and therefore aircrafts will become cheaper in development and production.

Further on, these effects will lead to more efficient aircrafts which need less fuel, provide less air pollution and a provide at least the same degree of safety by reduced costs. This ensures cheaper flights and will make the world an even more globalized one.

I may not have gone where I intended to go, but I think I have ended up where I needed to be.

(Douglas Adams)

Chapter 8

Conclusion

This thesis discusses several parts that are necessary for the implementation of an digital avionics platform. First of all, the architecture needs to be selected because it is the baseline for all upcoming considerations.

Integrated architectures already provide advantages in terms of reduced hardware resources and cabling and therefore reduced size, space, weight, power consumption and design constraints. But according to the approach of Distributed Integrated Modular Avionics (DIMA), the advantages of the federated and the integrated architectures can be combined which push these advantages even more and support others like the placement of physical interface at their natural best position and the avoidance of common cause errors. Therefore, it is concluded that such a DIMA architecture is the best solution for an avionics platform.

The next point that is discussed are communication systems which connect different nodes in such a distributed platform. Two classes emerge during consideration, the core and the subsystem communication system. Based on this classification, several different systems are evaluated and their attributes and suitability are compared. The result indicates that the choice of the communication system depends on the requirements, in terms of safety, flexibility and performance, of the hosted applications.

After choosing the communication system, several major parts of the architecture and its nodes need to be specified like the provided services, the used hardware, operating system and tool support, if needed. Several possibilities, including different operating systems and their interaction are discussed. Furthermore, examples of architectures are presented which can partly be suitable as a platform environment.

To close the prerequisites for a distributed platform solution, certification considerations are presented which allow to reduce the needed development and certification efforts. The concepts of Modular Certification and optimized processes discuss ways to increase efficiency for the certification of such a platform solution. This is one of the major pasts of this thesis which extends the current state of the art. Research provides information regarding process optimization and develops key criteria to achieve this. This research is evaluated based on a comparison to successfully accomplished projects and their actual output is described.

After every part of such a platform has been considered, implementation requirements and recommendations are provided, including major concerns like the change of the architecture and the reuse of already developed aircraft functions. This is the second major part which provide major contributions. These parameters provide considerations which are necessary for a successful implementation of such an approach. Furthermore, platform attributes, based on concerns raised before, are defined and benefits of such an approach are discussed.

Prospected further evolution and commercial challenges are also discussed as upcoming applications and future trends in aerospace development. Further research in this area will have to deal with new architectural approaches, advanced modularity and flexibility but also standardization to provide suitable solutions for different classes of application or even different classes of aircrafts. Regarding certification, the practical use of Modular Certification needs to be verified, and further efforts are needed to update and optimize this approach for future needs.

According to the results developed in this thesis, I am of the opinion that such a Distributed Integrated Platform Solution (DIPS) will be the digital core system in one of the upcoming aircrafts of which development planning is currently starting. Based on the major advantages which result in a reduction of development, production and operational costs, this approach will change aircraft function development.

Acknowledgment: I want to thank everybody who contributed to the surveys regarding "Efficient Software Certification" and "Integrated Architectures" with time, effort and knowledge, especially the DO-178C and the DO-297 working group, and TTTech, where most of the input came from.

Appendix A

Acronyms and Abbreviations

AC	Advisory Circular
AFDX	Avionics Full Duplex Switched Ethernet
APEX	Applications/Executive
API	Advanced Programming Interface
ARINC	Aeronautical Radio Incorporated
ASL	Advanced Services Layer
AUTOSAR	Automotive Open System Architecture
BAG	Bandwidth Allocation Gap
BIU	Bus Interface Unit
CNI	Communication Network Interface
COTS	Commercial-Off-The-Shelf
CPU	Central Processing Unit
CSMA	Carrier Sense Multiple Access
DAS	Distributed Application Subsystem
DER	Designated Engineering Representative
DIMA	Distributed Integrated Modular Avionics
DIPS	Distributed Integrated Platform Solution
EASA	European Aviation Safety Agency
EUROCAE	European Organization for Civil Aviation Equipment
ECR	Error Containment Region
ECU	Electronic Control Unit
EEE	Encapsulated Execution Environment
eLoc	Effective Lines of Code
FAR	Federal Airworthiness Requirement
FCR	Fault Containment Region
FTA	Fault-Tolerant Average

