Model-Based Engineering Approach for System Architecture Exploration

Julien Delange*, Christophe Honvault*, James Windsor*

*: European Space Agency, Keplerlaan 1, 2201AZ Noordwijk, The Netherlands
{firstname.lastname}@esa.int

Abstract: Safety-critical systems may contain a large number of functions having different security/safety levels and must ensure a continuous operational state. It is of prime importance to avoid errors propagation between system functions. One may identify two main solutions to tackle that problem. The first and classical solution relies on the federated architecture where different hardware nodes, each one executing one or several functions having the same security/safety level, are interconnected using communication channels. The second solution emerged recently and leaded to the definition of the integrated architecture where a same hardware node is able to execute several functions having different security/safety levels thanks to dedicated hardware (as Memory Management Unit) and software (as hypervisors).

These two architectures have their own advantages and drawbacks in term of dependability, mass, processing power, consumption, integration and validation efforts, costs, etc. As a consequence, choosing the architecture is difficult and system engineers have to rigorously evaluate the deployment strategy.

This paper presents an approach to automate the integration of an implementation on different architectures. As a result, it provides the ability to deploy the same code on several nodes (federated architecture) or on a partitioned system (integrated architecture). For that purpose, the TASTE tool-chain is extended to support the deployment on XtratuM, a hypervisor that is ported on space qualified processors. By using the new tool-chain, designers can automatically produce federated or partitioned systems and evaluate their efficiency in terms of resources consumption, performances as well as the impact in the development process.

Keywords: IMA, AADL, code generation, XtratuM

1. Context & Background

Avionics or aerospace systems are mission or life critical so that a failure may have catastrophic impact (premature end of mission, loss of life, etc.). As a result, these systems must be designed carefully in order to ensure a correct behaviour when they are operating. However, this has become extremely difficult due to the large number of requirements coming from different system domains (power, thermal, data handling, GNC, etc.) and have to be enforced in the implementation. As a consequence, the number of system functions increases significantly as well as interactions between these functions. It therefore becomes very complicated to evaluate the potential impact of modifications on the complete system. This problem becomes particularly difficult when a single system hosts functions with different levels of safety or security.

To address these issues, two typical solutions are proposed; the federated versus the integrated deployments.

1.1. Federated deployment

The federated deployment consists in separating system functions on different hardware nodes, depending on their security or safety levels. Using this solution, one computer will host functions classified at high safety/security levels and another executes functions that are less critical. By enabling such a separation, functions are isolated one from another, which reduces the potential impact of a failure from one function to another classified at higher levels.

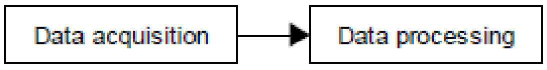


Figure 1- Data acquisition/processing, functional view

Figure 1 illustrates a basic system with two functions: one that acquires data and another that processes them (e.g. this can be a basic system that acquires a sensor raw value and periodically stores the corresponding engineering value). Figure 2 shows the deployment of these functions in a federated architecture: each function is deployed on a separated hardware node and the communication between them uses a physical network.

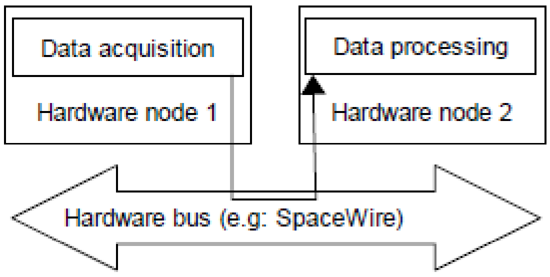


Figure 2- Data acquisition/processing federated deployment

1.2 Integrated deployment

The integrated deployment collocates all functions on the same computing node, no matter their safety/security level. However, each function is isolated from another using a dedicated separation layer. To avoid any potential error propagation of an error from one function to another, this separation layer provides the following features:

1. Space isolation: each function is associated with a single memory segment to store its data/code and communication outside the memory allocation is explicitly declared and granted.

2. Time isolation: each function has a unique time slice to execute its tasks. A function therefore cannot overrun its budget time so cannot impact another.

As a consequence of the second deployment option, the function can be allocated to a partition, which is defined as a dedicated set of computing resources (memory, CPU time, IO) allocated to a specific function A separation kernel is responsible for enforcing the temporal and spatial isolation of the partitions. The approach is referred to as Time and Space Partitioning (TSP).

The integrated deployment is illustrated in Figure 3: each function is bounded to a partition and the communication uses the software bus provided by the separation kernel.

