Benefits of Model Based System Engineering for Avionics Systems

Thierry Le Sergent, François-Xavier Dormoy, Alain Le Guennec
thierry.lesergent@ansys.com
francois-xavier.dormoy @ansys.com
alain.leguennec@ansys.com

9, rue Michel Labrousse,
31100, Toulouse, France

Keywords: Model Based System Engineering, Software Engineering, SysML, Domain Specific Language, SCADE, Avionics System, Aircraft Braking System

1. Introduction

Avionic system design is an extremely complex activity due to:

- Different concerns from different teams:
  - Complexity of functions to realize
  - Hardware and software redundancies to manage safety considerations
  - Communication protocols
  - Complex hardware network with configuration switches,
  - Etc.
- Amount of data to manage: 1,000s of data
- Resource sharing principle (CPU, network, …)

Current methods to manage system engineering complexity rely on refinement levels and view-points. Industries promote ICDs (Interface Control Documents) as a contractual means to assign the development of system components. These are basically large databases that gather the result of system architecture design.

In the domain of safety critical software, the benefits of a model based approach are well established. Now, using an example of avionics architecture, we will show how Model Based System Engineering (MBSE) brings many benefits when the tools are suited for system designers and applied to solve industrial challenges.

Proponents of MBSE provide strong arguments for improving the efficiency of complex system development, promising to achieve at least the same level of quality in a shorter time. However, this gain in efficiency must be proven. This paper details the way a realistic complex avionics system can be designed efficiently using a MBSE tool, with hundreds of data passed through ARINC 429 and ARINC 664-P7 messages.

A classical methodology is followed to manage the different levels of concern:

- Functional architecture
  - Functional data flow communications
- Software architecture
  - Function organization into software components
  - Software message definition and propagation between the components
- Platform hardware architecture
  - Definition of buses, switches, and computing units
  - Software to hardware mapping
  - Virtual link definition for the ARINC 664-P7 communication
- Software design, code generation and verification with SCADE Suite [10]

In order to sustain industrial deployment as a cost-effective MBSE tool, the following must be addressed:

- Interoperability – By conforming to a standard, such as SysML [6], users avoid the risk of vendor lock-in and can integrate with other technologies.
- Usability – In order to overcome reluctance to change, the tool should speak the language of system engineers rather than forcing engineers to force their design into an awkward formalism.
- Efficiency - Tools and methodology should bring savings in design, analysis, and document generation

The SCADE System Avionics Package delivers the following answers:

- Fully customizable interface driven by Domain Specific Languages
- Clean separation and consistent relationships of the Functional, Software and Hardware layers
- Templates for standard avionics protocols (ARINC 429, ARINC 664-P7, CAN provided)
- Intuitive hierarchical data modeling and automated ICD generation

These means are detailed in this paper; the design of the industrial case study that is first introduced has demonstrated the efficiency of the tool support.

2. Avionics system case study

For this case study, Dassault Aviation provided a simplified representative example of a Braking System, based on a COM-MON architecture. Principles of the COM-MON design are not detailed but focus is made on the ARINC 664-P7 and ARINC 429 communications between the COM, MON and other systems.

- At the functional level, COM and MON interact with 9 sub-systems. 175 functional data are considered.
- At the software level, COM and MON partitions are respectively interacting with 12 partitions. 14 ARINC 664-P7 messages and 48 A429 messages are defined.
- The platform level contains 4 Processing cores (CPU), 4 Switches, a dual A/B ARINC 664-P7 network, and 30 Virtual Links (VL).

Based on this example, we realized a complete retro-fit in SCADE System through the following steps:

- Description of the entire architecture (Functional, Software and Platform)
- Production of the application ICDs
- Management of data consistency checks across the complete application
- Management of Virtual Link paths to facilitate switch configurations
- Analysis of message allocation with respect to network bandwidth
- Synchronization of the system architecture with the software design

Screen shots and details of the complete example are provided after we have introduced all the tool features that enable efficient support of the design method.

3. Proposed Model Based Engineering Method and Tool Support

As introduced in the first section, a traditional system design approach is used, relying on several abstraction layers: functional, software (sometimes called “logical”), and platform. Our approach is unique in how it defines and manages the relations between these levels of abstraction.

The following technical means are hereafter detailed:

- Allocation and interface specification, relying on data management
- Customization of SysML objects
- Analysis of the model, in particular to produce the Interface Control Documents (ICD).

3.1. Allocation and interface specification

When designing a system in several abstracting layers, consistency and completeness of each layer must be ensured. The difficulty comes from the fact that the abstraction levels are not a simple hierarchical decomposition. The functional, software, and hardware views are all hierarchical, but with a different hierarchy.

The relationship between the different layers is called projection. SysML comes with a dedicated construct, allocations, to support this. Projections provide a straightforward way to show the realization of a function by a software component, and show how a software component is run by a hardware component.

Before focusing on the interface allocations themselves, it should be noted that the intention of a projection is to specify precisely which item shall be supported by which item. The projection must take into account that blocks may be instantiated several times. The difficulty arising from the hierarchical instantiation is solved in SCADE System though a block replica mechanism detailed in [13]. The result of the replication mechanism is that each object from the real world is represented with a dedicated object in the model. This replication mechanism is reused for the data objects presented below.