HAL	Hardware Abstraction Layer
IMA	Integrated Modular Avionics
L-FlexRay	Layered FlexRay
L-TTP	Layered Time Triggered Protocol
MEA	More Electric Architecture
MEDL	Message Descriptor List
NASA	National Aeronautics and Space Administration
NGU	Never-Give-Up
OEM	Original Equipment Manufacturer
OS	Operating System
PE	Processing Element
PIL	Platform Interface Layer
RMU	Redundancy Management Unit
ROBUS	Reliable Optical Bus
RTCA	Radio Technical Commission for Aeronautics
SAE	Society of Automotive Engineers
SCL	Synchronization and Communication Layer
SPIDER	Scalable Processor-Independent Design for Extended Reliability
TDMA	Time-Division Multiple Access
TT-Ethernet	Time Triggered Ethernet
TT-Gbe	Time Triggered Giga-Bit Ethernet
TTP	Time Triggered Protocol
TTPos	Time Triggered Protocol Operating System

Appendix B

Publications

High Speed and High Dependability Communication for Automotive Electronics
B. Rumpler, C. Weich and R. Wolfig
SAE World Congress, 2006

Time-Triggered Architecture based on FlexRay: Roadmap from High-Speed Data Networking to Safety Relevant Automotive Applications
M. Buhlmann, S. Poledna, G. Stoeger and R. Wolfig
Convergence, 2006

Layered FlexRay - An Environment for Distributed Safety-Critical Applications
R. Wolfig
Embedded World Conference, 2007

Subsystem Design Using Time-Triggered Protocol (TTP): Key Aspects of Control System Application Reuse
R. Wolfig
SAE Aerotech, 2007

Parameters for Efficient Software Certification
R. Wolfig
EUROMICRO, ERCIM / DECOS Workshop on Dependable Embedded Systems, 2007

Bibliography

[1] C. Adams. Reusable software components - will they save time and money? April 2005. Avionics Magazine.

[2] FAA Federal Aviation Administration. http://www.faa.gov.

[3] Federal Aviation Administration. Far25: Airworthiness standards: Transport category airplanes,. Technical report, 1965 - 2004.

[4] Federal Aviation Administration. Far23: Airworthiness standards: Normal, utility, acrobatic, and commuter category airplanes. Technical report, 1965 - 2006.

[5] Federal Aviation Administration. Dot/faa/ar-01/26: Commercial off-the-shelf (cots) avionics software study. Technical report, 2001.

[6] Federal Aviation Administration. Study of commercial off-the-shelf (cots) real-timeoperating systems (rtos) inaviation applications. Technical report, 2002.

[7] Federal Aviation Administration. Tso-c153, integrated modular avionics hardware elements. Technical report, FAA, 2002.

[8] Federal Aviation Administration. AC-145: Guidance for Integrated Modular Avionics (IMA) that implement TSO-C153 authorized Hardware Elements. Technical report, FAA, 2003.

[9] Federal Aviation Administration. Ac20-148: Reusable software components. Technical report, FAA, 2004.

[10] NASA National Aeronautics and Space Administration. http://www.nasa.gov.

[11] TTTech Computertechnik AG. www.tttech.com.

[12] EASA European Aviation Safety Agency. http://www.easa.eu.int.

[13] Airbus. http://www.airbus.com.

[14] C. M. Ananda. Science of civil aircraft advanced avionics architectures - an insight into saras avionics challenges, a present and future perspective. Technical report, 2007. Symposium on Aircraft Design.

[15] ARINC. *ARINC 659 - Backplan Data Bus.* Aeronautical Radio, Inc., 1993.

[16] ARINC. *ARINC 653 - Avionics Application Software Standard Interface.* Aeronautical Radio, Inc., 1997.

[17] ARINC. *ARINC 629 - Data Bus Specification.* Aeronautical Radio, Inc., 1999.

[18] ARINC. *ARINC 429 - Data Bus Specification.* Aeronautical Radio, Inc., 2001.

[19] ARINC/AEEC. *ARINC 664 - Aircraft Data Networks.* Aeronautical Radio, Inc., 2003.

[20] C. Weich B. Rumpler and R. Wolfig. High speed and high dependability communication for automotive electronics. In *SAE Transactions Journal of Passenger Cars: Electronic and Electrical Systems, March 2007*, 2006. SAE World Congress.

[21] title = BAE Systems.

[22] A. Bahrami. Complex integrated avionic systems and system safety. 2005. Europe/U.S. International Aviation Safety Conference.