1.3 Separation Kernels

Several partitioning systems solutions are available for the design of embedded safety-critical systems either under commercial conditions (e.g. PikeOS [13] or VxWorks [14]) or open-source license (as POK [6]). While most of existing products comply to the Integrated Modular Avionics (IMA) ARINC653 [7] Application Programming (APEX) layer defined for the avionics domain, a recent ESA study [12] supported the port of the open source XtratuM hypervisor to space qualified processors. XtratuM can isolate several partition, each one executing one or several functions, either directly (bare code) or on top of a compatible operating system as RTEMS [9]. This approach is justified by the fact that RTEMS is a product supported by ESA [15].

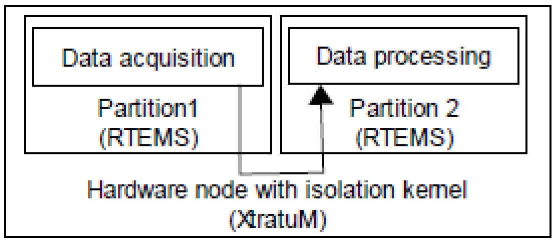


Figure 3- Data acquisition/processing integrated deployment

2. Problem

When designing either an integrated or federated architecture it is the responsibility of the system architect/integrator to assess the impact of the deployment strategy on the overall project. There can be clear benefits in terms of mass, power and volume for reducing the number of on-board computing nodes. This benefit must be traded-off against the increased complexity due to the integrated functions.

This trade-off must be performed early in the system lifecycle in order to initiate the subsystem and equipment procurement. As a consequence the system design is immature and shall need multiple interactions before a coherent concept can be agreed. The trade-off must include the following properties:

- **Independency of functional chains**: can functional chains be identified that are independent or can be isolated such that they can be partitioned. If there are too many dependencies between functions, moving them to a partitioned infrastructure shall introduce delays caused by the inter-partition communication. In addition, failure propagation can be an issue for functionally dependent application. How does a function react if a function it is dependent on has failed? In a true IMA system, functions must be independent of each other. This must be accounted for in the deployment strategy

- **IO latency**: in order to reduce the complexity of the partition operating system the IO drivers are usually implemented in a dedicated IO partition, such that each IO device is 'owned' by a partition. This means that a function cannot directly access the IO and must communicate via the intermediary IO partition. This will introduce latency to the IO traffic not experienced in a federated architecture.

- **SW reuse**: software originally intended for a federated architecture must be adapted to such an extent that the usefulness of reusing the software must be questioned.

- **Verification**: each computing node, embedded software, and the communication network between the nodes must be individual verified and validated. This can lead to an extensive test campaign with schedule implications if changes have to be introduced late in the lifecycle.

It is therefore essential that the system architect, or the system integrator, is able to follow an adequate process fully supported by tools in order to identify which deployment strategy is preferable.

3. Approach

The proposed approach is model centric. It relies in the description of the architecture of the system using a unique modelling language. However, a clear separation is maintained between the application and the architectures of both IMA and federated systems. This shall make possible to easily adapt the application to any of the two architectures.

In addition, non-functional requirements are attached to the application. This includes the dynamic characteristics (e.g. Worst Case Execution Time) and expected behaviour (e.g. activation frequency). These requirements are used to automatically check the scheduling aspects at model level on the two architectures using specialized tools. This already makes possible first iterations and refinements on the scheduling requirements and eventually select one the two architectures very early in the process.

Once the scheduling has been checked on the two architectures, the configuration and deployment code of the application on the two architectures is automatically generated for a dedicated target (e.g. PC/Linux workstation or LEON/RTEMS board). The application can then be executed on the target in order to validate the scheduling in a representative environment.

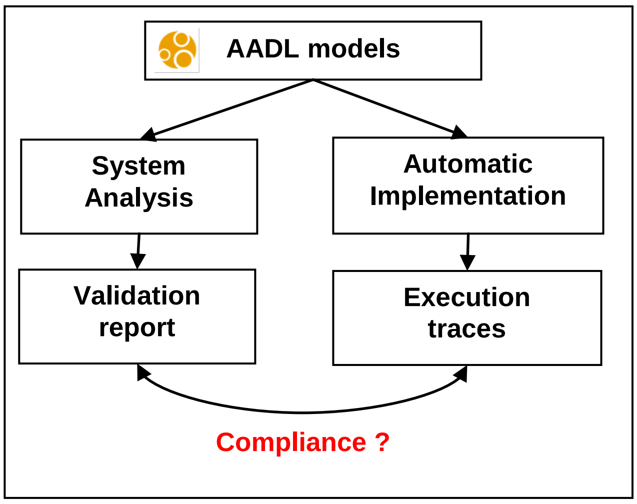


Figure 4- Process architecture

4. Process definition & implementation

The overall architecture of the proposed process is illustrated in figure 4. It relies on the Architecture Analysis and Design Language (AADL) [2] to specify system architecture with its execution constraints. Then, models are processed by appropriate tools to validate system requirements (such as schedulability) or produce system implementation. By using the same specifications during system development phases and relying on automatic tools, we improve the development process consistency and avoid traditional errors or pitfalls (misuse of tools, manual error coding, etc.).