Projection of the component interfaces can be realized in the same simple way as for block components. But this does not lighten the burden of making all interfaces consistent with each other. Let’s consider the following case:

- A function F produces data D, for several other functions defined in the functional decomposition.
- Each function is allocated to a software component, some sharing the same component, others not.

- The software components exchange messages in order to transmit the functional data.

It is easy to specify that the software component S supporting F shall produce a message M carrying D. But what about the other sides of the communication? The message M must be sent to all software components that support a function receiving data D. This rule must be maintained whatever the data propagation and function allocations, something that may evolve during the design process.

Another consideration is that ARINC 429 and ARINC 664-P7 messages can carry several data elements. The designer may define either one message carrying several data to other software components, even if some data are not required by all the message recipients, or to define several messages. All these challenges are supported by a concept of data.

3.2. Data management

SCADE System comes with an original and powerful means to specify the interface allocations efficiently. It relies on data objects that can be structured and propagated.

The data propagation mechanism first published in [13] is summarized here:

- Data are implemented as SysML blocks, allowing SCADE System to conform to the SysML meta-model, but managed as data in the tool.
- Data are propagated along block ports and connectors thanks to a dedicated feature of the tool
- Propagation of a data D out of a block B through one of its ports consists in the creation of a data proxy in the parent blocks instantiating B.
- All proxies are connected as a chain; the tool allows managing the data attributes (feature detailed in section 3.3) whatever the selection of one proxy of the chain.

This mechanism brings two important benefits. First, it allows handling a complete “path” through a block hierarchy and connections from a single data element. Second, it allows managing thousands of data, each broadcasted or multi-casted in a straightforward way by the designer, without graphical scalability issues by avoiding the creation of numerous ports and connectors (the proxies are automatically created from path selection).

Individual data are structured thanks to an internal model transformation:

- Data structures are defined with classical structured types
- Applying a structured type to data replicates the structure inside of the data definition. This allows for each data to have its own individual fields.

Structured data is an ideal means to model messages; the message fields represent place-holders for the functional data. Simple allocations of functional data (defined at the functional abstraction layer) to a field of a structured data message represents the fact the message carry the functional at a certain place in the message. This would not be possible without the internal replication of the message type because the field of type would be shared by all the messages sharing the same type.

Allocating a propagated functional data to a propagated message data implements the interface allocations between the abstraction layers.

An additional feature allows managing the user interface scalability: There may be several thousand of functional data, and data messages in a model. Managing the allocation between the different levels of abstraction is challenging for the usability perspective of a graphical tool. Even a traditional list of objects can be cumbersome for the users. SCADE System IDE comes with a customizable filtering mechanism that restricts possible candidates to correct allocations and propagation. The principle is the following:

- A checking rule verifies that a functional data FD propagated from function FA, allocated to Component CA, to function FB, allocated to component CB, is allocated to message data MD that goes from CA to CB. This check takes into account both the possible multiple propagation target of a data, and possible multiple allocations to messages to handle redundancies.
- In addition to automated verification and report generation at the end of the design, the checking rule is used dynamically by the UI to filter out the allocation and propagation possibilities that would lead to a violation of the rule. That way, the UI exposes a much shorter list of possible selections, and correct designs are realized much faster.

This mechanism extends very easily: users can program their own verification rules from the model API, and use them both for final batch verifications, and as filters in the allocation and the data propagation interfaces.

3.3. DSL vs. SysML: Configuration of the SCADE System tool

The method presented above only requires blocks, ports, connectors, data, and structured type objects. However, these constructs fall short of easily addressing a specific domain. The ability of a tool to meet the needs of domain expert engineers is a key success deployment factor, which must go beyond these few generic SysML objects.

There are traditionally two competing approaches when it comes to domain-specific system modeling:

- "Pure" Domain-Specific Languages (DSLs)
- Customizations of generic modeling solutions, e.g. UML with profiles such as SysML

Proponents of the first approach appreciate the total freedom that DSLs provide in order to best fit actual domain concerns, without incurring the cost of supporting legacy UML peculiarities. Proponents of the second approach claim that reliance on standards such as UML provide better long term sustainability and interoperability.

SCADE System stands out with an original hybrid approach that combines the best of both worlds: While the internals of SCADE System rely on UML/SysML and profiling (therefore providing full standard support and interoperability), SCADE System still presents itself as a domain-agnostic system modeling language and toolset, via a pure DSL "virtual layer" implemented transparently on top of UML and SysML. This layer is naturally extended to accommodate domain-specific needs without starting from scratch: The domain specialist simply designs a domain-specific language as a meta-model that specializes SCADE System's domain-agnostic meta-model. Additional information is conveyed by appropriate attributes in specializing meta-classes and relationships among them; additional constraints imposed by the domain are enforced by redefinitions of existing relationships.

SCADE System transparently handles the intricacies of UML profile management and dynamically adjusts its interface to the domain-specific "configuration", offering new modeling constructs (in toolbars, property pages, etc.) while forbidding others according to constraints of the domain. Moreover, models based on a domain configuration can be manipulated via a "pure" API derived from the domain meta-model, not UML concepts and stereotypes.

Figure 1 below shows the tool workflow in practice: the domain specialist designs the domain meta-model; from it, a "configuration plug-in" is automatically generated. By loading this plug-in file, SCADE System is transformed into a Domain Specific tool with the customized "creation palette" and object properties.

