TRANSFER OF ADVANCED ASAAC SW TECHNOLOGY ONTO THE EUROFIGHTER/TYPHOON

Thomas Brixel
European Aeronautic Defence and Space Company
(EADS Deutschland GmbH)

Keywords: *Software Development, ASAAC, Integrated Modular Avionic, EuroFighter*

Abstract

EuroFighter/Typhoon has to deal with obsolescence of avionic hardware and increasing performance requirements.

To improve reusability of software and ease the migration to new platforms a three-layer stack following Allied Standard Avionics Architecture Council standards is introduced.

The steps towards an Integrated Modular Avionic are illustrated.

1 Introduction

The presentation gives a short overview why and how the ASAAC standards are introduced to the EuroFighter program. It describes the challenges in introducing new technology to a program with respect to legacy software.

2 Motivation for ASAAC

The rapid development of computer technology is both a blessing and a curse. On one side it provides the computing power necessary to utilize advanced algorithms. On the other side the lifetime of computer components is much shorter than that of an aircraft, especially a military fighter aircraft. The lifetime of aircrafts is measured in decades whereas the components may become obsolete during development.

2.1 Current Situation

The development of EuroFighter started in the eighties of the last century. Now, with the first series aircrafts having been delivered to customers the necessity for an avionic upgrade is visible due to hardware obsolescence and increased performance requirements. To address the obsolescence issue and provide the processing capabilities for future enhancements a new hardware platform is introduced to the EuroFighter program together with a new software architecture.

The EuroFighter avionics is a federated architecture where proprietary avionic computers, so called Line Replaceable Items (LRIs), are connected via busses. The LRIs have a high functional integration and subsystem functionality is allocated to single LRIs. With this kind of architecture changes and extensions are expensive and time consuming. An enormous effort is spent on integration.

2.2 Route to IMA

The intention is to introduce an Integrated Modular Avionic (IMA) in the EuroFighter eventually. The Allied Standard Avionics Architecture Council (ASAAC) defined a set of open standards, concepts and guidelines for such an Integrated Modular Avionic. This includes hardware standards to allow manufacturers the production of common off the shelf components that fit into an ASAAC system, networking standards to allow interoperability between modules of different manufacturers and software standards to allow reusability of software. The IMA architecture will reduce the cost for development and maintenance of an aircraft.

For risk reduction purposes the legacy application software and avionics architecture remains unchanged in the first step. A set of

avionic computers is replaced by upgraded hardware with identical shape and connectors. A three layer stack (TLS) following ASAAC software standards is implemented on selected computers to prepare the application software to run on a modular avionic in the future. A commercial Real-Time Operating System (RTOS) is introduced. INTEGRITY from Green Hills was selected for the EuroFighter.

The upgraded LRIs consist of one Common EFEX Module (CEM) that provides access to external busses and three Common Processing Modules (CPM) that run the application software. The modules are connected through a VME backplane. In the ASAAC standard the terminology for a module is processing element.

3. The ASAAC Software Stack

In the meanwhile the North Atlantic Treaty Organization (NATO) agreed ASAAC as Standardization Agreement (STANAG) number 4626 [1]. Part II of the standard defines the software architecture.

The software stack has a layered software architecture where each layer provides an additional level of abstraction.

3.1 Three Layer Stack

The Module Support Layer (MSL) encapsulates the details of the underlying hardware and provides generic, technology independent access to low-level resources. Especially point-to-point unidirectional transfer connections are accessible through a network independent interface. Transfer connections were implemented for communication between modules over the VME backplane.

A Real-Time Operating System (RTOS) is the major component of the Operating System Layer (OSL). It controls the real-time behaviour of the processing element and its resources. The functionality offered by the Generic System Management (GSM) is as the name already implies generic and therefore it resides in the OSL. Functions of the GSM are health monitoring, fault management and configuration management.

The Configuration Management (CM) performs the configuration of the system according to information from the Run-Time Blueprints (RTBP) that are stored in files formatted in Extensible Mark-up Language (XML).

The application software resides in the Application Layer (AL).

