Aircraft Systems Modeling

Model Based Systems Engineering in Avionics Design and Aircraft Simulation

Linköping Studies in Science and Technology
Thesis No. 1394

Aircraft Systems Modeling

Model Based Systems Engineering in Avionics Design and Aircraft Simulation

Henric Andersson

Linköping University
INSTITUTE OF TECHNOLOGY

Department of Management and Engineering
Division of Machine Design
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköping 2009

Aircraft Systems Modeling
Model Based Systems Engineering in
Avionics Design and Aircraft Simulation

LIU-TEK-LIC-2009:2
ISBN 978-91-7393-692-7
ISSN 0280-7971

henric.andersson@liu.se
www.iei.liu.se/machine
Division of Machine Design
Department of Management and Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Printed in Sweden by LiU-Tryck, Linköping, 2009

Abstract

AIRCRAFT DEVELOPERS, like other development and manufacturing companies, are experiencing increasing complexity in their products and growing competition in the global market. One way to confront the challenges is to make the development process more efficient and to shorten time to market for new products/variants by using design and development methods based on models. Model Based Systems Engineering (MBSE) is introduced to, in a structured way, support engineers with aids and rules in order to engineer systems in a new way.

In this thesis, model based strategies for aircraft and avionics development are studied. A background to avionics architectures and in particular Integrated Modular Avionics is described. The integrating discipline Systems Engineering, MBSE and applicable standards are also described. A survey on available and emerging modeling techniques and tools, such as Hosted Simulation, is presented and Modeling Domains are defined in order to analyze the engineering environment with all its vital parts to support an MBSE approach.

Time and money may be saved by using modeling techniques that enable understanding of the engineering problem, state-of-the-art analysis and team communication, with preserved or increased quality and sense of control. Dynamic simulation is an activity increasingly used in aerospace, for several reasons; to prove the product concept, to validate stated requirements, and to verify the final implementation. Simulation is also used for end-user training, with specialized training simulators, but with the same underlying models. As models grow in complexity, and the set of simulation platforms is expanded, new needs for specification, model building and configuration support arise, which requires a modeling framework to be efficient.

The purpose of computing is insight, not numbers.
- Richard Wesley Hamming, in the 1950’th

Acknowledgments

THE WORK PRESENTED in this thesis has been carried out at the Division of Machine Design in the Department of Management and Engineering (IEI) at Linköping University (LiU), Sweden.

There are several people I would like to thank for their support and advice during the work; first I want to thank my supervisor, Professor Petter Krus, for his guidance and support. The most interesting and encouraging work during the period was when we performed the ModuLiTH (Modular Electric Vehicle) project course.

I want to thank my industrial sponsors and supervisors at Saab Aerosystems: Anders Pettersson who initiated the idea of starting a research project, Stefan Andersson who got me deeply involved in the MBSE program at Saab Aerosystems and Erik Herzog for his academic view and an interesting time together in the SPEEDS project.

At Machine Design in Linköping University I also want to thank Olof Johansson for encouraging support and cooperation in the NFFP project, and also Björn Lundén for his introduction to research and studies at the graduate level. For pleasant teamwork with papers, presentations and discussions in the MBSE area at Saab Aerosystems and at the Division of Machine Design, there are several people I am grateful to, among whom I would here like to mention; Bengt-Göran Sundqvist, Johan Ölvander, Anders Weitman, Sören Steinkellner, Hampus Gavel, Gert Johansson, Lars Karlsson, Ingela Lind and Ulrik Pettersson.

The work has been financially supported by the ProViking research program (connected to the Swedish Foundation for Strategic Research), NFFP (National Aviation Engineering Research Program) and Saab Research Council.

Linköping, Mach 2009

Henric Andersson

Papers

THIS THESIS IS based on the following five papers, which are appended and will be referred to by their Roman numerals. The papers are printed in their originally published state except for changes in formatting and correction of minor errata.

[I] Henric Andersson, Bengt-Göran Sundkvist, “Method and Integrated Tools for Efficient Design of Aircraft Control Systems”, *25th International Congress of the Aeronautical Sciences*, Hamburg, Germany, September, 2006

[II] Sören Steinkellner, Henric Andersson, Petter Krus, Ingela Lind, ”Hosted Simulation for Heterogeneous Aircraft System Development”, *26th International Congress of the Aeronautical Sciences*, Anchorage, Alaska, September, 2008

[III] Olof Johansson, Henric Andersson, Petter Krus, “Conceptual Design Using Generic Object Inheritance”, In *Proceedings of the ASME 2008 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference IDETC/CIE 2008*, Brooklyn, New York, August, 2008

[IV] Henric Andersson, Anders Weitman, Johan Ölvander, “Simulink as a Core Tool in Development of Next Generation Gripen”, In *Proceedings of Nordic Matlab User Conference 2008*, Stockholm, Sweden, November, 2008

[V] Henric Andersson, Erik Herzog, Gert Johansson, Olof Johansson, “Experience from introducing UML/SysML at Saab Aerosystems”, Submitted to for publishing in *Journal of Systems Engineering*, INCOSE.

Out of the five papers, Andersson is the main contributing author of papers [I] and [IV].

Contents

Appended papers

1
Introduction

SOFTWARE CONTENT in products is increasing, as is the number of electronic parts and components. Aircraft developers, like many other manufacturers, are today experiencing ever-growing complexity in products but also increasing competition in a global market. To specify, design and verify products before transition to end users is the challenge every industry must face, and become more efficient and careful so as not to let mistakes during development lead to end-product pricing or quality out of market acceptance. For aerospace products, regulations and authorities set specific requirements for safety and environmental reasons.

Consider an engineering challenge where a new function/feature added to an existing product is required, driven by customer needs and constraints by official regulations. Realization is performed by implementing and integrating new equipment and software, to ensure end-user training and provide documentation for usage, maintenance, and certification.

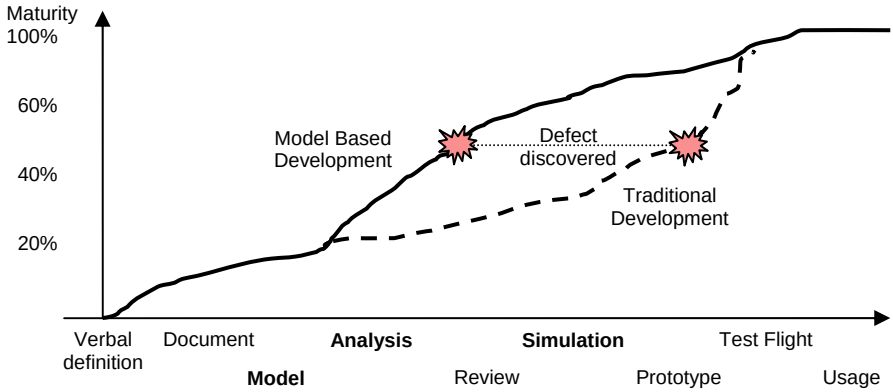


Figure 1. Value of the model based development approach

The development project life-cycle starts with a verbal formulation and ends with roll-out to customer and usage. There are obviously different ways to execute the project, depending on the choice of engineering means, as illustrated in Figure 1. Maturity of functionality is reached through several activities, where modeling, simulation and analysis "get the function to mature" more rapidly compared to traditional document-centric methods. Parts of both validation and verification are performed earlier with a Model Based Development (MBD) way of working, providing greater possibility to find defects or misunderstood requirements early.

With this as a guiding example, further reading of this text will open up and penetrate questions of different aspects of development within the real-time, safety critical and embedded systems area.

To support development of complex products, multiple methods and languages have been proposed in literature and evaluated in industry, as reported in [Mar 1991] and [Stevens 1998]. In recent years there has been much focus on MBD as means for managing complexity by improving specification clarity, consistency and validation support, see [Alford 1992], [Long 1996], [Oliver 1997], [Friedenthal 2000], [Wymore 2002], and [Engel 2008].

A central part of this thesis is thus to formalize the engineering design problem as the trade-off between skills, tools, and process implementation used within a development project.

1.1 Problem domain

The problem area for this thesis is how to achieve efficient development of good products with "Model Based Systems Engineering", and it focuses on questions that arise when a model based approach is to be used in a large development organization. Technical domain is the aerospace domain with integration of embedded and physical systems (avionics and aircraft subsystems). The problem is related to several systems engineering practices and the problem is formulated as;

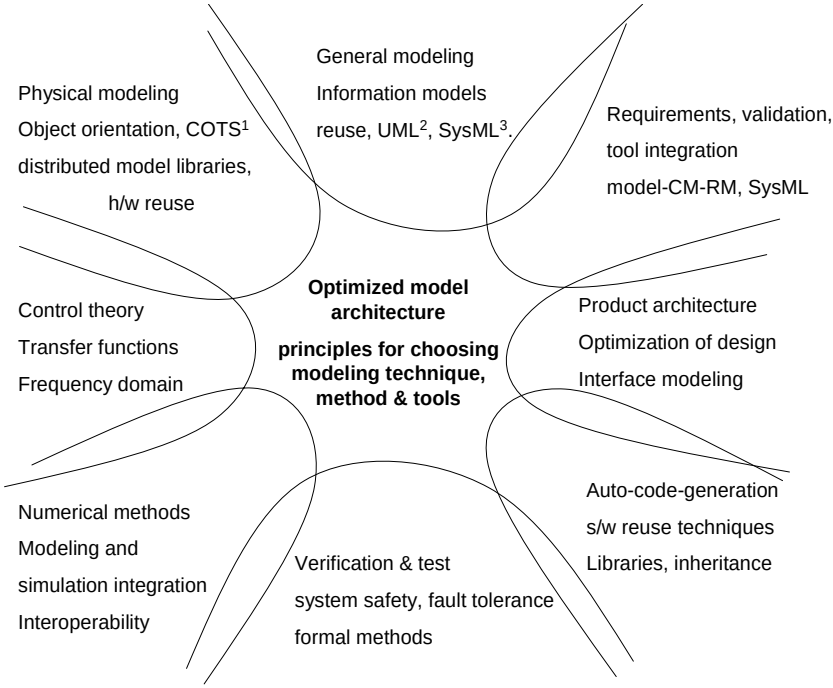
> *How to achieve an* **efficient transition** *from (abstract)* **concept evaluation** *to (detailed)* **development** *and further to* **sustained engineering** *based on a* **model based approach** *supporting* **systems including hardware and software**.