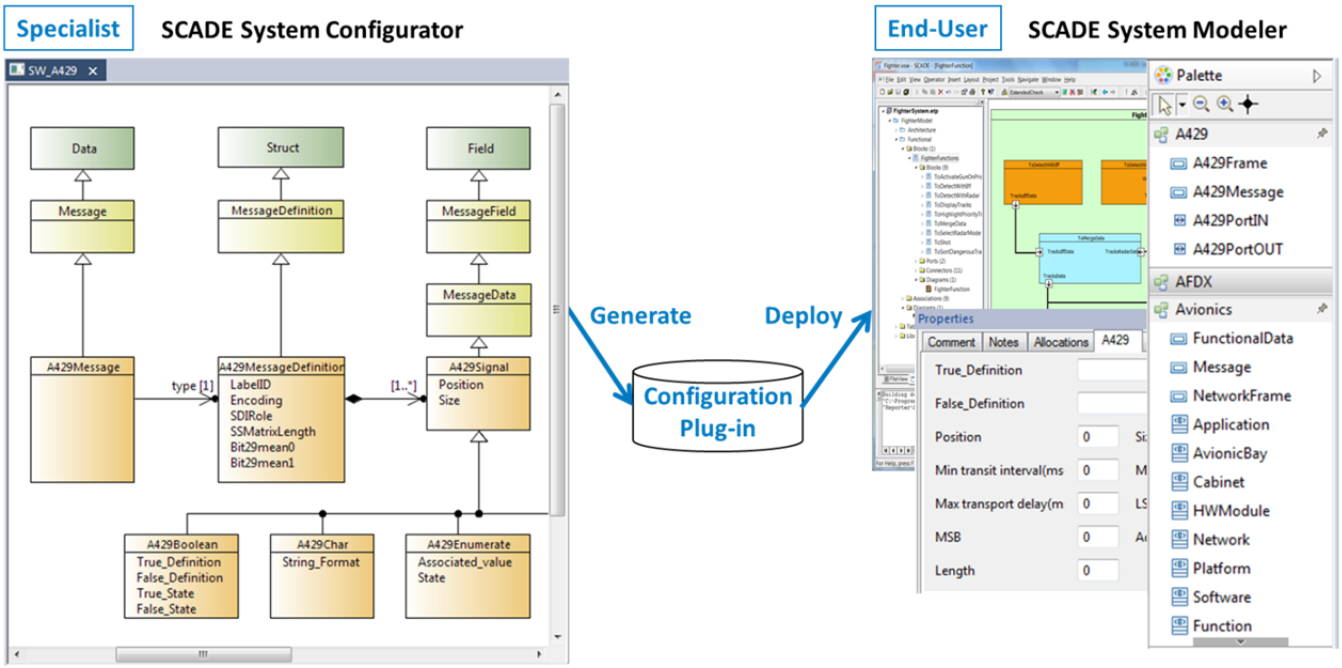

Figure 1: System Configuration Workflow

Figure 1 also shows an extract of the ARINC 429 meta-model.

- At the top of the diagram, Data, Struct and Field refer to SCADE System meta model;
- Below, Message, MessageDefinition, MessageField and MessageData are defined in the "generic" Avionics configuration. They all inherit from SCADE System meta classes
- Finally at the bottom of the diagram, the ARINC 429 meta-classes refine the Avionics meta-classes, providing all information needed for the definition of ARINC 429 messages.

Similar meta-models are provided for ARINC 664-P7 and CAN messages. The same technique is used to define all meta-classes needed at the platform definition layer, with the physical ports, switches, etc.

3.4. ICD production

Now that we have described the data management process, Interface Control Documents can be generated from the model. These Excel sheets are needed by many different development teams, each requiring different information, presented in the way they expect. For example:

- Partition table
- Messages definition with their parameters
- Messages source and targets with ARINC 664-P7 ports, transmission rate, length, etc.
- ARINC 664-P7 Virtual Links definition

Each table being specialized for a specific usage, the tool should not rely on dedicated built-in tables, but offer a means for each user to customize their own view. SCADE System comes with a simple but powerful means:

- A dedicated dialog allows defining the hierarchy of objects to display in a table: the list of possible children, references, or even arbitrary queries is proposed for selection.
- For each line in this tree, the list of possible attributes, references, or arbitrary queries is proposed to create columns in the table.

These lists of possible children, references and attributes are directly exposed from the model API. This includes the domain meta-model API specified. So, in a few clicks one can for example set-up the table of all ARINC 429 messages, with their target and with the parameters from data allocated from the functional design.

All table examples listed below have been produced for the designed aircraft braking system. They comply with the original MS Excel tables that were given at the start of the project. Of course the displayed tables can be exported in a click to MS Excel files.

4. Application to an Avionics system case study

We will now apply the methodology and tools to the avionics system case study described at the beginning of the paper. As explained in section 3 we proceed in layers focusing on data management (producing and consuming).