[23] Günther Bauer and Michael Paulitsch. An investigation of membership and clique avoidance in TTP/C. In *19th IEEE Symposium on Reliable Distributed Systems, 16th - 18th October 2000, Nürnberg, Germany*, pages 118–124, 2000.

[24] Technische Universität Berlin. www.tu-berlin.de.

[25] B. Boehm. *Software Engineering Economics.* Prentice Hall, 1981.

[26] Certification Authorities Software Team (CAST). Position paper cast-19 - clarification of structural coverage analyses of data coupling and control coupling. Technical report, 2004.

[27] R. Walter C.B. Watkins. Transitioning from federated avionics architectures to integrated modular avionics. 2007. Digital Avionics Systems Conference.

[28] SAE Airplane Safety Assessment Committee. Arp 4761 - guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment. Technical report, 1996.

[29] The Boeing Company. http://www.boeing.com.

[30] P. Conmy and J. McDermid. High level failure analysis for integrated modular avionics. Technical report, 2001. Australian Workshop on Safety Critical Systems and Software.

[31] FlexRay Consortium. http://www.flexray.com/.

[32] FlexRay Consortium. *FlexRay Communications System Electrical Physical Layer Application Notes*. Version 2.1, FlexRay Consortium, 2006.

[33] FlexRay Consortium. *FlexRay Communications System Electrical Physical Layer Specification*. Version 2.1, FlexRay Consortium, 2006.

[34] FlexRay Consortium. *Specification of the FlexRay Protocol*. Version 2.1, FlexRay Consortium, 2006.

[35] T. Cornilleau. Dassault aviation feedbacks on its military and civil ima applications. 2007. ARTIST2 meeting on Integrated Modular Avionics, Rome.

[36] Allied Standard Avionics Architecture Council. Asaac - an overview, issue: 01. Technical report, 2000.

[37] R. Chillarege D. Siewiorek and Z. Kalbarczyk. Reflections on industry trends and experimental research in dependability. Technical report, April-June 2004. IEEE Transactions on dependable and secure computing, Vol. 1, N0. 2.

[38] Empresa Brasileira de Aeronáutica. http://www.embraer.com.

[39] Decos. Requirements specification - encapsulated execution environment. Technical report, 2004.

[40] Decos. Design specification - encapsulated execution environment. Technical report, 2005.

[41] J. C. Knight E. A. Strunk and M. A. Aiello. October 2004. Digital Avionics Systems Conference, Salt Lake City.

[42] Wilfried Elmenreich, Günther Bauer, and Hermann Kopetz. The time-triggered paradigm.

[43] D. Emery. July 2001. Real-time and Embedded Systems Forum.

[44] Condor Engineering. Afdx - protocol tutorial. Technical report, 2005.

[45] H. Heinecke et al. Automotive open system architecture - an industry-wide initiative to manage the complexity of emerging automotive e/e-architectures. 2004. Convergence.

[46] H. Forsberg and K. Karlsson. 2006. 25th Digital Avionics Systems Conference.

[47] K. Frazer. Return on software process improvement. Technical report, 1997.

[48] M. Gasiorowski. Differences and similarities in do-178b compliance in an ima system vs a federated product. 2005. FAA National Software Conference.

[49] P. Groessinger. Software certification for a time-triggered operating system. Technical report, 2005.

[50] SAE Systems Integration Requirements Task Group. Arp 4754 - certification considerations for highly-integrated or complex aircraft systems. Technical report, 1996.

[51] Kopetz H., Obermaisser R., Peti P., and Suri N. From a federated to an integrated architecture for dependable real-time embedded systems. 2004.

[52] P. Grillinger H. Kopetz, A. Ademaj and K. Steinhammer. The time-triggered ethernet. 2005. Real-time and Embedded Systems Workshop, Arlington.

[53] P. Grillinger H. Kopetz, A. Ademaj and K. Steinhammer. The time-triggered ethernet (tte) design. 2005. 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05).

[54] Robert Hammett. Flight-critical distributed systems: Design considerations. *IEEE AESS Systems Magazine, June 2003*, 2003.

[55] Inc. HighRely. Do-178b costs versus benefits. Technical report, HighRely, Inc., 2007.

[56] V. Hilderman. Certifying an rtos to do178b: Tips & tales. October 2001.

[57] Honeywell. The more electric architecture revolution. 2006.