Further explanations of the components of this problem formulation are:

- **Efficient transition** is when information from one activity is available and not in any major degree misunderstood in the following activity or activities.

- **Concept evaluation** is to compare and choose from alternative solutions in the earlier stages of a project where the information is not so detailed or complete in every aspect, but rather abstract.
- (Detailed) **development** is to add the technical details to a selected concept, but also make other choices (as regards the details)
- **Sustained engineering** is to add improvements/upgrades to an existing product, where a mix of detailed and conceptual information is available.
- **Model based approach** is an engineering way-of-working, based on models instead of documents. The models carry information in a format that can be transformed and used for both analysis and documentation purposes.
- The application domain is **systems including hardware and software**, also called hybrid or heterogeneous systems. Hardware is not only computer hardware or electronics, but also pistons, pumps, wheels and other mechanical, fluid or structural hardware.

The central focus in the work is the model based approach, opening up further interesting questions, for example "What is the optimal model architecture?" and "How to choose modeling technique in a specific situation?" These questions relates to systems engineering practices, as shown in Figure 2, and that are partly covered or discussed in this thesis.



1) Commercial-of-the-shelf, 2) Unified Modeling Language, 3) Systems Modeling Language

Figure 2. Problem domain and related engineering practices

For reuse and utilization of verified solutions, component libraries are created. One objective is to use models to optimize the system design on a higher level of abstraction than in current methods. Modeling techniques of the future also need to represent more aspects of the system in the same model. For this there is a need for model (or meta-model) alignment with proven applicable development processes and standards.

1.2 Contributions

Many research programs in this area can present powerful techniques and methods with small examples or with a specific or narrow problem to solve, but scaling up to industrial usage is sometimes of less concern. In this work, scalability is one of the underlying areas of interest, with the assumption that results should be useable for a wider range of employees, not only graduates or senior engineers.

In this thesis a survey is made of different modeling techniques and modeling domains and their further integration. The integration is studied both in terms of combination of modeling techniques and tools, but also integration of the model based approach into the development process. Contributions from this thesis to the field are the combination of emerging modularization technologies together with model based systems engineering and with a scalability focus: scalability in the sense of large systems, large organizations and growing datasets of engineering information. Definition of modeling domains based on tools and related techniques/methods creates a basis for analysis of existing, engineering environments or plan for buildup of new ones. Hopefully, the results from this and similar work will influence the future design of IDE (Integrated Development Environment) for aerospace, automotive or other companies with embedded systems development.

1.3 Research method

This work was performed as a survey of model-based techniques, tools and standards related to aircraft/avionics Systems Engineering. This research field is by nature multidisciplinary. The work does not take up actual systems or system models, but the available techniques and tools that can support engineers in building and using models.

This can be viewed as the meta^2 activity according to [Muller 2004], see Figure 3. Meta^0 is the actual product creation and the development of system model(s), meta^1 is the development methods used to create and manage the product including its related models. Meta^2 is the main focus of this thesis; a study of available methods and tools for the meta^1 activities. At meta^3, the research method for comparing tools and methods at meta^2 is defined.

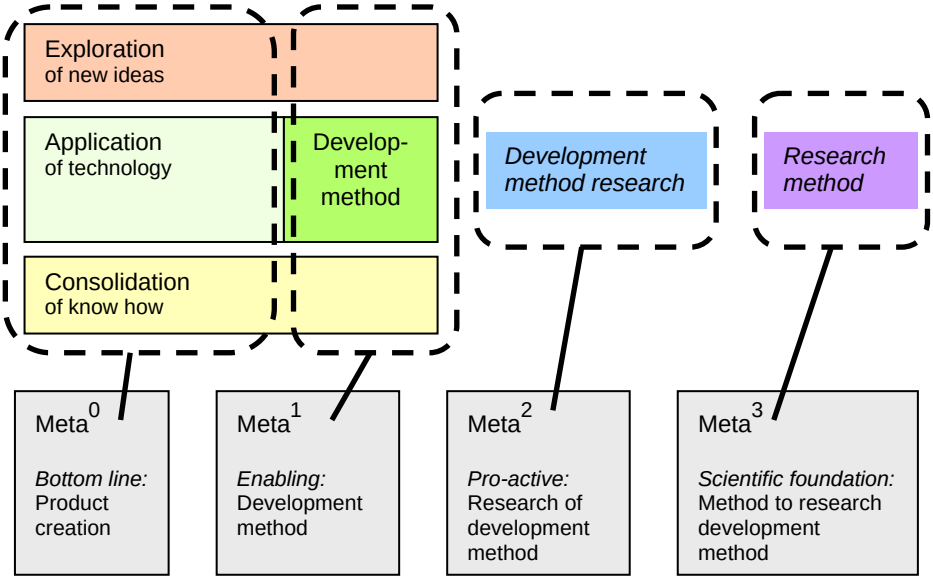


Figure 3. Research method formulation for method research adopted from [Muller 2004]

Because this is an industrial thesis with a focus on industrial large-scale application of both emerging and proven techniques, the method of falsification by reasoning based on academic reports and industrial experience is appropriate and was chosen as the research approach.

1.4 Thesis outline

This thesis consists of an extended summary and five papers. The remaining parts of the summary are outlined as follows; chapter 2 deals with the technical domain of this work; aerospace, avionics, and technical standards affecting development methods within the embedded systems and safety critical systems domains. In chapter 3 the systems engineering discipline is presented, together with development process models/standards and how the model based approach supports the discipline.

Further details regarding basic modeling concepts and applicable modeling techniques are presented in chapter 4 and chapter 5 describes how the tools and techniques are integrated into an engineering development environment. Implications of introducing model based approaches in terms of organizational changes and strategies are presented in chapter 6, and finally chapter 7 concludes the thesis with an evaluation and discussion of results obtained.

2
Introduction to Aerospace and Avionics

THE AEROSPACE ERA has more than hundred years of history. As new technologies have matured and been adopted into aerospace usage, new disciplines contribute to the development of the next generations of products. No one man is any longer able to possess knowledge of all parts of an aerospace project; it is a multidisciplinary challenge and it all has to be preformed in teams. The need for an integrating discipline, which divides the overall challenges into smaller engineering problems, is introduced; Systems Engineering. The Systems Engineering team has the overview of technical areas, formulates new, preferably independent, engineering tasks and sets the common rules. One of the technical areas viewed as a discipline in its own right is avionics, with rapid evolvement enabled by new generations of computer and communication hardware, software languages, and development tools. So, even avionics has grown and become a "multidisciplinary discipline".

Avionics, in the way it is thought of today, is about half as old as aerospace. Avionics is briefly "electronics for aircrafts" and the main function for which the avionics system is responsible is CNS; Communication, Navigation and Surveillance. As in several modern products, new techniques, partly based on software, enable/require both new design solutions and new ways of working when performing the design activity. This chapter provides a background to avionics development and the challenges met in development based on modern, modular architecture.

2.1 Background to building blocks in aerospace

In order to design and build a complex product, with hundreds of people involved, with time from first product ideas to introduction in the order of years, a balanced break-down structure into simpler parts is required. In some sense, the success of a project is dependent upon how well the break-down structure (or architecture) supports the development work. Designing the architecture is to define major building blocks in a system, applying abstraction and decomposition and thereby defining parts and interconnections. The system can be analyzed as a collection of parts together with the relationships between them.

However, in many cases the problem is pushed from the parts to the connections between them which are still complex. The connections can be synthesized into a new field of knowledge such as electromagnetism, physical chemistry, etc. Providing system architecture for avionics applications is in line with this reasoning where the protocols on the buses have turned into a “component”.

It has been proven in practice that a modular approach allows the design of products that satisfies varying requirements through the combination of distinct building blocks. Modularity also improves the ease of development, production, maintenance, reuse and recycling. In development, modularity enables concurrent engineering as teams can work in parallel, relatively independently, with each part/module of the system. In production, modularity allows concurrent assembly where modules can be preassembled, in parallel, separate from the final assembly of the complete product [Blackenfelt 2001].

When viewing the architecture of an evolving system with a lifecycle perspective it is not possible to predict and propose a sustainable technical solution. With a growing proportion of the system implemented in software and/or with electronic parts, choosing and maintaining the architecture is an important issue for the future. An appropriate design will improve the capability to build advanced functionality, and the architecture of the network and choice of components also contributes to cost. The system architecture is an increasingly important part of the complex and configurable product.

Thus, on the one hand, the introduction of building blocks (modularization) simplifies the management of complex products, while on the other hand the interface and configuration (tables) handling are added as a growing and in itself complex activity that will be further discussed herein.

2.2 Saab, Gripen and engineering challenges

Most of the work in and the inspiration for this thesis are related to development projects at Saab Aerosystems, Sweden. The JAS 39 Gripen lightweight fighter aircraft serves as a reference for modeling of large (complex) systems/products, se Figure 4. A short introduction to Saab and Gripen is therefore given here.