We tailor this generic process for exploring different architecture patterns, as illustrated in 5. We separate specifications in two layers: one generic that represents software aspects (tasks, subprograms, data types, etc.) and another with architecture-specific aspects. By doing so, we keep the common part (the application) in models and reuse it with platform-specific components. As illustrated in 5, we tailor this process to study federated and IMA architectures and investigate the impact of selected architecture on software aspects.

Next sections introduce the modelling language (AADL) and then, explain its supports for validating system requirements or generating the implementation on different architectures.

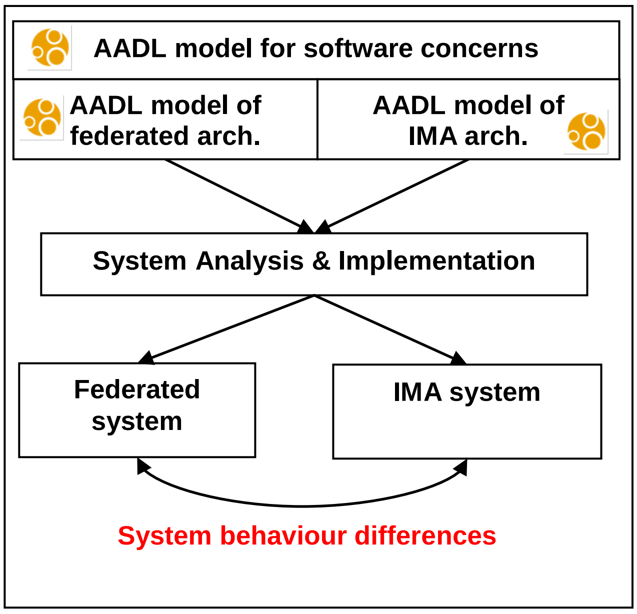

AADL model for software concerns
AADL model of federated arch.
AADL model of IMA arch.
System Analysis & Implementation
Federated system
IMA system
System behaviour differences

Figure 5 - Tailoring the process
for architecture exploration

4.1. Overview of AADL

The Architecture and Analysis Design Language [2] (AADL) has been designed by the SAE (Society of Automotive Engineers) and aims at modelling systems architecture. It allows the description of both software and hardware concerns and focuses on the definition of clear block interfaces. Models are expressed using a graphical or textual interface.

The AADL language [2] defines several components categories: hardware, software and hybrids. Hardware components are:

1. `Processor`: model the physical processor (e.g.: x86 or LEON) and its associated operating system (e.g. Linux or RTEMS[15]).
2. `Virtual processor`: separate the processor into several parts, either for the hardware (modelling several cores) or software (modelling partitions) aspects.
3. `Memory`: model any memory in the system, from RAM to ROM, or hard drives.
4. `Bus`: model a regular bus such as Ethernet.
5. `Virtual bus`: represent either part of a bus (for example, modelling QoS aspects of a bus) or software concerns (protocols, etc.).
6. `Device`: model any other device in the system (sensor, actuator, etc...).

Software components are :

1. `Process`: as a UNIX process, represents an address space where threads are executed.
2. `Thread`: as POSIX threads, supports subprogram execution.
3. `Subprogram`: models instructions flow. It represents a function or a procedure in a language (e.g. C/Ada) executed by a thread.
4. `Data`: models either a data type (e.g. integer, float, etc.) or a data value (shared variable, etc.).

A special hybrid component (`system`) aggregates all system components altogether and so, makes the whole model.

A component can contain other components, in order to describe the hierarchy of the system. For example, a `process` component may contain several `thread` components.

AADL artefacts are associated with properties to represent specific aspects, such as timing requirements (period/deadline of a task) or constraints (size of a partition). The built-in standard property sets address most system properties while users can also add their own. AADL contains a set of properties, which can be extended by the user.

AADL components may use `features`, to model communication interfaces with other components. (For example, the arguments of a function can be considered as features for an AADL `subprogram`). Then, the `connections` section of a component connects these features, describing the data flow within the system architecture (For example, connecting thread `features` to `subprogram features` means that the data of the thread are used as arguments in a function-call).