4.1. Functional architecture

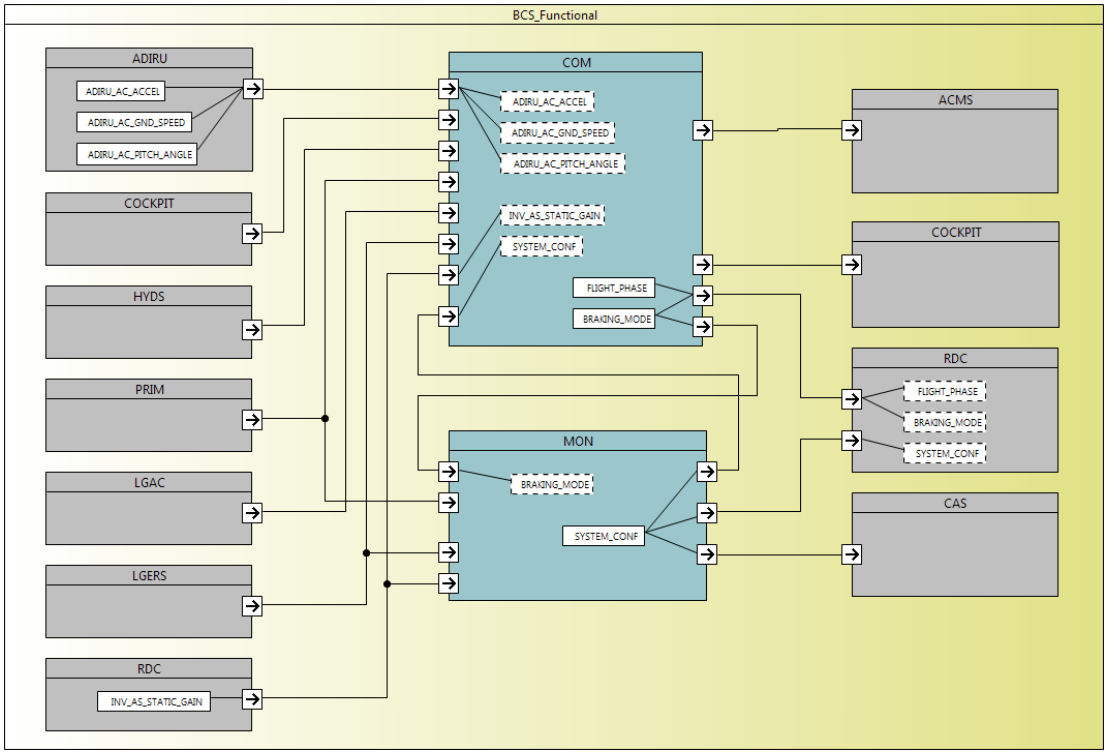

Figure 2: Functional architecture

Beginning with the functional architecture, we define the functions of the Braking System and the data produced and consumed (required) for each function.

The COM function of the Braking System needs 3 functional data produced by the ADIRU Function (ADIRU_AC_ACCEL: Aircraft Acceleration; ADIRU_AC_GND_SPEED: Aircraft Ground Speed; ADIRU_AC_PITCH_ANGLE: Aircraft Pitch Angle). The graphical diagram shows the functions and their data flow dependencies. Designers have fine-grained control over the functional data displayed in the diagram.

As a result of this stage, Functional Data with producer/consumer tables are available for each Function as illustrated in Figure 4 below for the ADIRU and a global table with all functional data produced and consumed by the Braking System application. Automated checks allows immediate detection of functional data that are not produced or functional data that are not consumed (useless data).

A	B	C
Name	Block Source	Block Target
ADIRU_AC_ACCEL	ADIRU	COM
ADIRU_AC_GND_SPEED	ADIRU	COM
ADIRU_AC_PITCH_ANGLE	ADIRU	COM

Figure 3: ADIRU functional data table

4.2. Software architecture

The software architecture defines the software components of the Braking System and the messages produced and consumed (required) by each software component. Again the messages are propagated in the software architecture by the data propagation mechanism.

In the example illustrated below, the BCS_COM software component of the Braking System requires one message (MSG_ADIRU_COM_C10: ARINC 667-P7 message) produced by the ADIRU component.

The Software architect may use several diagrams to focus on ARINC 664-P7 or ARINC 429 communication as illustrated in figures 5 and 6 below.