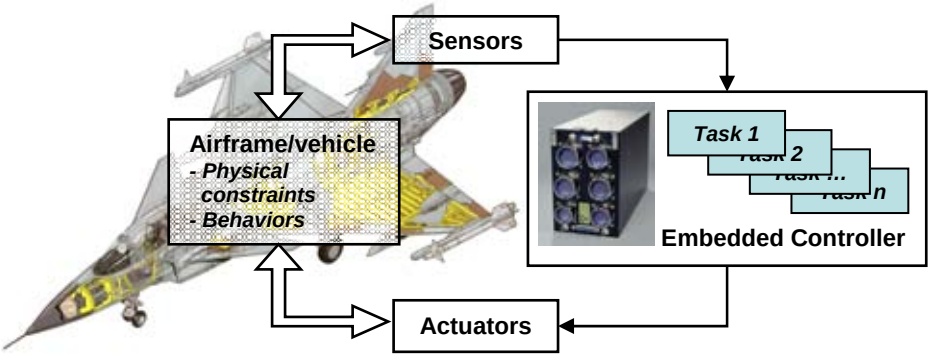


Figure 4. The Gripen fighter as a system model example

The company develops and produces aircraft systems, and the main product during the last decade is the Gripen system, including aircraft, integrated sensors and weapons as well as ground support, operation support and training equipment. Traditionally, the company has a history of few large programs, tightly coupled to a single customer - the Swedish Air Force. The commercial environment for Saab Aerosystems is changing in several ways:

- New product development and production programs with shorter turnaround are being introduced, such as the UAV program (Unmanned Aerial Vehicle) as described in Paper [V],

- The Gripen program is expanding into a multi-customer, export environment which is forcing Saab to improve its handling of product variants and upgrade programs for older configurations,

- Customers expect Saab to identify, fund, and develop new capabilities rather than the traditional customer-paid development model,

- The market's desire to contract for complete systems instead of parts, for example deliver integrated solutions rather than provide aircraft and support equipment under separate contracts.

- Early value creation and communication with suppliers and throughout the supply chain is playing an increasingly critical role.

As a consequence of the changes identified above there is a drive to improve engineering productivity and quality and the introduction of MBSE and, for example, UML/SysML is considered to have great potential. Modeling and simulation is also regarded as a key enabler for early validation and value creation within the supply chain in collaborative product development, as stated in [Fredriksson 2002].

A major internal change initiative at Saab Aerosystems, EMPIRE, described in [Backlund 2000], was conducted over the period from 1994 to 2000 and was connected to the research programs “Lean Aircraft Research Program”, (LARP, based at LiTH, Sweden) and “Lean Aircraft Initiative” (LAI, based at MIT, USA), see [Lilliecreutz 2000]. With the EMPIRE project, a set of systems engineering techniques/tools was introduced along with supporting processes and methods. Further change initiatives in the SE area have been conducted, establishing the prerequisites for continuing engineering support and process improvements, such as Requirements Management, Product Data Management as well as graphical modeling and code generation. The ongoing methods and tools change program at Saab Aerosystems is shortly called MBSE, and reported achievements in Papers [II], [IV] and [V] are related to the MBSE program.

2.3 Aircraft and avionics architecture

In the following pages a short historical description is given of different generations and different design principles of the core avionics architecture. At the end of the section, aircraft systems are defined and one way to categorize aircraft systems is described. The description is based on [Spitzer 2001], [Moir 2004], [Tooley 2007], [Watkins 2007], and on Saab experiences.

2.3.1 History of core avionics architecture

In the 1950s and 1960s, avionics architecture was standalone systems with point-to-point connections, as shown at the left side of Figure 5. Communication was performed unidirectionally by analog voltages, relay/switching contacts and similar.

As more systems were added, especially the cockpit became crowded, but also the weight of controls, displays, relays and wiring increased. More and more information needed to be shared between systems in order to improve functionality and starting in the early 1970s more flexible information handling was enabled through the introduction of digital computers. The transfer of information was nonetheless still performed mainly by analogue means, requiring A/D and D/A converters.

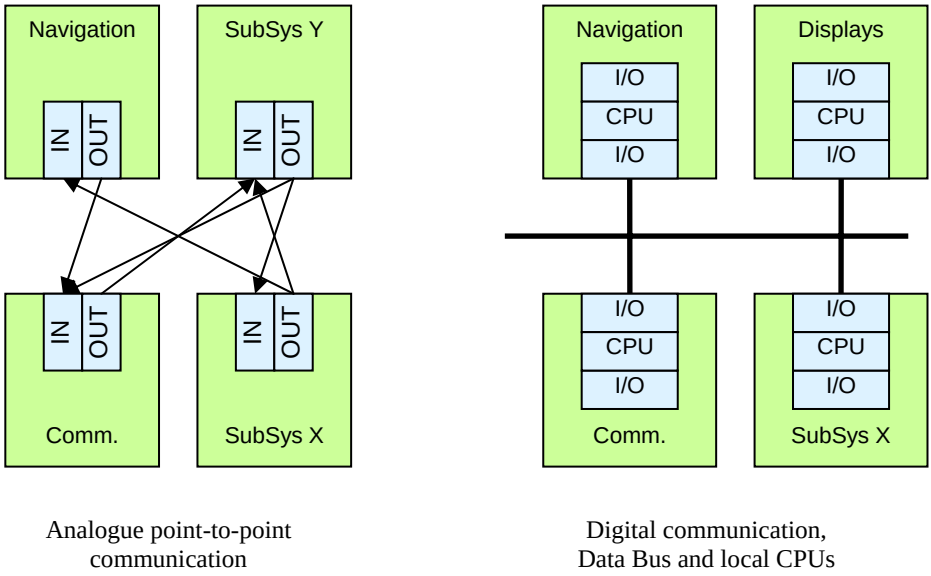


Figure 5. Distributed avionics architectures

The rapid evolvement of avionics technology took off with the introduction the microprocessor, enabling data transfer in digital form. Problems associated with bias and drift, which occurs in analog systems operating in a large temperature span, were reduced. The digital solution also had the benefit that bidirectional data exchange could be supported. Serial data-links became standard. In the late 1970s a major part of information passing was performed with designs using digital communication over serial data-links.

A large step in the increasing "bandwidth over weight ratio" was taken by sharing the interconnecting media, using a parallel communication link, and the 1553 Data Bus[1] was born, and the architecture layout is illustrated by the right side of Figure 5. Subsystems could now send data between themselves, one at a time in a defined sequence, making the system lighter but also more flexible: modifications or additions of new equipment were not longer such an integration nightmare.

[1] The MIL-STD-1553 specification was released in August of 1973 by USAF and was first used in the F-16 fighter aircraft. Successors are MIL-STD-1553A in 1975 and MIL-STD-1553B in 1978, and the "B" version is still in use in many programs, but there exist additions & restrictions.

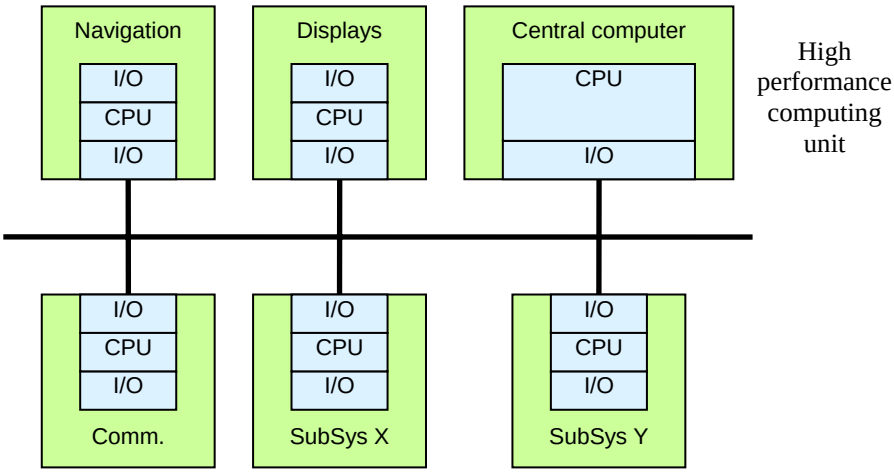


Figure 6. Federated avionics architecture with centralized computing unit

Display technique also made it possible to present more data and in a more flexible way. The Human Machine Interface (HMI) was increasingly based on software designs, and may be regarded as a subsystem in its own right. Different design philosophies regarding "degree of centralized computing" were developed. Many aircraft developers kept a separated or decentralized avionics architecture, with dedicated Electronic Control Units (ECU) for specific aircraft functions/subsystems. At Saab Aerosystems, for example, the design choice for the avionics core functions (such as navigation, communication, displays and weapon control) in fighter aircraft was to centralize the computational resources in one (central computer) or a limited set of Central Processing Units (CPU), in what is called an integrated architecture, see Figure 6. Most of the algorithms and data exchange between subsystems could now be done within the central computer, with the benefits of shorter time delays in time-critical calculations and simplified interface handling. Most of the information in the whole system is available in the same software component (Computer Software Configuration Item, CSCI), so signal/data integration is simplified. Functions such as sensor-data-fusion are easier to implement in a centralized architecture.

As hardware performance increased with new generations of CPUs, it was possible to build large software components in the range of MSLOC (Mega Source Lines Of Code). The drawback as the system grows is dependability, in terms of different aspects, for example:

- Execution dependency; if one arithmetic operation fails (e.g. division by zero or data out of range), the whole subsystem will crash, and the computer needs to reboot. It is more difficult to handle failures locally.

- Criticality dependability, (a consequence of execution dependency); All software in a CSCI has to be developed according to the same (highest) Design Assurance Level (DAL, see "Definition of Design Assurance Level", on page 27), even though only parts of the functionality are critical. More tests have to be performed, and this drives development time and cost.

- Development dependability; because building (compilation & linking) of all software in the large CSCI is done as one activity, all development teams need to be synchronized.

- Security dependability; it is difficult to restrict access to secret parts of the functions/code, while building, testing or debugging, for example should be not too complicated to perform.

In commercial aerospace, led by the business jet community and based on COTS items, the Integrated Modular Avionics (IMA) has evolved, and been adapted also to the military application.