[58] Kenneth Hoyme and Kevin Driscoll. Safebus. pages 68–73, 1992. In 11th AIAA/IEEE Digital Avionics Systems Conference.

[59] China Aviation Industry Corporation I. http://www.avic1.com.cn/english/englishindex.asp.

[60] Bombardier Inc. http://www.bombardier.com.

[61] J.B. Itier. A380 integrated modular avionics - the history, objectives and challenges of the deployment of ima on a380. 2007. ARTIST2 meeting on Integrated Modular Avionics, Rome.

[62] B. Argrow J. Elston and E. Frew. A distributed avionics package for small uavs. Technical report, 2005. American Institute of Aeronautics and Astronautics.

[63] R. Viswanathan J. Litlefield-Lawwill. Advancing open standards in integrated modular avionics: An industry analysis. 2007. Digital Avionics Systems Conference.

[64] Martin Peller Josef Berwanger and Robert Griessbach. byteflight - a new high-performance data bus system for safety-related applications. Technical report, BMW AG, 2000.

[65] A. Ademaj K. Steinhammer, P. Grillinger and H. Kopetz. A time-triggered ethernet (tte) switch. 2006. Conference on Design, Automation and Test in Europe.

[66] F. Wolf K. Tindell, H. Kopetz and R. Ernst. Safe automotive software development. 2003. Design, Automation and Test in Europe.

[67] Aaron D. Kahn. The design and development of a modular avionics system. Master's thesis, Georgia Institute of Technology, School of Aerospace Engineering, Atlanta, GA 30332, apr 2001.

[68] V. Khanna and M. DeWalt. Reusable software component (rsc) reality show. 2005. Software/CEH Conference: Norfolk.

[69] H. Kopetz. Sparse time versus dense time in distributed real-time systems. 1992. In Proc. of ICDCS.

[70] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston, 1997.

[71] H. Kopetz. On the fault hypothesis for a safety-critical real-time system. Technical report, 2003.

[72] H. Kopetz. Dependable computing in 2031 - back to the start?, June 2006.

[73] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. pages 933–940, 1987. IEEE Transactions on Computers.

[74] Hermann Kopetz and Günther Bauer. The time-triggered architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[75] Hermann Kopetz, Martin Braun, Christian Ebner, Andreas Krüger, Dietmar Millinger, Roman Nossal, and Anton Schedl. The design of large real-time systems: The time-triggered approach. *Proceedings of the 16th Real-Time Systems Symposium, December 1995, Pisa, Italy*, Dec. 1995.

[76] Lamport, Shostak, and Pease. The byzantine generals problem. In *Advances in Ultra-Dependable Distributed Systems, N. Suri, C. J. Walter, and M. M. Hugue (Eds.), IEEE Computer Society Press.* 1995.

[77] Mitsubishi Heavy Industries Ltd. http://www.mrj-japan.com/.

[78] LynuxWorks. Lynxos-178 rtos. http://www.lynuxworks.com/rtos/rtos-178.php.

[79] Gruber M. Decos project proposal. 2004.

[80] G. Stoeger M. Buhlmann, S. Poledna and R. Wolfig. Time-triggered architecture based on flexray: Roadmap from high-speed data networking to safety relevant automotive applications. 2006. Convergence.

[81] NASA Langley Formal Methods. http://shemesh.larc.nasa.gov/fm/.

[82] R. Obermaisser. *Event-Triggered and Time-Triggered Control Paradigms - An Integrated Architecture*. Kluwer Academic Publishers, Real-Time Systems Series, 2004.

[83] R. Obermaisser. Supporting heterogeneous applications in the decos integrated architecture. 2007. ARTIST2 meeting on Integrated Modular Avionics, Rome.

[84] Vienna University of Technology. www.tuwien.ac.at.

[85] Norm Ovens. An industry perspective of integrated modular avionics. 2005. Software and Complex Electronic Hardware Standardization Conference.

[86] P. Parkinson. Safety-critical software development for integrated modular avionics. 2003.

[87] Holger Pfeifer. Formal verification of the TTP group membership algorithm. In *FORTE*, pages 3–18, 2000.

[88] S. Poledna. *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Kluwer Academic Publishers, Boston, 1996.

[89] S. Poledna. Ttp-tools - the tool set of the time-triggered architecture. 2004. The Monterey Workshop Series.

[90] S. Poledna. The future of aerospace data communication. September 2005. Aerospace Engineering.