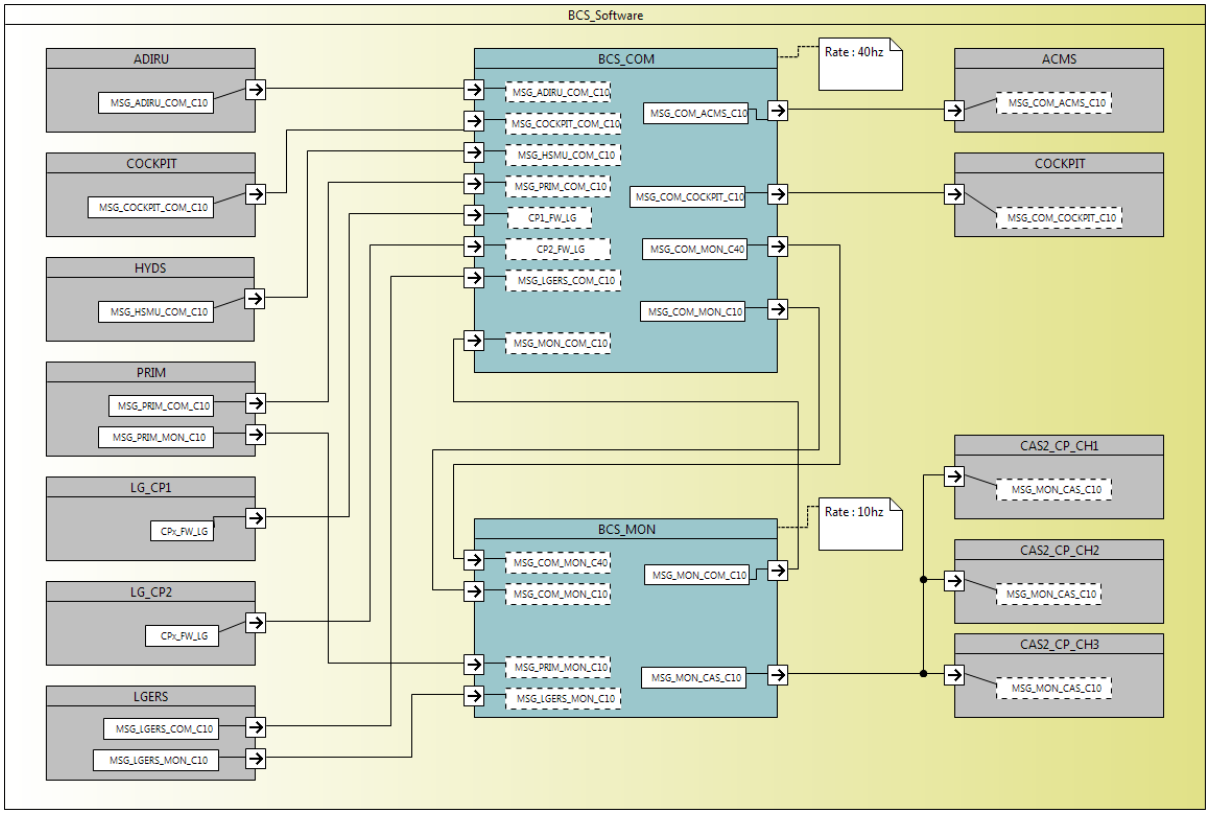

Figure 4: Software architecture focusing on ARINC 664-P7 Communication

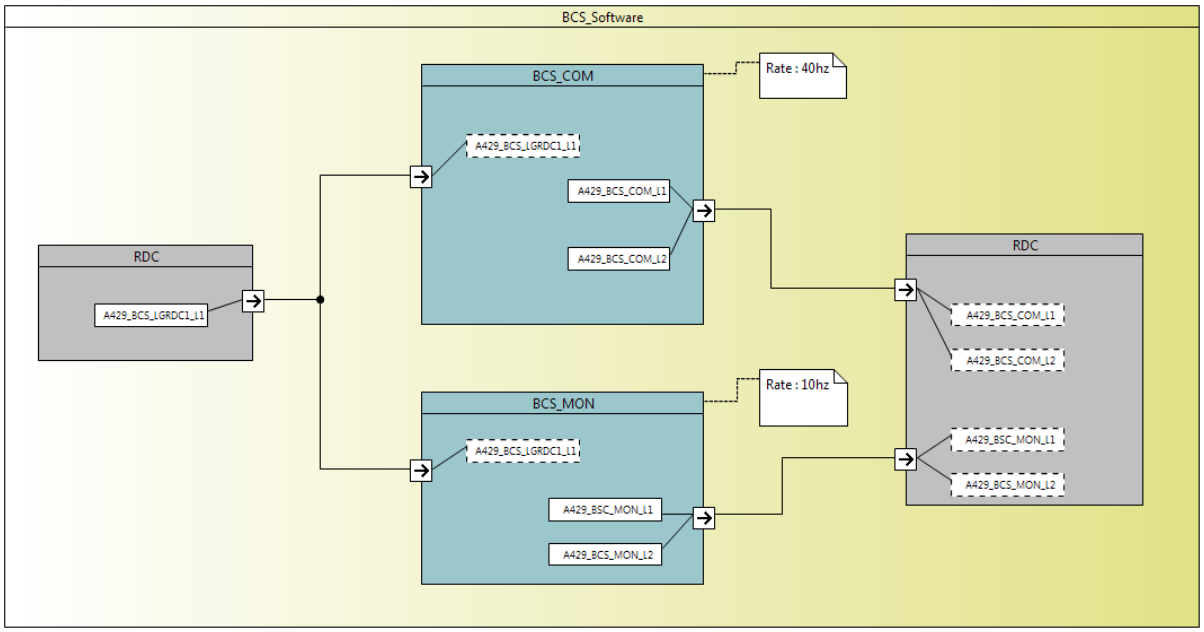

Figure 5: Software architecture focusing on ARINC 429 Communication

At the software level we also describe all messages (ARINC 664-P7 and ARINC 429). Figure 7 illustrates the MSG_ADIRU_COM_C10 (ARINC 664-P7) message, with its three data set containing the functional data AC_GND_SPEED, AC_ACCEL and AC_PITCH_ANGLE.

		A	B	C	D	E
		Name	DS_ID	Address	Length	Message Type
1	CP_FW_LG	CP_FW_LG				NonProtocol
2	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10				NonProtocol
4	Res	Res		0	4	
5	FS1	FS1		4	1	
6	FS2	FS2		5	1	
7	FS3	FS3		6	1	
9	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	1	8	4	
10	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	2	12	4	
11	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	3	16	4	

Figure 6: ARINC 664-P7 Message definition Table

Data Set information are gathered in another Table as illustrated below in figure 8.