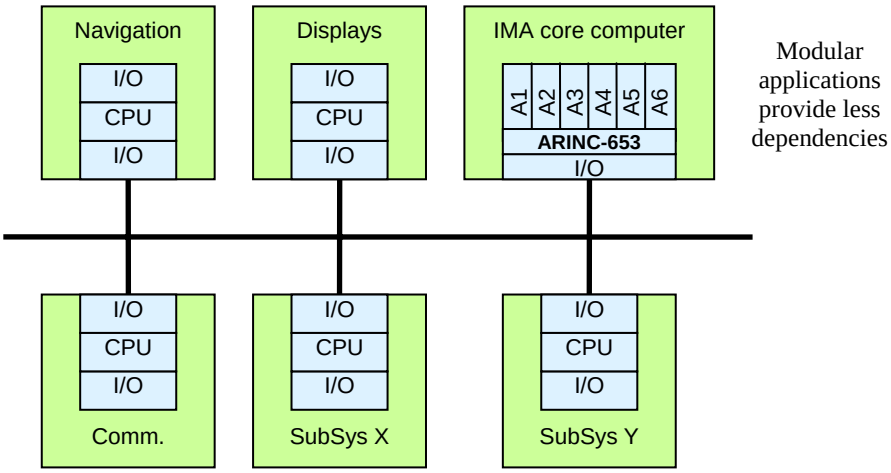


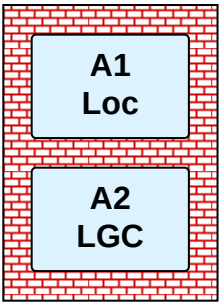
Figure 7. Integrated Modular Avionics (IMA) architecture

With the IMA architecture, shared computational resources are still centralized but explicitly allocated to "applications"[2], (see A1-A6 in Figure 7) through the use of configuration tables. These shared resources include the computing processors, common communications network, and common I/O units. During the allocation process, it is possible to maintain the flexibility to dynamically manage free resources through control of/changes to the configuration tables. With usage of [ARINC-653] partitioning, the applications can be safely separated so applications of different criticality/DAL level

[2] Names found in literature are Partition, Application or Hosted Function

may be hosted in the same physical resource. The principle is physical separation/division in space and time (meaning memory addresses and execution timeframes). ARINC-653 separation of software modules drastically reduces dependability as well as some of the drawbacks of the integrated architecture.

The term “brick-wall partitioning” is used in the context of ARINC-653 to emphasize the strong support for separation and protection mechanisms, see Figure 8. This technique thus makes it impossible for system events in one partition of the operating system (OS) to interfere with events in another; it can be compared to modern OS which provides security through Virtual Machine (VM) brick-wall partitions. It is called virtual because it seems as if each partition were its own separate computer.



- High degree of modularity
- A failure stays within one application
- Enabling different DAL/criticality level
- Separated functions and documentation
- Supports parallel development
- Strong separation of s/w from h/w
- Enables third parties separated delivery of applications

Example: Applications Loc (Localization) and LGC (Landing Gear Control) are separated through ARINC-653

Figure 8. Brick-wall-partitioning

Further refinement of the architecture framework is Distributed Integrated Modular Avionics (DIMA), meaning that several distributed computational units may host applications in an even more flexible manner. The DIMA technique will not be further discussed here.

2.3.2 Data bus standards

The MIL-STD-1553B, as mentioned above, is a dominating standard in the military avionics business. Its key characteristics are:

- Configuration: bidirectional with centralized bus controller (BC)
- Bit rate: 1 Mb/s
- Number of remote terminals (RT): 31

MIL-STD-1773B is an equivalent implementation for fiber optics providing greater immunity to high-intensity radiated electromagnetic fields (HIRF).

In the commercial avionics business ARINC-429 has proven to be one of the most popular bus standards. It employs a unidirectional standard known as Mark 33 Digital Information Transfer System (DITS). ARINC-429 is a two-wire differential bus which can connect a single transmitter or source to one or several receivers or sinks. Two speeds are available; 12.5 kb/s and 100 kb/s.

ARINC 664 is based on IEEE 802.3 Ethernet and utilizes COTS (commercial off-the-shelf) hardware thereby reducing costs and development time. Built upon ARINC 664 is AFDX (Avionics Full-Duplex Switched Ethernet) which is a deterministic bus for safety critical applications. AFDX was developed by Airbus for the A380 and provides guaranteed bandwidth. It utilizes a star topology network of up to 24 end systems tied to a switch, where each switch can be bridged together to other switches on the network. By utilizing this kind of structure, AFDX is able to reduce wiring and minimize aircraft weight.

Several other data bus standards are specified for different purposes; many are defined as ARINC standards, but they will not be further documented herein.

2.3.3 Aircraft systems

In parallel to the core avionics system, design and development of other aircraft systems, such as the fuel system, power supply system, or mission computing system, is done in close interaction. The avionics system is responsible to provide, for example, communication/control means, navigation data, and reliable computational resources to the other systems.

According to [Moir 2004], aircraft systems can be categorized into the following groups:

- Airframe/Structure (e.g. fuselage, wings, and aerodynamics)
- Vehicle systems (e.g. fuel, propulsion, and flight control)
- Avionic systems (e.g. navigation, controls & displays, and communication)
- Mission systems (e.g. weapons, data links, and mission computing)

For many engineers, the design and development activity for a specific subsystem takes place at a single domain level. The systems map onto the knowledge domains in which many engineers are educated or into which they develop their careers. There is an increasingly need to consider integration issues, for example, the engineers who design the display system will need some knowledge of the entire weapons system, its driving requirements, life cycle considerations, and configuration management strategies for the product (or even the product family).

For an introduction and brief overview of the characteristics of different types of aircraft systems, including the design and development aspects, see [Moir 2004]. Every aircraft system has its own specific technology concerns, even though the design methodology may be similar. An example of specific design methodologies and considerations related to fuel system development is given in [Gavel 2007].

3
Model Based Development

MODELS OF SYSTEMS and products tend to be of greater value to developing and manufacturing companies as more and more information is kept within models. Models may be of many different kinds, from cost estimation to spare part logistics. The main focus is this thesis is on system models representing a system (e.g. an aircraft's fuel or navigation system), that is composed of hardware, software and, when applicable, human interaction, as shown in Figure 9. A complete aircraft model is in turn composed of several such system models, but is still a system model, even though at a higher complexity level. At an abstract level these system models define the names and relations among parts: these are collectively called system architecture or structure. When details of functions, flows and physical equations are added, the model can be used to predict performance and dynamic behavior; it becomes a simulation model.

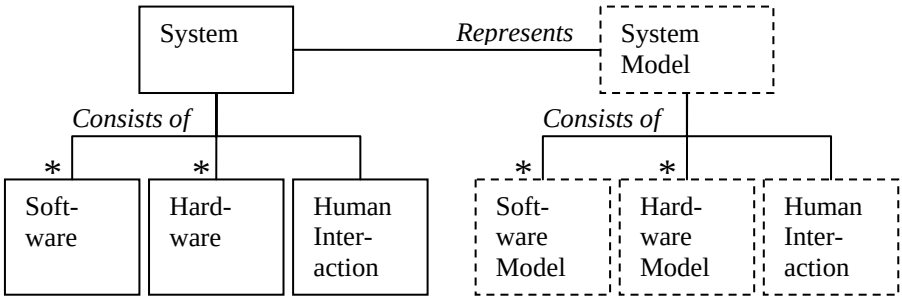


Figure 9. A system model represents the system

Other kinds of models are also briefly treated, such as cost models and meta-models (special focus in Paper [III]).

There are several approaches to model based development depending on application domain and complexity of the product, but also depending on historical and organizational factors. Some of those approaches, with relevance for aircraft and avionics development, are covered below.

3.1 Introduction to the modeling approach

Models in different forms have always been used in engineering. There are several definitions of what a model is and the concept of 'model' can be defined as:

> *A model is a simplified representation of a real or imagined system that brings out the essential nature of this system with respect to one or more explicit purpose.*
>
> [Larses 2005]

Models are used implicitly in the mindset of the engineer, in terms of construction of physical models/prototypes, in terms of symbolic models such as mathematics or written text, and with the introduction of computers, through the use of CAE tools. Block diagrams without well defined semantics are an example of symbolic models. Formal symbolic models, which might also be called mathematical or analytical models, have proven to be very important tools for clarifying and solving engineering problems.

There are many definitions of model based systems engineering (MBSE), model based development (MBD) and how they relate, but in this thesis these refer to:

MBSE: *The engineering approach which uses a central model to capture requirements, architecture and design, for support of the systems engineering activities.*

MBD: *Development based on abstract representations with predefined semantics and syntax, supported by engineering tools.*

Relation: *The principle of MBSE is used to perform MBD.*

Using the model(s) as the central/single source of information there are three "utilization points" in which the chosen modeling technique(s) must fit in order to gain value in a model based approach, as illustrated in Figure 10. With the model M in the center, it utilizes the engineering activities;

- Create understanding. Through a clear structure, common graphical notation or mathematical expressions, understanding of the engineering problem/solution is improved.
- Communication within an engineering team is enabled, also based on common notation or language, but also on visual representation of the model.
- Data. With the introduction of new engineering tools, meta-models, databases and analysis techniques, more information may be stored and it is nowadays

possible to analyze more aspects of a system/product on relatively simple computers.

In a harmonized modeling approach, support for all those three points must exist; a chain is only as strong as its weakest link.

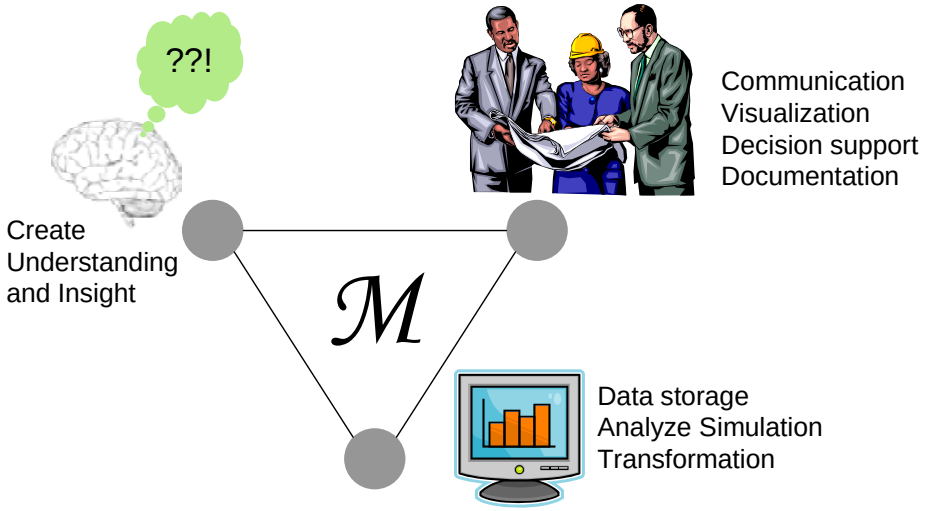


Figure 10. The three utilization points of modeling: understanding, communication and data

Many of the emerging techniques for model based development focus on a third utilization point (data storage), but are weak in the other two. One example is the many UML tools with automatic code generating add-ons introduced onto the market recent years. In most cases the focus is solely on the tool, its capabilities and features, but a defined and scalable method that supports *understanding* and *communication* is sometimes missing.