4.2. IMA and federated architectures modelling

As AADL supports the modelling of both software and hardware concerns and provides extensions mechanisms, we can tailor it to capture different architecture patterns. In that context, we use or adapt it to represent IMA or federated architectures.

Modelling federated architectures does not require specific pattern or properties and the built-in AADL constructs provides everything that is required:

1. Application concerns are represented using several `processes` connected by `buses.`
2. Each `process` (application) is bound to a `processor` (execution support) and a `memory` (for storing code and data).

On the other hand, describing IMA architecture requires using specific modelling patterns and properties to capture the partitioning/isolation policy. For that purpose, we tailor the language as follow:

1. AADL `processor` components are decomposed into `virtual processor`. `Processors` represent the isolation layer (separation kernel) with its time isolation policy while `virtual processors` model partitions environment. Application `processes` are

no longer associated to a `processor` (a regular OS) but to a `virtual processor` (a partition execution environment) to describe allocation of application over partitions.
2. AADL `memory` components are divided with memory sub-components to model segments and the space isolation policy. `Processes` (application) are bound to these `memory` sub-components to represent segment allocation within the partitioned architecture and partitions isolation.

4.3. Validating scheduling from models

As IMA architectures separate partitions with timing isolation, analyzing their execution would show its impact on application behaviour before implementing the system. To do so, we rely on tools that process AADL models and analyze system properties related to timing requirements. They provide the ability to validate scheduling concerns using specific heuristic computations or just simulation.

In our process, this analysis effort is carried by Cheddar [6], a scheduling validation tool. It already supports several analysis techniques for federated architecture and provides necessary features to describe specific scheduling requirements using a dedicated language. Thus, we can tailor Cheddar to specify IMA timing requirements so that designers can analyze the impact of the execution platform (IMA or federated) on the application behaviour.

4.4. Code Generation

Once system requirements (such as scheduling) are validated at specification-level using appropriate tools, implementation needs to be created. Since producing the system by hand is error-prone and could make the implementation inconsistent with validated requirements, code is produced from specifications with appropriate tools.

For that purpose, our process relies on Ocarina [8], an AADL toolsuite with code generation functions that produces C or Ada code from models. It already supports federated architectures by generating POSIX-compliant code. On the other hand, it needs to be tailored to support IMA specific configuration directives and multi-layered architectures (time/space isolation, inter/intra-partition communications, etc.).

For that reason, we modify Ocarina and add IMA support for Xtratum [3] so that it can generate implementation that targets both federated and IMA systems with respect to their specific requirements.

4.5. Run-Time Support

Generated code is then linked with a run-time that supports application execution. System developers choose an appropriate run-time platform according to the expected architecture pattern. Federated architecture are deployed on top of well-known operating systems such as RTEMS [9] or RT-Linux while IMA systems uses specific projects such as XTratuM [3] or POK [5].

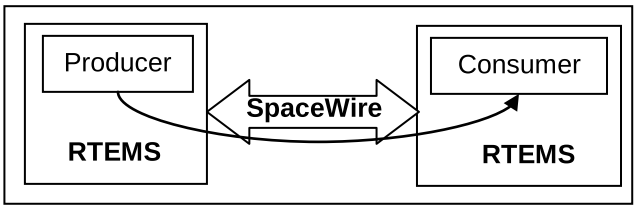


Figure 6 - Federated deployment

As our study aims at studying the differences of the integration of the same software architecture on different architectures, we use two representatives run-times. The federated architecture (figure 6) is built with two nodes executing RTEMS that communicate through a SpaceWire link. The IMA deployment (figure 7) is designed with two RTEMS instances on the same processor using XtratuM [3] which isolates each one in terms of time and space.

5. Case-Study & Results

5.1. Case-Study Specifications

Our case-study deploys a producer/consumer application on federated and IMA architectures. It consists of two tasks, which timing properties are reported in table 1:

1. One cyclic that produces data periodically
2. One sporadic triggered when receiving data

	Task 1	**Task 2**
Type	Cyclic	Sporadic
Period	100ms	100ms
Execution time	3ms	5ms

Table 1 - Timing requirements of each task

Each task is bound to a separated process and the difference between federated and IMA deployment consists in few modelling variations:

1. Federated architecture associates each `process` to a separate `processor` connected through a SpaceWire bus to transfer data, as depicted in figure 6.
2. IMA architecture collocates the two `processes` on the same processor and communication is performed using inter-partition communication, as shown in figure 7. Each partition has a fixed time slice of 200ms to execute its associated application.