		A	B	C	D	E	F	G	H
		Name	LSB	MSB	Type	TrueDefinition	FalseDefinition	OperationalMin	OperationalMax
1	CP_FW_LG	CP_FW_LG							
2	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10							
4	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	0	0	real			0.0	4096.0
5	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	0	0	real			-4.0	4.0
6	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	0	0	real			-180.0	180.0

Figure 7: ARINC 664-P7 Data Set Table

4.3. Functional to Software mapping

Now that we have defined the functional and the software architecture, the functional to software mapping consists in the allocations of functions to the software components (illustrated in figure 9), and the functional data mapping to software messages (illustrated in figure 10). Allocation tables are used for that purpose.

		A	B
		Function	Software
1	FunctionInSoftware1	ACMS	ACMS
2	FunctionInSoftware10	LGAC	LG_CP2
3	FunctionInSoftware11	LGERS	LGERS
4	FunctionInSoftware12	RDC	RDC
5	FunctionInSoftware13	MON	BCS_MON
6	FunctionInSoftware14	PRIM	PRIM
7	FunctionInSoftware2	ADIRU	ADIRU
8	FunctionInSoftware3	CAS	CAS2_CP_CH1
9	FunctionInSoftware4	CAS	CAS2_CP_CH2
10	FunctionInSoftware5	CAS	CAS2_CP_CH3
11	FunctionInSoftware6	COCKPIT	COCKPIT
12	FunctionInSoftware7	COM	BCS_COM
13	FunctionInSoftware8	HYDS	HYDS
14	FunctionInSoftware9	LGAC	LG_CP1

Figure 8: Function to Software allocation Table

169	DataInMessage71	ADIRU_AC_ACCEL	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_ACCEL
170	DataInMessage72	ADIRU_AC_GND_SPEED	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_GND_SPEED
171	DataInMessage73	ADIRU_AC_PITCH_ANGLE	BCS_ICD::B_Software::BCS_Software::ADIRU::MSG_ADIRU_COM_C10::DS_ADIRU_AC_PITCH_ANGLE

Figure 9: Extract of Functional Data to Software Message allocation Table

Automated consistency checks confirm that these allocations are consistent with the data and message propagation: the message to which a functional data element is allocated to must be produced by a software component to which the producer function is allocated to, and must be propagated to the software components to which the consumer function is allocated to. Inefficiencies are also detected, for example when a message carries a data not needed by all recipients of the message. One can now analyze in detail the design tradeoff for message definitions.

4.4. Platform architecture

At this stage we capture the platform components (CPUs, switches, buses, external devices, etc) describing the possible resources (CPU, memory, bandwidth, etc) available to the architect to deploy the application. The BCS platform consists of 4 CPUs, 4 Switches, an external device (RDC) and several buses as described in Figure 11.