Other terms besides MBD and MBSE with similar meaning are Model Based Design (MBD), Model Driven System Design (MDSD) or simply Model Based Engineering (MBE), where MBSE is wider because it explicitly spans the whole life-cycle.

Definition of terms

Basic definitions of terms used in this thesis (if not otherwise stated) are based on aerospace standards [ARP 4754, 1996] and [ARINC 653, 2006] and are also a basis for aerospace engineering whether it is model based or not. Examples of such terms found in these standards and used in this thesis, but not explicitly defined, are; requirement, validation, implementation, integration, safety, and specification. Here are some terms with explicit definitions:

Complexity	An attribute of systems or items, which makes their operation difficult to comprehend. Increased system complexity is often caused by such items as sophisticated components and multiple interrelationships.
Criticality	Indication of the hazard level associated with a function, hardware or software, considering abnormal behavior alone, or in combination with external events.
Design Assurance Level (DAL)[3]	The level of rigor of tasks performed to items(s) in the process. The DAL is used to identify the RTCA/DO-254 and DO-178B objectives that need to be satisfied for an item.
Partitioning	The mechanism used to separate portions of a system or an item with sufficient independence such that a specific development assurance level can be substantiated within the partitioned portion.

Some other terms used in several places in this thesis are defined below.

Meta-model
A meta-model is simply a model of models (that are similar to each other), and defines what modeling elements (classes), properties (attributes) and connections (relations) there exist in a specific modeling framework or technique. An example of a meta-model can be found as an attachment to Paper [III].

Mid-scale simulation
The activity performed, when some simulation models of aircraft subsystems, developed with different modeling techniques, are integrated into a larger model, complex enough to not be simulatable in a desktop modeling and simulation tool. This kind of simulation activity is described in Papers [I] and [II].

Large-scale simulation
When several simulation models of the aircraft subsystems are integrated and specific arrangements for performance or interoperability exist, the simulation is considered to be large-scale. Examples of such arrangements are real-time execution, “pilot in-the-loop simulation” or “hardware in-the-loop simulation” (HILS) configurations.

3.2 Systems Engineering

In this thesis, Systems Engineering (SE) is interpreted as the engineering activities that are general regardless of technical discipline. It includes integration of the engineering and project management interface, but also integrates work in the different technical disciplines, as illustrated in Figure 11.

This understanding of SE in mainly based on INCOSE definitions and the INCOSE Systems Engineering Handbook, [INCOSE 2007].

[3] See also ”Definition of Design assurance Level” at page 27

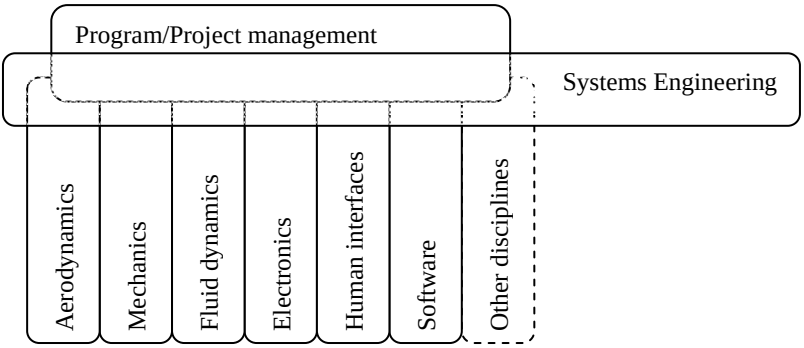


Figure 11. Systems engineering in relation to other engineering/management disciplines.

Activities included in SE are typically;

- Specification & Requirements Management
- Product breakdown & architecture
- Management of engineering budgets; Weight, Power, Cooling…
- Modeling, Simulation & Optimization techniques
- Information handling & control
    - Interfaces
    - Changes
    - Configuration
    - Traceability
- Risk Status & Control
- Subcontractor Management
- "-ilities"
    - Safety
    - Availability & Reliability
    - Maintainability
    - Reusability
    - Security
- Performance of SE activities in interaction with technical domains
    - Education & training
    - Decision logging & communication
    - Design Reviews
- Planning; Writing the Systems Engineering Management Plan

The planning of engineering methods/activities is most important in a project's start-up phase, but has to be ongoing throughout the project as it includes activities in a product life-cycle perspective, which are not all possible to set at an early point. Here a just-in-time approach is preferable; decisions, descriptions, and education in each respective activity/practice are done just ahead of when it is required in the project. More about planning and experiences of implementation of MBSE into a real project is found in Paper [V].

3.3 Development process models and standards

For industrial activities a range of defined models/processes exist for developing complex products. Models used in aerospace are adapted from those and instantiated with specific needs. As the aerospace area is a wide one, further other useful standards, for example avionics, are adopted from the electronics or communication areas. Interchange of knowledge and standards between automotive and aerospace in the area of development methods for embedded systems is also ongoing. This section introduces some definitions to support development of complex products, especially relevant for embedded/avionics systems.

3.3.1 Product and System lifecycle

A widely used systems/software development model is the two dimensional model with system lifecycle phases versus process activities with visualization according to RUP[4]. Here the process activities are adapted from EIC/ISO 15288, see Figure 12.

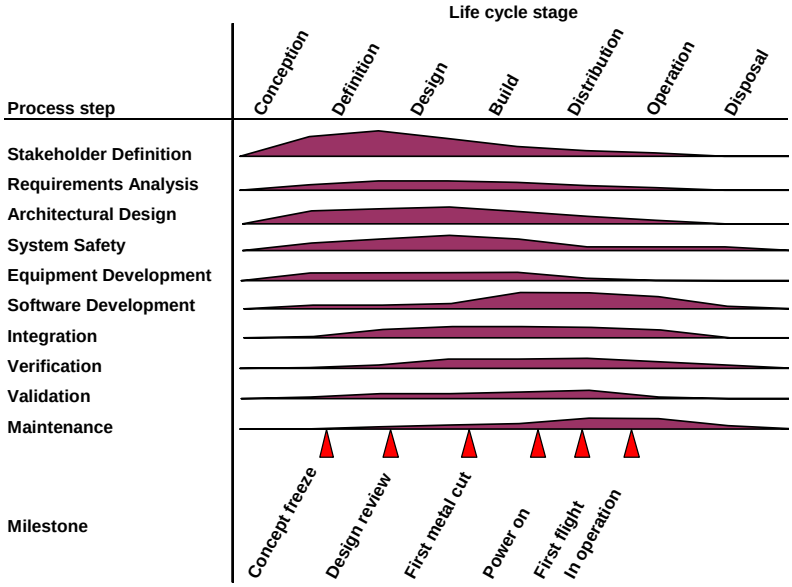


Figure 12. Process activities during product lifecycle

Due to the multi-customer scenario and product strategy as described in section 2.2, the traditional product lifecycle model is enhanced with a system lifecycle definition including the “system-phases” of a whole product family seen from a development point of view. This definition may serve as a template when designing or changing the

[4] Rational Unified Process by IBM

engineering environment (selection of methods and tools). Each system-phase requires different capabilities and performance of the engineering environment.

Table 1. Definition of system-phases for a product family

System phase	Conception	Core development	New variants	Enhancements	Maintenance
Main work	User needs elicitation, Explore design alternatives, Trade off study Optimization	Definition, specification, design, implementation and initial production	Variants specification, design and implementation. Production adaption	Rework of system, integration of new functions/features Obsolescence management	Maintenance and support of system. Corrections. User feedback handling
Main objective	Defined optimal layout	First product release	Defined product family	Keep product competitive	Keep customers satisfied

The system lifecycle phases in Table 1 are used in this work to analyze the long-term effects of different choices of development method and its supporting engineering environment.

3.3.2 Concept generation and selection

Models play a central role in the concept phase of a project, simply because they involve the study of technologies and components that may still not exist, so experiments on a physical product are impossible.

Looking more deeply into the conceptual work, the aim is to determine the technical principle. According to [Ulrich 2000], the conceptual phase itself may be divided into two different activities: concept generation and concept selection. In the early stages of the conceptual phase, concept proposals are easily dismissed without deeper analysis. After selection of a smaller set of promising concepts, a new iteration for concept generation is performed, both by combining other concepts and by “inventing” new ones based on ideas and experience gained from earlier iterations, see Figure 13. As the number of concepts decreases, the need for deeper analysis with more sophisticated (and expensive) methods will increase.

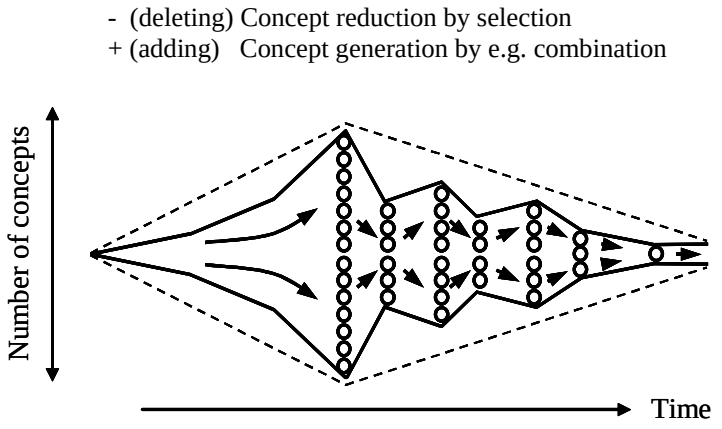


Figure 13. Concept generation and selection model

The value of more advanced modeling this early in the design process is questionable, though, due to the large degree of uncertainty. A combination of simple spreadsheet tools and uncomplicated modeling is normally sufficient early in the conceptual phase when concepts are still relatively numerous. Before the final selection is made, the information package as a base for the selection/decision needs to be complemented with more refined analysis such as dynamic modeling, performance calculations, safety analysis, or in-depth cost assessment.