The layers of the software stack itself are independent from the actual application running on a LRI. The hardware dependent MSL and the hardware independent OSL can be reused in different LRIs. The application layer is also hardware independent, which eases the migration to new hardware platforms.

3.2 Tailoring of the Software Stack

Not the complete functionality of an ASAAC software stack is required.

Health monitoring and fault handling is still performed by the application software. The chosen Real-Time Operating System, Integrity from Green Hills, performs the major tasks of the Operating System Layer.

The work focused on the implementation of communication services required by the application software and their configuration.

The resulting software stack should be reusable for safety critical and any other kind of application. Therefore the use of INTEGRITY functionality was restricted to the certifiable subset available in INTEGRITY DO-178b.

3.3 Virtual Channels and Transfer Connections

Virtual channels allow communication independent from the number and location of the participants. The application uses identifiers that are local to a process to access the virtual channel. A virtual channel may be attached to an arbitrary number of processes for receiving and/or sending. For transmission of messages to other modules the virtual channel is attached to transfer connections.

When a message is sent to a virtual channel it is delivered to each attached receiver.

Services are provided for blocking and non-blocking transmission. A timeout can be specified for blocking transmissions.

4 Implementation of the Software Stack

An international team was formed to develop the software stack following ASAAC standards for the next generation of avionic computers in the EuroFighter. I was a member of that team from the beginning.

The target hardware was not available from the beginning. The target hardware and hardware related software services where developed in parallel to the software stack. Common off the shelf (COTS) hardware boards were used for development of the software stack.

4.1 The Prototype

When development started no commercially available version of an ASAAC software stack existed. The standards were available as drafts and a prototype of the software stack written in C already existed. This prototype was originally written to run on LynxOS but the Real-Time Operating System selected for EuroFighter is INTEGRITY from Green Hills. So the prototype was rewritten to run on INTEGRITY without much considerations of a proper software design.

The decision was to start development using that prototype which was with hindsight not really optimal.

The Prototype had some deficiencies that required an extension or reimplementation of the existing functionality but then it also contained functionality we did not use.

The attachment of virtual channels to transfer connections was performed as a local transmission to an extra process on the same processing element, which forwarded the data. We considered this as too time consuming as each message had to pass some extra layers on the sender as well as on the receiver side. But more important it also violated an important requirement that the communication shall be independent from the location of participants, which could not be realized with this implementation.

The implementation of the MSL used common off the shelf libraries that are not available for our target.

4.2 Adaptation of Prototype to Requirements

The prototype was first adapted to the requirements that were imposed from the legacy application software and hardware platform.

So was the implementation of transfer connections via fibre-channel or ATX and usage of the COTS library for backplane communication removed.

The application software uses a mailbox mechanism for communication between objects that can reside on a single or on different processing elements. It ensures that a message is stored in the receivers' mailbox if no error is indicated. This mailbox mechanism has to be mapped to virtual channels. In the ASAAC standard a send operation is successful when the message was handed over to the Operating System Layer. This does not guarantee that it is delivered to the receivers. To avoid the necessity of a time consuming handshake at application level the behaviour of the communication services was modified to be suitable for replacement of the mailbox mechanism.

In deviation to the ASAAC standard our virtual channel management uses a transmission protocol between instances of the Operating System Layer. As a consequence the data sent via transfer connections differs from the specified format, which makes it incompatible to other implementations of the software stack. After the application software is fully migrated to an Integrated Modular Avionic a standards conformant software stack can be used.

The fact that INTEGRITY DO-178b does not have a file system leads to another deviation from the standard. The Run-Time Blueprints are not stored in XML files. Instead the configuration data is provided as constants in Ada packages. This also eliminates the need of a

XML parser as the Ada compiler checks the syntax. The values adherence to defined ranges can be ensured by use of appropriate Ada types.

4.3 Migration to Ada

Integration tests showed that the software stack developed from the prototype does not satisfy the quality and reliability requirements of avionic software. Therefore we decided to re-implement it in Ada, which is the official programming language for the EuroFighter program. The Ada implementation was much more stable from the beginning and showed that there are good reasons to use Ada in security or mission critical systems.