[91] P. Peti R. Obermaisser and H. Kopetz. Virtual gateways in the decos integrated architecture. 2005. Workshop on Parallel and Distributed Real-Time Systems.

[92] P. Peti R. Obermaisser and H. Kopetz. Virtual networks in an integrated time-triggered architecture. 2005. WORDS 2005.

[93] E. Retko. Integrated modular avionics (ima) trends and challenges. 2005. Software and Complex Electronic Hardware Standardization Conference.

[94] Leanna Rierson. Software reuse in safety-critical systems. Master's thesis, Rochester Institute of Technology, 2000.

[95] G. Romanski. September 2001. CrossTalk.

[96] RTCA. *DO-178B - Software Considerations in Airborne Systems and Equipment Certification.* Radio Technical Commission for Aeronautics, Inc., 1992.

[97] RTCA. *DO-254 - Design Assurance Guidance for Airborne Electronic Hardware.* Radio Technical Commission for Aeronautics, Inc., 2000.

[98] RTCA. *DO-278 - Guidelines for Communications, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance.* Radio Technical Commission for Aeronautics, Inc., 2002.

[99] RTCA. *DO-297 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations.* Radio Technical Commission for Aeronautics, Inc., 2005.

[100] B. Rumpler. Complexity management for composable real-time systems. Technical report, 2006. Object and Component-Oriented Real-Time Distributed Computing, ISORC.

[101] John Rushby. Partitioning for safety and security: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA.

[102] John Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2001.

[103] John Rushby. Modular certification. Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA, September 2001.

[104] John Rushby. Modular certification. Menlo Park, CA, September 2001. Dagstuhl seminar on Dependability of Component Based Systems.

[105] C. Salmon and C. Lee. The certification of systems containing software developed using rtca do-178b. Technical report, 2006. ERA Report 2006-0036 Issue 3.

[106] Cary R. Spitzer. *Digital Avionics Handbook, Second Edition.* CRC Press, 2006.

[107] Magnus Öst. Simulation of safety critical networks. Master's thesis, Royal Institute of Technology Stockholm, 2001.

[108] Victoria Stavridou and R. A. Riemenschneider. Provably dependable software architectures. pages 133–136.

[109] Suchoi. http://www.sukhoi.org/.

[110] D. L. Swanson. Evolving avionics systems from federated to distributed architectures. 1998. Digital Avionics Systems Conference.

[111] Derivation Systems. http://www.derivation.com/.

[112] Wind River Systems. Vxworks 653 platform. http://www.windriver.com/products/platforms/safety_critical/index.html.

[113] Janos Sztipanovits and Gabor Karsai. Embedded software: Challenges and opportunities. In *EMSOFT*, pages 403–415, 2001.

[114] Saab Technology. Distributed integrated modular avionics - evolution to the future.

[115] TTTech. Protocols for aerospace control systems - a comparison of afdx, arinc 429, can, and ttp. 2004.

[116] TTTech. *Specification of the TTP/C Protocol*. Version 1.1, TTTech Computertechnik AG, 2004.

[117] TTTech. The time-triggered technology and the two-level design approach, 2004.

[118] G.M. Uchenick. Partitioning communications system for safe and secure distributed systems. 2007. Digital Avionics Systems Conference.

[119] NASA Constellation Program: Orion Crew Vehicle. http://www.nasa.gov/mission_pages/constellation/orion/index.html.

[120] Ben L. Di Vito. A model of cooperative noninterference for integrated modular avionics. In Charles B. Weinstock and John Rushby, editors, *Dependable Computing for Critical Applications—7*, volume 12, pages 269–286, San Jose, CA, 1999. IEEE Computer Society.

[121] Richard Warrilow. The avionics platform. 2004.

[122] A. Wilson. The evolving arinc 653 standard and itŠs application to ima. 2007. ARTIST2 meeting on Integrated Modular Avionics, Rome.

[123] J. Wlad. Software reuse in safety-critical airborne systems. Technical report, 2006.

[124] R. Wolfig. Layered flexray - an environment for distributed safety-critical applications. 2007. Embedded World Conference.

[125] R. Wolfig. Parameters for efficient software certification. 2007. EUROMICRO, ERCIM / DECOS Workshop on Dependable Embedded Systems.

[126] R. Wolfig. Subsystem design using time-triggered protocol (ttp): Key aspects of control system application reuse. 2007. SAE Aerotech.