When large sets of data are to be handled during concept creation and selection, structuring of information in a Function/Means tree is convenient. This is described in section 4.4.2 on page 59 and more details can be found in Paper [III].

3.3.3 Development and modeling standards

In this section, guiding standards with major influence on avionics development are briefly described. Requirements stated in these standards are intended to be applied by policies and procedures that define the requirements for project implementation (or instantiation) of the documented enterprise processes.

ISO 15288

ISO 15288 is a process framework covering Systems Engineering standard processes and life cycle stages and defines the processes, divided into four categories:

- technical processes
- project processes
- agreement processes
- enterprise processes

Each description contains purpose, activities and outcomes. Life cycle stages described are concept, development, production, utilization, support, and retirement.

ISO 15288 is related to other systems engineering descriptions including the INCOSE handbook, ISO 12207, and CMMI.

SS EN-9100

EN-9100 is based on ISO 9100:2000 and instantiated for the aerospace industry. The standard considers for example organizational issues of the developing company/project and requires processes in the areas of:

- planning and control of the product development activities
- responsibility and authority for design and development
- identification of mandatory steps and methods of configuration control
- review of input data to ensure consistency with requirements
- appropriate review, verification, and validation at each development stage
- structuring the design effort into significant elements
- interface management between different groups
- "design tasks to be carried out shall be defined according to specified safety or functional objectives of the product in accordance with customer and/or regulatory authority requirements"

The last (process) requirement in the list says, as in several standards, that there must be explicitly stated (product) requirements as a basis for:

1) Formulating technical goals in order to execute the project
2) Verifying the product against for airworthiness/certification purpose

Separation of these two kinds of product requirements is important, in the light of experience at Saab Aerosystems, because they have different purposes, but there is no good support for how to actually handle the separation in standards.

ANSI/EIA-632

The [ANSI/EIA-632 1999] standard "Processes for Engineering a System" from American National Standards Institute defines an approach to engineering (or re-engineering) a system, incorporating industry best practices. The approach has three parts:

a) A system is one or more end products and sets of related enabling products that allow end products, over their life cycle of use, to meet stakeholder needs and expectations;

b) Products are an integrated composite of hierarchical elements, integrated to meet the defined stakeholder requirements;

c) The engineering of a system and its related products is accomplished by applying a set of processes to each element of the system hierarchy by a multidisciplinary team of people having the needed knowledge and skills.

A system consists generally of a product breakdown and specification structure as described in Figure 14.

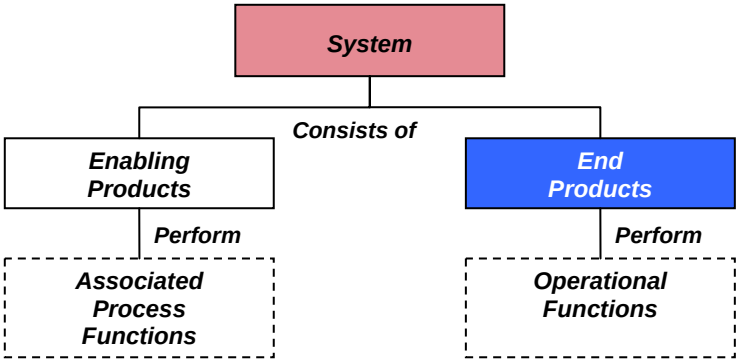


Figure 14. ANSI/EIA-632 definition of Enabling and End products

Each product is broken down into sub-systems in a hierarchical manner, producing the system-of-systems view shown in Figure 15. This explicitly means that each system at every level has its own set of enabling products, which in the model based case include the actual models of the end product(s).

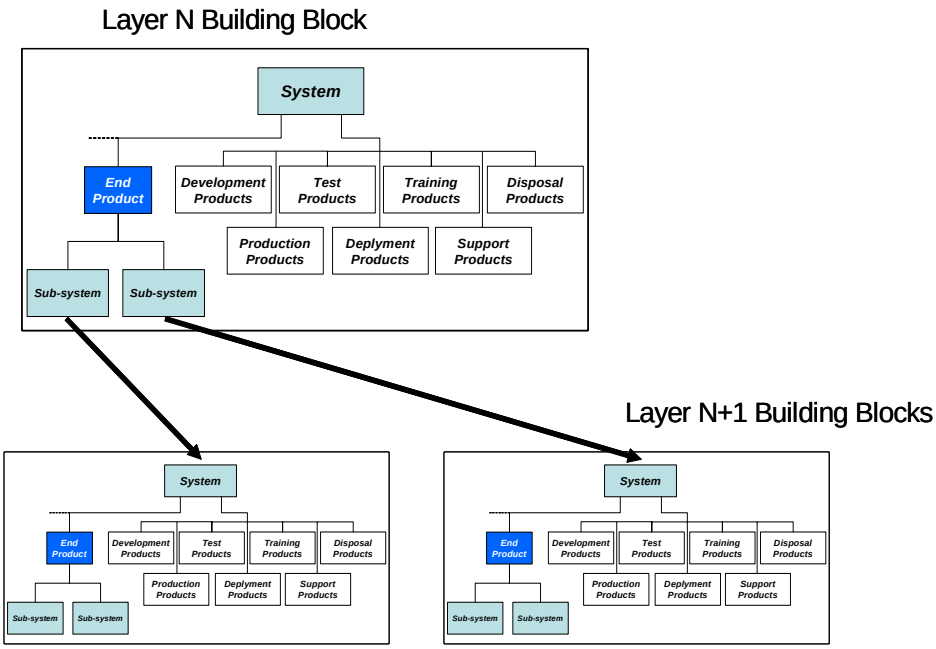


Figure 15. Building blocks in layers according to EIA-632

The EIA-632 standard clearly distinguishes between “acquirer requirements” and “other stakeholder requirements”. Sources of other stakeholder requirements include, for example, government and industry regulations, international conventions, environmental constraints and company directives. In general, other stakeholder requirements place constraints on the system development, both on the resulting product and the processes for developing it. It is usually impossible to meet all requirements for a particular system since they are conflicting relative to one another, so early and thorough requirements analysis is crucial, preferably by means of modeling (and simulation when appropriate).

RTCA

Guiding standards and recommendations from Radio Technical Commission for Aeronautics ([RTCA 2009], which is a private non-profit organization for standardization) have strong influence on development of avionics in civil aviation. With increasing influence also in the military sector are the RTCA/DO-xx series, where worth to be mentioned are;

DO-178B, “Software Considerations in Airborne Systems and Equipment Certification” is widely referenced and regarded as “the bible” within the air-borne software community. It helps developers, in a structured way, to be confident and show that the software aspects comply with airworthiness requirements [RTCA, 1992]. The definition of a new version, DO-178C, is ongoing and aims to take emerging software development techniques and trends, such as model based methods, object oriented methods and formal methods into consideration.

DO-254, “Design Assurance Guidance for Airborne Electronic Hardware” is providing help to aircraft developers and suppliers of aircraft electronics to assure and show that equipment safely performs its intended functions. [RTCA, 2000]

DO-297, “Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations” is focused on IMA-specific aspects of safety and design assurance. [RTCA, 2005]

Aerospace Recommended Practice (ARP)

Aerospace Recommended Practice, APR 4754, “Certification Considerations for Highly-Integrated Or Complex Aircraft Systems” [ARP 4754 1996] addresses development of aircraft and systems that implement aircraft functions. It is partly aligned with software aspects in DO-178B, and hardware aspects in DO-254. It does not include specific details of software or electronic hardware development.

Definition of Design Assurance Level

Design Assurance Level (DAL) or criticality level is one of the fundamental definitions for development of software for modern aircraft. The definition is used in RTCA as well as in ARP as a foundation for several development and certification activities. It

gives implications on the choice of workflow, tools and modeling techniques. Criticality is defined in five levels according to the likely consequences of a system failure, as shown in Table 2.

Table 2. Definition of Assurance Levels

Level	System failure	Failure description	Probability description	Likelihood of failure (per flight hour)
A	Catastrophic failure	Aircraft loss and/or fatalities	Extremely improbable	Less than 10^{-9}
B	Hazardous/ severe major failure	Flight crew can not perform their tasks; serious or fatal injuries to some occupants	Extremely remote	Between 10^{-7} and 10^{-9}
C	Major failure	Workload impairs flight crew efficiency; occupant discomfort including injuries	Remote	Between 10^{-5} and 10^{-7}
D	Minor failure	Workload within flight crew capabilities; some inconvenience to occupants	Probable	Greater than 10^{-5}
E	No effect	No effect	Not applicable	

Examples of functions with different DAL:

Critical: - Display of speed information

- Sensing and calculation of remaining fuel quantity

Non-critical: - Recording of operational data for tactical evaluation (military)

- Cabin entertainment functions (civil aviation)

To comply with the objectives according to a higher level is of course more costly than to a lower one. How big the differences in increased development costs actually are is a debatable issue and depends on methodology, tools, and skills. Figures in the range of 75% to 200% are used for a level A/B system compared to a level D/E system. [Hilderman 2007] states that if ARP and RTCA are implemented efficiently, the initial increased avionics development cost is much less and will probably be cost-effective over the long-term because of improved reusability, fewer bug fixes, less down-time, and increased user/market acceptance. Reuse of software components is the major benefit/saving in aviation development according to [Matharu 2006].

Model Driven Architecture

The Model Driven Architecture (MDA) approach, as described in [Mellor 2002], is a way to support separation of functional specification from implementation. MDA is used in the development of software intensive systems where automatic code generation is part of the process. Its underlying concept is to separate ‘do the right thing’ from ‘do

the thing right' by introducing platform-independent models (PIMs) and platform-specific models (PSMs). Translation from PIM to PSM is defined by rules in a platform definition model (PDM) and this translation is generally performed by automated tools. Translation (or generation) from models to different source code languages, such as ADA, C++ or Java is used, but also translation to hardware (firmware) specification languages, e.g. VHDL is emerging in the field of hardware/software co-design [Rieder 2006].