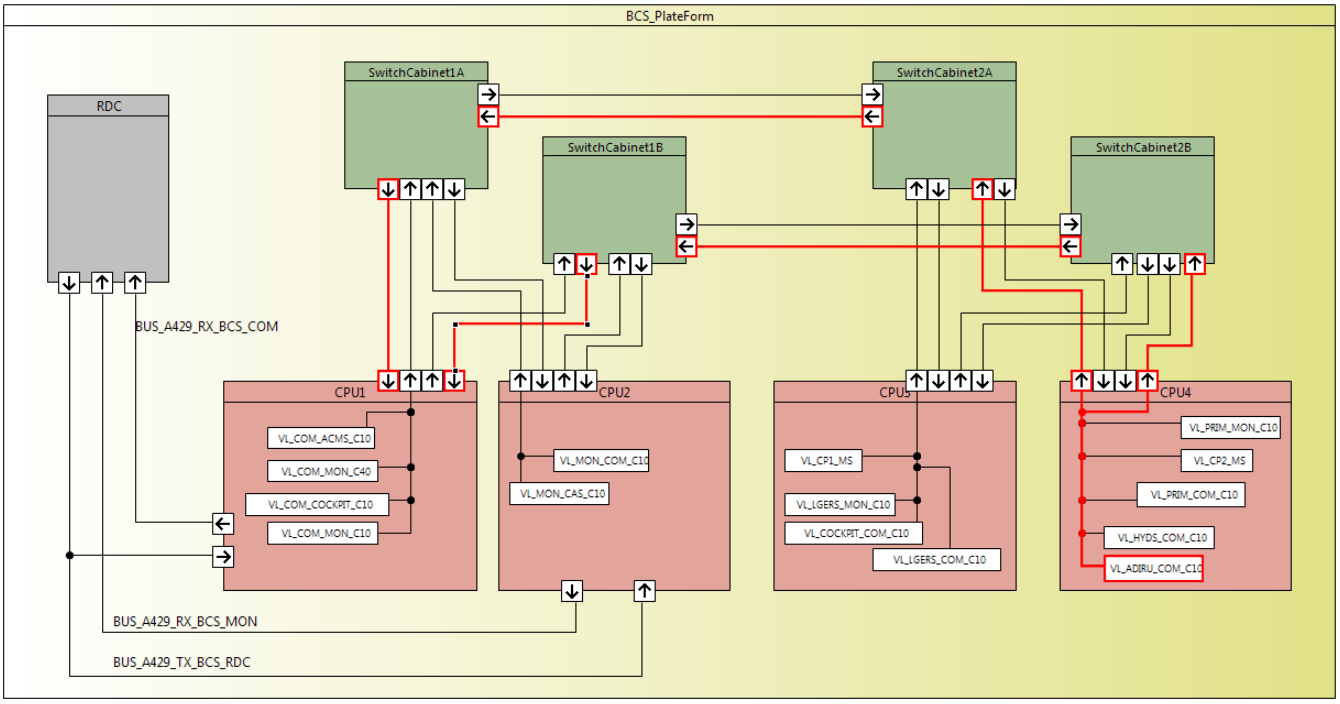

Figure 10: Platform architecture illustrating Chanel A, Chanel B Virtual link routes

At this platform level, we allocate software components to the CPUs and define message paths between them. For ARINC 664-P7, these paths are known as Virtual Links. Again, these paths are implemented as dedicated data propagated in this architecture. For example, with the ADIRU software being hosted in CPU4 and the COM being hosted in CPU1, the VL associated to MSG_ADIRU_COM_C10 are defined for channel A and channel B, following the classical redundancy means in ARINC 664-P7 communication.

Virtual Links routing are automatically produced in Tables enabling configuration of switches.

4.5. Software to Platform mapping

One of the important tasks for the integrator is to allocate the software components onto the platform and optimize usage of available resources. This is one of the main challenges for IMA (Integrated Modular Avionics).

Allocation of the software components (Partition) to CPUs is illustrated in Figure 12 below.

		A	B	C
		Name	Source	Target
1	Partition1	Partition1	ACMS	CPU3
2	Partition10	Partition10	LG_CP1	CPU3
3	Partition11	Partition11	LG_CP2	CPU4
4	Partition12	Partition12	LGERS	CPU3
5	Partition13	Partition13	RDC	CPU3
6	Partition14	Partition14	PRIM	CPU4
7	Partition2	Partition2	ADIRU	CPU4
8	Partition5	Partition5	CAS2_CP_CH1	CPU3
9	Partition6	Partition6	CAS2_CP_CH2	CPU4
10	Partition7	Partition7	CAS2_CP_CH3	CPU3
11	Partition8	Partition8	COCKPIT	CPU3
12	Partition9	Partition9	HYDS	CPU4
13	PP32_BCS_COM_P	PP32_BCS_COM_P	BCS_COM	CPU1
14	PP32_BCS_MON_P	PP32_BCS_MON_P	BCS_MON	CPU2

Figure 11: Software (Partition) to Platform projection

In the same way software messages allocated to Virtual Links are illustrated in table 13 below.

8	MessageToFrame2	MessageToFrame2	MSG_ADIRU_COM_C10	VL_ADIRU_COM_C10
9	MessageToFrame3	MessageToFrame3	MSG_COM_COCKPIT_C10	VL_COM_COCKPIT_C10

Figure 12: Extract of Message to Virtual Link allocation

We can then perform resource usage checks, such as bandwidth checks, as illustrated in Figure 14 below.

Category	Code	Message
Error	CHK_102	**The sum of software needs exceed hardware capacity (CPU2.HWPort3.bandwidth= 0.0 < 680.0) at link**
Error	CHK_102	**The sum of software needs exceed hardware capacity (CPU2.HWPort4.bandwidth= 0.0 < 560.0) at link**

Figure 13: ARINC 664-P7 bandwidth checks results

4.6. Braking System ICD (Interface Communication Document)

Thanks to this final allocation step we have a complete description of data communication from the functional level down to platform integration from which ICD tables are automatically produced as illustrated in Figure 15 below.

		A	B	C	D
		Name	Address	Length	Rate
1	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10			
3	MSG_ADIRU_COM_C10	MSG_ADIRU_COM_C10			
5	Res	Res	0	4	
6	FS1	FS1	4	1	
7	FS2	FS2	5	1	
8	FS3	FS3	6	1	
10	DS_ADIRU_AC_GND_SPEED	DS_ADIRU_AC_GND_SPEED	8	4	
11	DS_ADIRU_AC_ACCEL	DS_ADIRU_AC_ACCEL	12	4	
12	DS_ADIRU_AC_PITCH_ANGLE	DS_ADIRU_AC_PITCH_ANGLE	16	4	
14	To_BCS	To_BCS			40

Figure 14: Extract of Braking System ICD

5. Comparison with other approaches

The NASA handbook [4] states that a clean process must be set up to "*identify and resolve interface incompatibilities and to determine the impact of interface design changes*". Management of Interface Control Documents (ICDs) [5] is indeed at the center of most industries' system engineering processes, but most often supported by tedious manual processes based on MS Excel files cross analysed and reviewed by the different engineering teams involved.

More robust processes rely on databases. They provide scalability, and the database schema enforces some design rules. However, this is not as powerful as the meta-modelling capability provided by UML-based tools, and does not support a graphical representation which is an essential communication means between engineers.

SysML [6] is the most known standard in system engineering, but it does not come with specific constructs to model software messages that are exhanged between "blocks". SCADE System Avionics is compliant with SysML, and provides such constructs. Other standards have been set-up which focus on the component interfaces. We now compare our approch with three of them that have been deployed in the industry.