This example clearly shows the difference between these architectures patterns: the federated uses a hardware-based isolation model (by using different processors and hardware bus) while the IMA relies on a software-based isolation mechanism (the XtratuM [3] isolation layer).

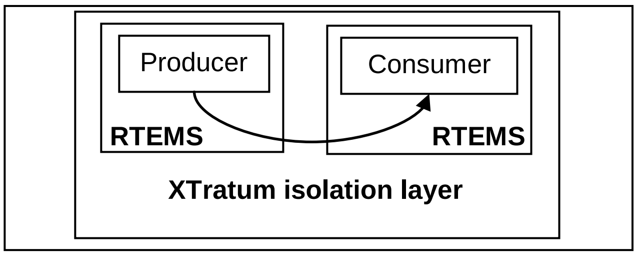


Figure 7 - IMA deployment

5.2. AADL modelling

AADL models are separated in two parts:

1. Software concerns (deployment-agnostic) describe application aspects: data types to be used, tasks and processes. This is reused by all architectures.
2. Deployment aspects describe the underlying architecture that supports application execution. One model is designed for each studied architecture pattern.

The software aspects are defined with two AADL `process` components: one for the producer and another for the consumer, each one containing a `thread` component to execute the application (subprograms). These two processes exchange data using AADL `ports connection`. Figure 8 depicts these aspects using the AADL graphical notation.

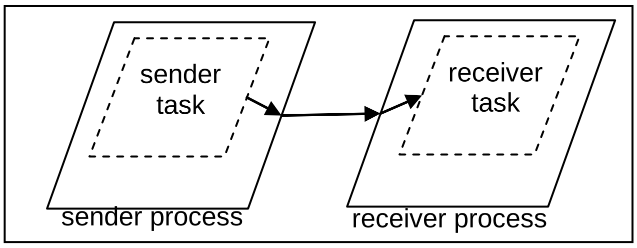


Figure 8 - AADL model of software aspects

Then, the federated architecture is described by adding two nodes connected through a bus. Each node contains a processor component (hardware processor and the execution runtime), a memory (to store application code and data) and a device (communication through the bus). Then, we associate software concerns (processes and connections) to the hardware architecture. This is illustrated in Figure 9 that depicts the association between the generic software components and hardware components of the architecture.

Deployment of the software architecture in an IMA platform with AADL is illustrated in Figure 10. Each hardware component is decomposed into sub-components to represent the separation: the main `memory` component is divided in two `memory` segments, each one allocated to a separate partition (`process` component). Similarly, the `processor` component is divided into two `virtual processor` components, each one representing a partition runtime that supports application execution. Finally, as we don't use any bus, this hardware architecture does not contain any device or bus: the application connection is handled using the inter-partitions communication layer provided by the isolation kernel layer.

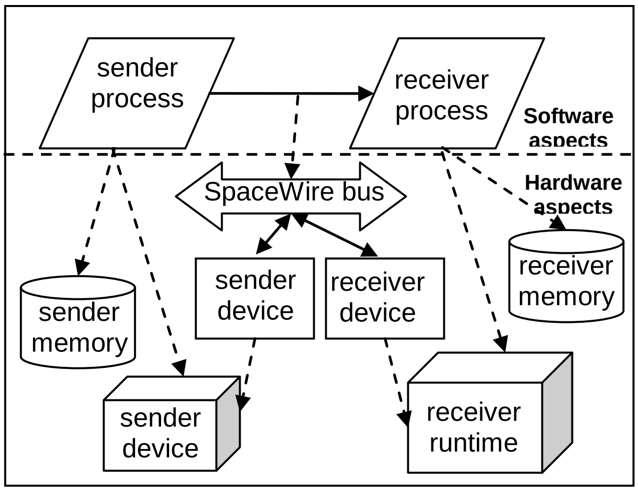


Figure 9 - Federated architecture model with integration of software aspects

Specific timing requirements are specified by associating specific properties on the textual notation of AADL components. Unfortunately, due to a lack of place, it cannot be included in this article but can be found on our main project website [1].