A specific UML subset or profile[5], Executable UML, "xUML"[6], is proposed for standardization to [OMG 2005] to support the MDA approach. Executable UML is the evolution of the Shlaer-Mellor method to UML. As reference to the Shlaer-Mellor method, see [Shlaer 1988] and [Starr 1996].

3.3.4 V-model

Several different types of models can be used to describe the process of product development. The V-model is a popular way to illustrate development of a whole system, see Figure 16.

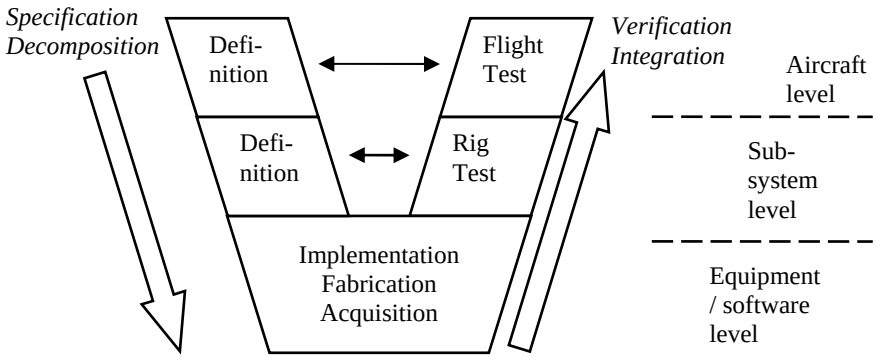


Figure 16. A simple illustration of the V-model

On the left side, definition, specification and modeling activities are performed, mainly without any real parts of the product available. On the right side integration and test activities, with real parts/articles are included. The V-model tends to be rather top-down oriented, depending on the interpretation.

3.3.5 Iterative and incremental methods

Developing and delivering a larger system in increments is a way of reducing risk. Incremental methods for model driven software development have a history dating back

[5] A UML profile is a standardized set of extensions and constraints that through a generic extension mechanism tailor UML for a particular use.
[6] Two abbreviations exist; xUML and also xtUML as in 'executable & translatable'

to the early 1990s. The core of the software models used for generating code was object models documented with class diagrams, use-cases and finite state machines.

Since then a number of software development methods have appeared, ranging from the waterfall method to highly incremental ones like the extreme programming (XP) method [Beck 2000]. A common feature of modern incremental methods is that they foster some general values for a successful "culture", further enforced by tool and library supported best practices.

Extreme Programming (XP)

Extreme programming was formulated by Kent Beck, as a consistent set of values that serve both human and commercial needs; communication, simplicity, feedback and courage. These values are expressed as practices.

1. The Planning Game – Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.
2. Small releases – Put a simple system into production quickly, and then release new versions on a very short cycle.
3. Methaphor – Guide all development with a simple shared story how the whole system works.
4. Simple design – The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.
5. Testing – Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.
6. Refactoring - Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
7. Pair Programming – All production code is written with two programmers at one machine.
8. Collective ownership – Anyone can change any code anywhere in the system at any time.
9. Continuous integration – Integrate and build the systems many times a day, every time a task is completed.
10. Work no more than 40 hours a week as a rule. Never work overtime two weeks in a row.
11. On-site customer – Include real, live users, available full-time to answer questions.
12. Coding standards – Programmers write all code in accordance with rules emphasizing communication through the code.

Telelogic Harmony

Harmony [Douglass 2007] is a UML/SysML oriented incremental development method or process, which can be customized for system types like embedded software development. It has 15 best practices, similar to those of XP, but is targeted towards larger projects with well documented team member roles, competences and task descriptions for roles. Examples of best practices are: 12 – Use Frameworks, 13 – Apply Patterns Intelligently, and 15 Manage Interfaces. Figure 17 shows an overview of the Harmony method.

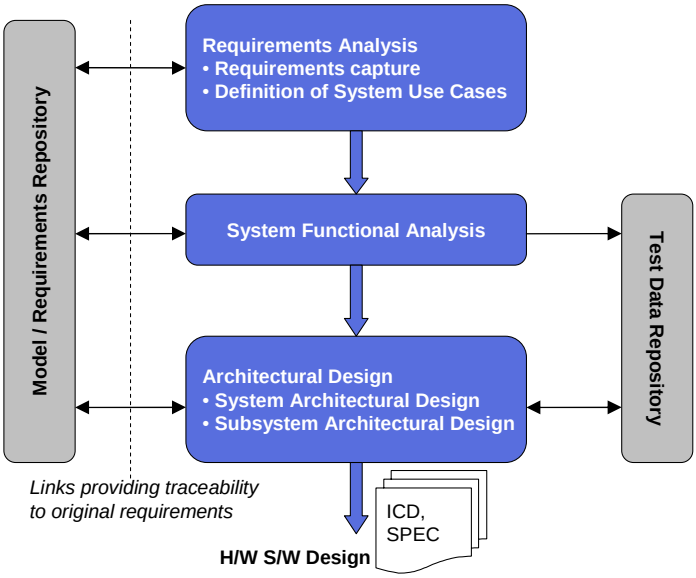


Figure 17. Simplified overview of the Harmony development method

The systems engineering component of Harmony (Harmony-SE) uses a "service request-driven" modeling approach with the Systems Modeling Language (SysML) notation. System structure is described by means of SysML structure diagrams using blocks as structure elements. Communication between blocks is based on messages (service requests). Provided services are in the receiving part of service requests and state/mode change or operations (activities) are described as operational contracts.

Harmony-SE has the following stated key objectives:

- Identify / derive required system functionality.
- Identify associated system states and modes.
- Allocate system functionality / modes to a physical architecture.

Functional decomposition is handled through decomposition of activity operational contracts.

3.4 Classification of Models and Modeling Domains

Ever since modeling became a practice for specification or problem solving in science and engineering, the number of available techniques and tools has increased more and more rapidly. This is partly caused by the evolution of work stations/computers, but also thanks to the demonstrated value of modeling in the area of complex problem. Naturally, every modeling technique fits best for one small set of "problems", even though it may be used for a broader set. In large development projects it comes to a choice or trade-off between on one hand the usage of many specialized, powerful tools, and on the other hand the use of a few multipurpose, but usually "dull" tools and techniques. Many attempts have been made to classify modeling techniques, and the classification made herein is mainly based on the problem areas and available (substitutable) tools.

3.4.1 Value of models

When choosing a model technique, it should demonstrably add sufficient value to the project and it is important to recognize what characteristics a technique has. This list, adopted from [NASA 1995], includes:

- Credibility in the eye of the decision maker
- Responsiveness
- Transparency
- User friendliness

These characteristics are crucial to the acceptance of a modeling technique for use by a team. Relevance is determined by how well a technique addresses the substantive cost-effectiveness issues in a trade-off study.

A history of successful predictions gives credibility to a model, but full validation (proof that the prediction a model gives represents reality), is very difficult to achieve since observational data is not always available or is of lower quality.

The responsiveness of a model is a measure of its ability to distinguish among the different alternatives being considered in e.g. a trade-off-study. A responsive avionics architecture cost model, for example, should be able to distinguish the costs associated with different system architectures or designs, operations concepts, or logistics strategies.

Another important characteristic is transparency, which means that the model's mathematical relationships, algorithms, parameters, supporting data, and inner workings (i.e. its meta-model) are open to the user. The value of this visibility is in the traceability of the models results. The results may not be accepted by everyone, but one knows anyway how they were derived. Transparency also helps models gain acceptance. It is easier for a model to be accepted when its documentation is complete and open for comment, which is facilitated by open languages, such as the [Modelica] language.

Proprietary models, on the other hand, are often difficult to gain acceptance for because of the lack of transparency. Examples of models that are documented, but that do not have source-code open to the users, are often seen in the case of component libraries for simulation tools from vendors with a proprietary policy, such as the [SimElectronics] toolbox from The MathWorks.

User friendliness towards end-users is about how the engineer can learn to use the modeling technique, prepare the inputs and interpret the outputs/results. User friendliness towards super-users is related to the effort needed to update, validate, administer and support a model, but also to aid end-users to interpret advanced analysis of the outputs/results. One may also note that friendliness for the administrator (support organization) play a role in the work of installation, upgrading, bug reporting and contact with vendors.

3.4.2 Specification and Analysis models

One classification is to divide models into "specification" and "analysis" models. An example from solid modeling and hardware/structure development is the following:

- A specification model is the definition of surfaces (shape) and the content (materials) of a component. It is typically done in a 3D CAD (Computer-aided design) tool, in a visual prototyping manner.
- A connected analysis model is used for stress analysis on the same component, based on the specification, but with information on boundary conditions (spectrum of forces) added.

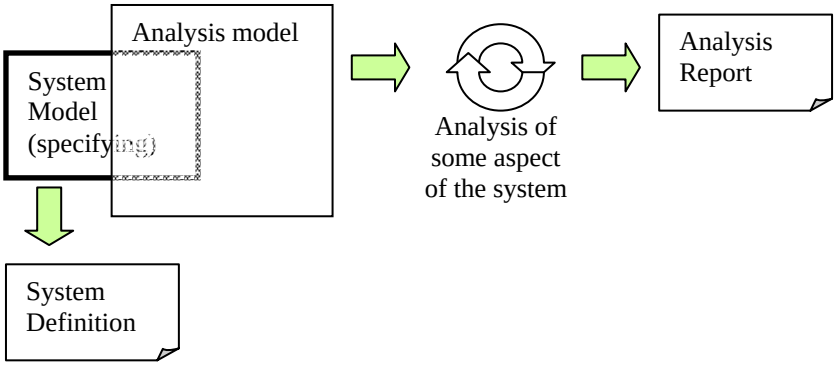


Figure 18. The specifying model is the basis for definition and for analyses of a system.

An analysis is performed with a subset of information from the specification model, but with additional information for the specific analysis to be performed, as shown in Figure 18. The same specification system model can consequently be the basis for performing analyses of several aspects of the system. Examples of analysis from avionics

design are fault tree analysis (TFA), formal methods analysis, and analysis by simulation.