EAST-ADL [7], initially defined in the European ITEA EAST-EEA project that has been then aligned with the AUTOSAR automotive standard was a good source of inspiration. In the same way as the method presented here, the central features of EAST-ADL, is its multi-layer approach, which defines multiple abstraction levels and distributes all development information across these levels. Just like the SCADE System Avionics package, EAST-ADL is supported by a UML Profile. The main difference is in the focus we have put on the detailed definition of the component interfaces that gather information from the different levels of abstraction. On the other hand EAST-ADL provides a focus on timing information that has not been yet considered in the SCADE System Avionics Package.

The SAE "Architecture and Analysis Description Language" (AADL) [8], has its roots in the avionics domain. AADL allows for the description of both software and hardware parts of a system. In contrast to SysML's limited generic components of "block", "port", etc., AADL provides precise modeling concepts to describe the runtime architecture of application systems in terms of concurrent tasks, their interactions, and their mapping onto an execution platform. It also separates the definition of block interfaces from their implementation, but does not make the links between functional, software and platform interfaces in the way presented in this paper.

The more recent FACE standard [9] also focuses on the non-ambiguous specification of interfaces for "Unit Of Portability" (UoP). It describes the component interfaces at three different layers, conceptual (e.g. mass, length …), logical units (e.g. Kg, millimeters …), and platform implementation (e.g. uint8, float64 …). This complements very well our approach. Indeed it allows setting a "meaning" to the functional data used at the top level of the process we have presented, while FACE itself does not detail the way this information shall be carried into software messages.

It is perfectly possible to define with SCADE System Configurator the EAST-ADL, the AADL, and the FACE meta-models. On-going work to support the FACE standard as a complement of the Avionics package presented here will be published in the coming months.

6. Conclusion

This paper presents a model based development approach and tooling enabling efficient and agile process development focusing on data. Applied to a case study based on a braking system example, we have demonstrated a complete flow from functional architecture capture down to platform deployment showing the main benefits of a model based approach:

- Automated checks ensuring that the functional, software, and platform architecture are consistent with each other
- Data flow checks ensuring proper usage and production of data all along the process
- Platform resource and usage domain checks enabling design tradeoff in platform definition
- Automatic generation of ICD (Interface Communication Document)
- Automatic generation of configuration files (for OS and switches)
- Full description of communication (CAN, ARINC 429, ARINC 664-P7, discrete)
- Full description of Software architecture

Other important topics covered at tool and methodology level are not addressed in this paper:

- Requirement traceability all along the process
- PLM/ALM (Product or Application Lifecycle Management) linkage
- Product Line Engineering and Variant Management

- Synchronization of System and Software engineers work

Starting from the provided ICD files of the braking system example, the complete retro-fit in SCADE System has been achieved in a couple of months. Thanks to the model based solution, with a user interface that speaks the language of the avionics system designer, design modification and regeneration of consistent ICDs is now mature. This demonstrates a large efficiency and the capability to significantly accelerate the development cycle in handling design changes.

Well supported by tools with features detailed in this paper, we believe the system design methodology can reach the maturity level the software component design has achieved since several years.

7. Acknowledgment

We want to thank Mickael Lafaye and Jean-Pascal Rotteleur from Dassault Aviation for their help and the fruitful inputs they provided during the Avionics Case Study

8. References

1. “Systems Engineering Handbook, a Guide for System Life Cycle Processes and Activities”, SE Handbook Working Group, INCOSE, January 2010.
2. “ARP4754 Rev A. Guidelines for Development of Civil Aircraft and Systems”, SAE Aerospace, Revised 2010-12
3. “DO 297/ED124 IMA, INTEGRATED MODULAR AVIONICS (IMA) DEVELOPMENT GUIDANCE AND CERTIFICATION CONSIDERATIONS”, RTCA - 2005
4. “NASA Systems Engineering Handbook”, NASA/SP-2007-6105 Rev1
5. “Training Manual for Elements of Interface Definition and Control”. Vincent R. Lalli, Robert E. Kastner, and Henry N. Hartt, NASA Reference Publication 1370, January 1997.
6. “OMG Systems Modeling Language (OMG SysML)”, OMG, Version 1.2, June 2010
7. EAST-ADL, http://www.east-adl
8. “Architecture Analysis & Design Language (AADL)”, SAE Aerospace AS5506B. http://standards.sae.org/as5506b/
9. “Future Airborn Capability Environmennt (FACE)”. http://www.opengroup.org/face.
10. Esterel technologies SCADE products, http://www.esterel-technologies.com
11. “SCADE System, a comprehensive toolset for smooth transition from Model-Based System Engineering to certified embedded control and display software”, Thierry Le Sergent, Alain Le Guennec, François Terrier, Yann Tanguy, Sébastien Gérard. ERTS 2012
12. “Integrating System and Software Engineering Activities for Integrated Modular Avionics Applications”, Thierry Le Sergent, Frederic Roméas, Olivier Tourillion. 2012 SAE Aerospace Electronics and Avionics Systems Conference, Phoenix Arizona, USA.
13. “Data Based System Engineering: ICDs management with SysML”, Thierry Le Sergent, Alain Le Guennec. ERTS 2014