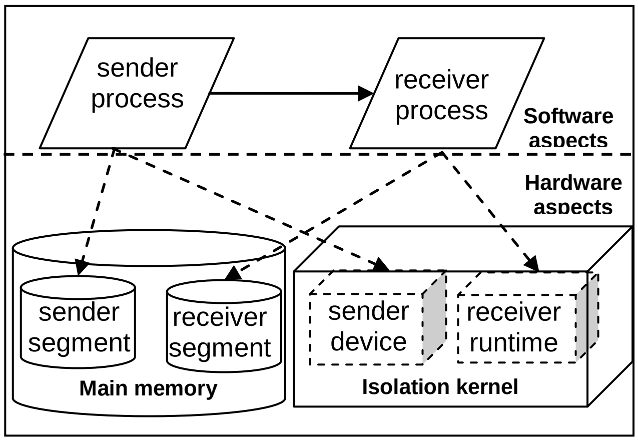
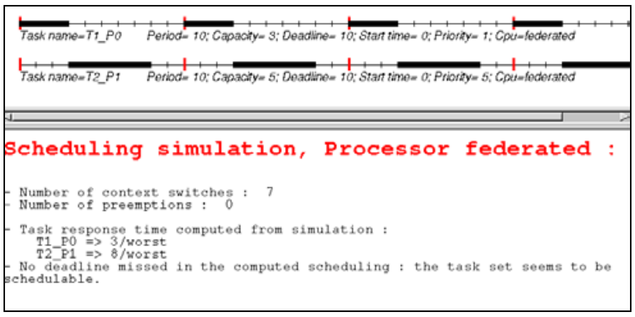


Figure 10 - IMA architecture model with integration of software aspects

5.3. Scheduling validation

Before implementing architectures, one interest consists in validating some characteristics. This provides the ability to check for potential error early and avoid any re-engineering efforts due to late design errors detection. Due to the specific scheduling policies of studied architectures, one concern is to validate timing requirements using system specifications. For that purpose, the proposed process simulates the architecture and ensures timing constraints enforcements.


Task name=T1_P0 Period= 10; Capacity= 3; Deadline= 10; Start time= 0; Priority= 1; Cpu=federated
Task name=T2_P1 Period= 10; Capacity= 5; Deadline= 10; Start time= 0; Priority= 5; Cpu=federated

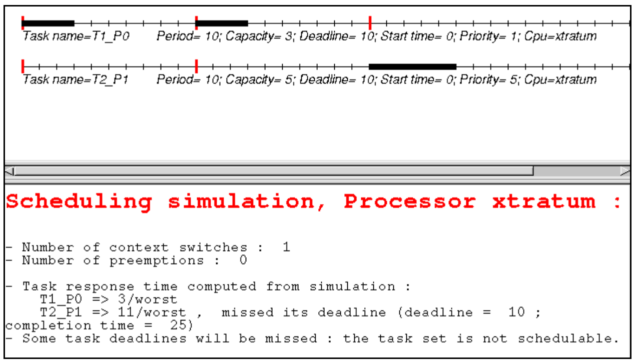
```
Scheduling simulation, Processor federated :

- Number of context switches :  7
- Number of preemptions :  0

- Task response time computed from simulation :
    T1_P0 => 3/worst
    T2_P1 => 8/worst
- No deadline missed in the computed scheduling : the task set seems to be
schedulable.
```

Figure 11 - Scheduling validation of the federated architecture

To do so, we rely on the Cheddar [6] scheduling analysis tool. It processes system specification and validate timing requirements either by using feasibility tests (such as Rate Monotonic Analysis) or simulating its execution according to tasks specification (period, deadline, execution time) and deployment constraints (scheduling algorithm, processor, etc.).


Task name=T1_P0 Period= 10; Capacity= 3; Deadline= 10; Start time= 0; Priority= 1; Cpu=xtratum
Task name=T2_P1 Period= 10; Capacity= 5; Deadline= 10; Start time= 0; Priority= 5; Cpu=xtratum

```
Scheduling simulation, Processor xtratum :

- Number of context switches :  1
- Number of preemptions :  0

- Task response time computed from simulation :
    T1_P0 => 3/worst
    T2_P1 => 11/worst ,  missed its deadline (deadline =  10 ;
completion time =  25)
- Some task deadlines will be missed : the task set is not schedulable.
```

Figure 12 - Scheduling validation of the IMA architecture

Scheduling diagrams produced by Cheddar are shown on figure 11 (federated architecture) and figure 12 (IMA architecture) and depict tasks activation time. While tasks of the federated architecture enforce their deadline as expected, the one from the IMA architecture is delayed due to partition timeslot. In fact, as partitions timeslots are longer than tasks period, partitions switch avoid tasks to be dispatched on time. In our case-study, the sender task cannot send new data instance on time because the second partition still own the processing resource when it is supposed to be activated.

5.4. Implementation Generation & Execution

Once system specifications are analyzed, it is of particular interest to compare them with the run-time behaviour. For that purpose, we implement the system using of automatic code generator, Ocarina [7, 8], which is able to integrate the same application code on different architecture.