As the Ada implementation was based on existing C code and not done from scratch it showed many similarities to the C code. The software design of the prototype still remained.

4.4 EFEX Connection

A major challenge was the integration of external busses into the ASAAC software stack. The LRIs use MILBUS (MIL-STD-1553, STANAG 3838), EFABUS (STANAG 3910) as well as the newly introduced EFABUS Express (EFEX). These bus systems follow a completely different approach than the ASAAC virtual channel concept. The ASAAC standard provides message based and streaming communication that rely upon packet switched networks or dedicated connections. As well as its older equivalents, EFEX is a multiplexed bus that transmits messages scheduled according to a predefined transaction table.

In the first run with the unmodified legacy application we decided to implement a remote procedure call mechanism. The application used Target Specific Ada Packages (TSAP) to access MILBUS and EFABUS on the old LRIs. Calls to the Target Specific Ada Packages are now propagated to the Common EFEX Module that is the only processing element that can access the EFEX hardware. The calls are translated to EFEX services, which in some cases require that they are retained and finished when the complete information required by an EFEX service is available. Virtual channels are used for communication. The remote part on the EFEX module receives the service requests on a virtual channel and returns the result on another virtual channel that is assigned to the task that performed the TSAP call.

The CPU of the EFEX module proved to be too weak to handle the amount of communication through all the ASAAC layers plus the conversion of the TSAP services. This performance problem was partly solved on the supplier side by tuning the hardware settings. The other part was solved by a change in the software stack.

In the second run the Remote Procedure Call services on the EFEX module directly send and receive on a transfer connection. This relieved the CPU load on the EFEX module. The virtual channel management is bypassed. The protocol (acknowledgment of messages) is not necessary because synchronisation is performed by the queue of the transfer connections. The downside is that we lost some flexibility. The use of TSAP services is restricted to a single CPU. This is acceptable because the application software that was spread over 6 processors on the old hardware is now combined into a single program.

4.5 Redesign of Virtual Channel Management

Two severe problems were found during integration with the application software.

The configuration managers on each processing element send synchronisation messages after the configuration is finished. It leads to errors when the receiver of these messages has not finished the configuration itself. The ASAAC standard defines the message format for the logical interface between configuration managers and defines that it uses virtual channels but it does not address this problem.

The send operation of the transfer connections blocked the caller when the queue of a transfer is full. As the majority of messages is exchanged between two modules this can lead

to a deadlock situation when the queues in both directions are full.

The software stack was re-implemented several times but at no time there was a cut and a real re-design. The software design of the virtual channel management resulted from the migration of C code to Ada and a gradually growing functionality and complexity.

Therefore, even in a relative late phase of development, we decided to re-design the Virtual Channel Management from scratch. Now it contains only the necessary functionality and uses a strict type concept to avoid type conversions and run-time checks. It also avoids using addresses instead of Ada types wherever possible. This led to a much improved software with respect to maintainability and testability.

5 The Software Stack in Everyday Life

The software stack passed the formal reviews that are mandatory for the EuroFighter development process. It is in use since the middle of the year and proved to work on the development COTS system and in the target hardware on test benches and rigs. Recently it accomplished its first mission in an aircraft even though only on ground. The first flight tests are planned for the forthcoming year.

6 Conclusion

It seems to be true that everybody has to make mistakes by himself. The developed software reached a usable status quite fast, but definitely we would do better with the experience we have now.

The development started as a technology project. Some challenges had to be met to bring it to a mature, stable and reliable state. It is best practice to leave a prototype as a prototype and start development of production software from scratch following the standardised development process.

Currently the advantages of a platform independent application on top of a layered software stack are not visible to everybody. Some see only the disadvantages, mainly the performance overhead of this software architecture.

In my opinion this will change soon. The application software can run virtually anywhere when ASAAC is available on various platforms.

References

[1] *STANAG 4626 – MODULAR AND OPEN AVIONICS ARCHITECTURES, PART II: SOFTWARE*, Draft 1, NORTH ATLANTIC TREATY ORGANIZATION (NATO), 2004.