While Ocarina already supports federated architectures since several years, its compliance with IMA architecture was limited to POK [5], an ARINC653-compliant run-time oriented for the avionics domain. So, we tailor the Ocarina code generation tools to support XtratuM [3], a hypervisor that is able to execute and isolate several RTEMS instance on the same processor. This deployment is more representative of space-related applications.

The implementation of the federated architecture relies on RTEMS 4.8 for LEON2 processors. We also use a dedicated version that supports the RASTA board and their associated SpaceWire interfaces. On the other hand, the IMA architecture uses the XtratuM 3.1 hypervisor with RTEMS 4.8 as partition run-time (tailored to be executed on top of XtratuM).

Finally, to avoid timing issues due to the instrumentation code, metrics are sent after system execution. This is particularly important for the federated architecture, where the standard output uses the serial line and potentially consumes significant processing resources.

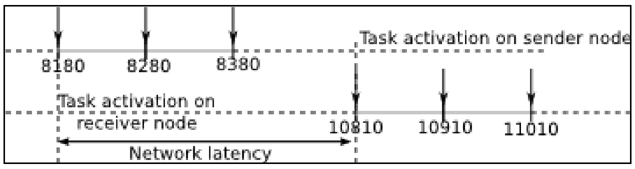

Task activation on sender node
8180
8280
8380
Task activation on receiver node
10810
10910
11010
Network latency

Figure 13 - Tasks activation time during system execution (federated deployment)

Sender	7980	8080	8180	8280	8380
Receiver	10810	10910	11010	11110	11210

Table 2 - Activation time of tasks on federated deployment (in ms)

5.5. Tasks activation validation at execution

Tasks activation times are reported in table 2 (Federated architecture) and table 3 (IMA deployment). We also report them in figure 13 and figure 14 in order to have a graphical overview of timing differences between these two architectures.

In the federated architecture (table 2 and figure 13), there is a huge time delay between activation of the producer and receiving by the consumer. This time difference is due to the network latency: the sender node must invoke additional code to send the data through the SpaceWire interface and the receiver must also invoke networking driver code to retrieve fresh data instance. While this introduces latency between data production and consumption, each task enforces its deadline.

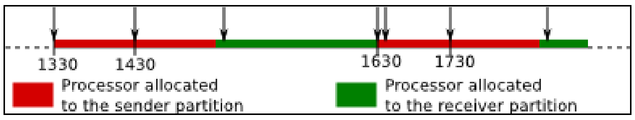


Figure 14 - Tasks activation time during system execution (IMA deployment)

Sender	1330	1430	1630	1630	1730
Receiver	1440	1840	2240	2640	3040

Table 3 - Activation time of tasks on IMA deployment (in ms)

The IMA architecture (Table 3 and figure 14) does not have this latency issue because tasks communication is performed locally, using the hypervisor. However, the partitioning policy has an impact on tasks activation and may delay data processing. In the current example, as each partition is executed during 200ms, and thus, tasks execution from idle partitions is delayed. Moreover, as the partitions exchanged only one data, the receiving partition receives one data instance (the fresher one) when activated so that some data are lost. This would be solved by using queuing ports.

These preliminary results demonstrate that a deployment strategy would have an impact over application concerns and there is no ideal solution that may solve every potential issue: if deadlines are correctly enforced in the federated architecture, network latency could be a problem when data must be processed quickly. On the other hand, partitioned architectures avoid network latency concerns but may delay tasks activation, depending on the partitioning policy.

Finally, it is also interesting to notice that results obtained during execution are different from the simulation. In consequence, if preliminary validation is an important matter when designing safety-critical architecture, inspecting the implementation would still be mandatory.

5.5. Memory footprint analysis

Automatic implementation generation provides the ability to retrieve metrics and evaluate advantages and drawbacks of each deployment strategy. In our case, we evaluate the memory footprint and compare the cost of the implementation on selected architectures.

	Size
Sender node	235124 bytes
Receiver node	234 916 bytes
Total	470 040 bytes

Table 4 - Memory footprint of binaries for the federated deployment

Table 4 and table 5 report the memory footprints of our example on both federated and IMA architectures. Binaries were compiled with the RTEMS tool-chain (with GCC and its associated tools) and stripped to remove useless symbols.

The results show that the lowest memory footprint is obtained with the IMA architecture. Indeed, each partition has an approximate size of 150 Kbytes while applications that target a federated architecture as big as 230 Kbytes. This difference comes from the run-time costs: binaries for federated architectures have to embed device drivers code and other layer that are either useless in the IMA deployment (no required driver or function provided by the isolation layer).

	Size
Sender partition	154308 bytes
Receiver partition	154756 bytes
Isolation kernel	85 268 bytes
Total	394 332 bytes

Table 5- Memory footprint of binaries for the IMA deployment

This shows that IMA architectures present great benefits compared to traditional run-time. A lightweight memory footprint implies that produced application will have less code to be reviewed and so, it would reduce validation activities, reducing the development effort and its associated costs.

6. Conclusions & Perspectives

A seamless process aiming at facilitating the System architecture exploration has been defined. The tools supporting that process have been integrated and their use has been fully automated. The approach has been validated through its execution on a very simple application. As a result, the analysis of different architectures for the execution of an application is straightforward. The proposed approach allows the identification of the main drivers of the future implementations at an early stage. This analysis can be moved forwards up to the generation of code and its execution on a representative platform. This execution is the ultimate proof of the respect the timing requirements and the only able to provide valid budget reports.

Thanks to the automation of verification and generation tasks, the proposed process is particularly adapted to the development of complex systems using an incremental approach. After first verifications and selections at model level, additional verifications and adaptations can be easily performed on a representative environment.

Further steps are still needed to completely validate the approach. The process has been developed and validated on an extremely simple application having a limited number of requirements. In a next step, we intend to apply the same process to a Use Case more representative of complex space applications that include safety or security requirements. This will require the refinement of the architectures of both IMA and federated systems.

The study case has selected XtratuM to support the partitioning but other IMA solutions for space applications exist or are currently developed as VxWorks653 and PikeOS. These two products could be supported in the future in order to enlarge the exploration area.

ESA has initiated several studies (as COrDeT [16]) and supports a PhD project (Definition, realization and evaluation of a software reference architecture for use in space applications [17]) via the Networking/Partnering Initiative. These activities are dealing with component models where the modelling activities are focused on the functional parts of the applications. The integration of these models, describing the application software, with the models describing the architecture of the execution platform clearly appears of prime interest. This should make possible to automatically generate deployment solutions from the high-level definition of the functions of the system. To this end, the proposed process shall be extended to integrate new models or tools able to transform the models into AADL. Of course, the new process shall be able to manage the complete set of requirements in order to support their verification and ensure their traceability.

Glossary

AADL	Architecture Analysis & Design Language
IMA	Integrated Modular Avionics
QoS	Quality of Service
TASTE	The Assert Set of Tools for Engineering

References

[1] TASTE: http://www.assert-project.net/taste
[2] AADL: http://www.aadl.info
[3] XTratuM: http://www.xtratum.org
[4] PolyORB-HI-C User Guide, 2007.
[5] POK: http://pok.safety-critical.net
[6] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar : a flexible real time scheduling framework. ACM, 11 2004.
[7] T. Vergnaud, B. Zalila and J. Hugues. Ocarina documentation, see http://ocarina.enst.fr.
[8] J. Hugues, F. Kordon, L. Pautet, and T. Vergnaud. A Factory To Design and Build Tailorable and Verifiable Middleware. In Proceedings of the Monterey Workshop 2005 on Networked Systems: realization of reliable systems on top of unreliable networked platforms.
[9] RTEMS: http://www.rtems.com
[10] ARINC653: http://www.computersociety.it/wp-content/uploads/2008/08/ieee-cc-arinc653_final.pdf
[11] James Windsor, Kjeld Hjortnaes, « Time and Space Partitioning in Spacecraft Avionics », smc-it, pp. 13-20, Third IEEE International Conference on Space Mission Challenges for Information Technology, 2009.
[12] T. Pareaud, L. Planche, D. Mylonas et al. “Securely Partitioning Spacecraft Computing Resources: Validation of a Separation Kernel”, Proceedings of the DASIA 2011 Conference. DAta Systems In Aerospace. 17–20 May 2011. San Anton, Malta.
[13] J. Almeida & M. Prochazka. “Safe and Secure Partitioning with PikeOs: Towards Integrated Modular Avionics in Space”. Proceedings of the DASIA 2009 Conference, DAta Systems In Aerospace, 26–29 May 2009, Istanbul, Turkey.
[14] WindRiver VxWorks 653 Platform. http://www.windriver.com/products/platforms/safety_critical_arinc_653/
[15] RTEMS Centre. http://rtemscentre.edisoft.pt
[16] J.L.Terraillon, A. Jung. Faster, Later, Softer: COrDeT, a reference on-board software architecture for spacecrafts, Proceeding of ERTS 2010.
[17] M. Panunzio and T. Vardanega: ”A Component Model for On-board Software Applications”. Proc. of the 36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA’10), September 2010 [109].