3.4.3 Modeling domains

To divide and sort modeling tools and related techniques/methods into modeling domains was a means to analyze strengths and weaknesses of different modeling methods/tools and to organize the work in the MBSE change program at Saab Aerosystems (2006 – 2009). One purpose was to verify that the efforts were broad enough and that all domains were covered by appropriate investigations/studies and investments to evolve the organization further. Objectives were to achieve higher efficiency, quality, and an attractive engineering environment. The change program covered more than support only for avionics design and aircraft simulation, but the main contributions to this area are related to these modeling domains, so all the defined modeling domains are described here.

Overview

The modeling domains are defined as shown in Figure 19. These modeling domains are discussed in this section, but the main focus is on "Usage, Needs and Requirements" and "Architecture and interfaces". The other domains are briefly defined here, but they are related to more specific modeling techniques, which are covered in chapters 4 and 5.

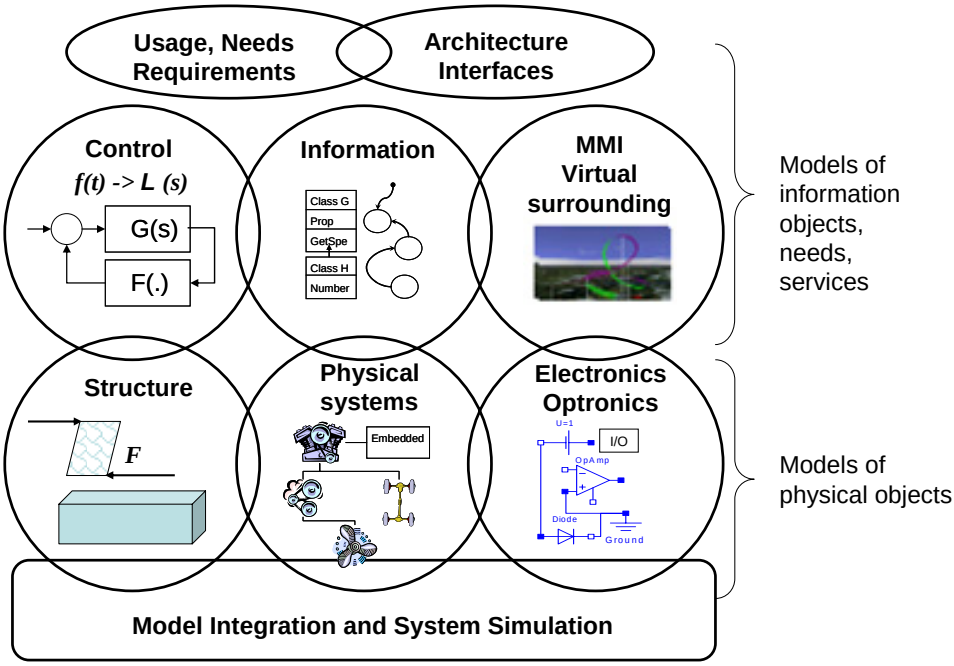


Figure 19. Definition of Modeling Domains

The lower part is related to physical objects and their properties, such as; space, time, energy and matter, whereas the upper part relates to information (and information about information).

Usage, Needs and Requirements

Design of a product with embedded electronics/software (e.g. avionics) is typically governed by requirements as regards its capabilities. The domain of usage, needs and requirements contains modeling means for specifying the context of the product at such a level of detail that a competitive product for that context can be engineered.

Functional requirements related to usage are preferably modeled by descriptive use-cases in UML or SysML. Discrete-event simulations on an abstract level are also valuable to show interaction with users, connected systems or the environment. Non-functional requirements are elicited with specialized processes and associated information is handled by a combination of means, such as databases, spreadsheets and plain documents. Examples of non-functional, technical aspects are;

- System safety
- Performance
- Reliability
- Availability

Examples of more non-technical aspects are;

- Cost
- Security
- Obsolescence handling
- Flexibility & Growth Potential
- Affordability & Risk

For an example of sources for technical requirements in Flight Control Design, see Figure 1 in Paper [1].

Cost modeling may be performed by several methods, where COCOMO and COSYSMO [Valerdi 2008] is widely used. Cost models based on the product breakdown is easy to set up and align with other development activities, but has to be kept on an aggregation level, in order not to grow too large. Another cost prediction method is the Lichtenberg Method [Lichtenberg 2000] involving an analysis group in a creative, multi-disciplinary process where qualitative and quantitative data are captured and modeled.

In addition, handling of assumptions (as complement to requirements) is a growing field of interest, supported by, for example, ARP 4754, and the emerging SPEEDS technique. See the SPEEDS section on page 55 for more details.

Architecture modeling

The architecture model of a system defines the system boundary, its internal parts (components or subsystems), the behavioral responsibility of each component, and interfaces (internal and external). To complete the picture, a top level model describes actors acting on the system but also the physical environment of the system to operate in. A simple example of a top-level architecture model in UML/SysML notation is shown in Figure 20. Different views are required to describe and analyze possible architectures, or to "explore the design space". In addition, good tool support is needed to enable modeling, analysis, design and document generation.

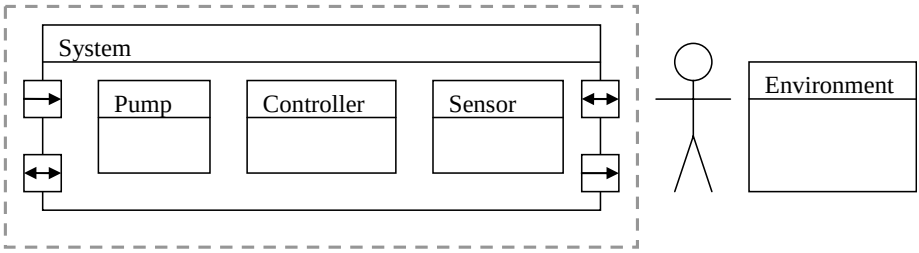


Figure 20. Example of a top-level architecture model in UML/SysML notation

The decomposition provides a modular framework allowing any refined version of a module to be plugged in without considering (or redo) the rest of the model.

For parts of a product (sub-systems) performing safety critical functions, visualization of the data-flow is crucial in order to be able to really understand the causality (dependencies) and perform design reviews in larger teams. Several books and research results in the area of architectural modeling for embedded systems exist, among which can be mentioned [Lavi 2004], [Muller 2004], [Larses 2005], and [Weilkiens 2008].

Control

The core of the control modeling domain is the closed loop view of the system. It also provides modeling methods to specify operation schemes of the product, its subsystems and components. The control model specifications range from high-level mission scenarios, through intermediate abstract control-theory models, down to executable code in the real-time operating computer hardware or control mechanisms implemented in mechanical, electrical, hydraulic and/or other hardware. Models' characteristics are causal, signal-flow oriented, and the dynamics is based on differential or difference equations, which enables transformations (e.g. Laplace or Z-transforms) between the time and frequency domains.

Information

The information domain contains modeling facilities for specifying the terminology, explicit meaning, naming and representational format of information for storage and communication between different co-operating actors in the avionics systems. An actor can be anything ranging from a human operator to software and hardware. Information modeling has a specific relevance in development through modeling, with the fact that everything managed within the kind of models treated here, is information. Specifying the semantic rules for a model is also a model; its meta-model.

Man Machine Interface & Virtual surrounding

The domain of Man-Machine Interface (MMI) and virtual surrounding contains modeling of display surfaces, real-time interaction and recorded history of behavior. It focuses on the interaction between human and machine, i.e. possibilities for the user (pilot) to get situation awareness in information-loaded scenarios, facilities for communicating the mission planning, and operating situation.

Structure

Basic information in the structure domain is spatial and material related. Usage of CAD models enables the designer to verify at an early stage that equipment can be install and connected to electrical contacts and cooling connections. The structure domain contains modeling facilities for specifying, dimensioning, calculating and analyzing the properties of primarily load-carrying hardware components in the avionics product.

Physical systems

The domain of physical systems contains modeling facilities for specifying, simulating and virtually interacting with the mechanical hardware and fluid/energy flows. In early phases, concept evaluation with relatively simple models is performed. In later stages, verification of the system through the model is the goal, even though final verification generally requires the end product configuration. Differential equations and especially differential-algebraic equations (DAEs), see definition in section 4.2.4, are powerful for modeling in this domain.

Electronics, Optronics

The domain of electronics and optronics contains modeling facilities for specifying, simulating, and virtually interacting with the electronics circuitry and optronic components of the avionics product and its environment. Analyzes during design, such as Fault Tree Analysis (FTA), and model based diagnostics/troubleshooting during maintenance are fundamental techniques within this domain.

Model integration and system simulation

Models are integrated for the purpose of verifying some aspects of the integrated system. One example is 3D digital mock-ups for installation and collision analysis used in the CAx based process. For software/system verification, mid- to large-scale simulation as reported in sections 5.3.3 and 6.2.2 fall in this domain.

Tool support

For every modeling domain there are a set of tools for modeling (defining or specifying the system), but also for analyzing some aspect of the modeled information. Table 5 in section 5.2.8 “Tools summary” on page 75 shows examples of tools used for modeling and analysis in the different modeling domains.

4
Modeling and analysis techniques

IN THIS CHAPTER various modeling techniques specifically applicable for the aerospace and avionics industries are presented and classified. Different properties of models, such as static/dynamic or discrete/continuous are defined. Analysis of the system through the model is penetrated, and in particular the widely used simulation technique, where the introduction of errors through fault injection during a simulation is described.

4.1 Introduction and classification of techniques

A modeling technique is based on some way to describe the phenomenon/system under consideration. The definitions, general names and properties of modeling elements and their relationships are denoted as the meta-model. Many attempts have been made to classify modeling techniques and tools in a classification tree. Those trees are dependent on the context for which the classification is to be used, so no general classification is meaningful here. An example of a classification tree from [Cassandras 1999] is shown below in the section on behavior modeling techniques.

One main classification is to separate structure models from behavior models. These two ways of viewing a system, structure and behavior, are the essential views of any system description. Behavior is *what* it does and structure is *how* it is built. The two views, with a mapping of behavior onto structure, form the system description. If the desired behavior is defined separately from a structure, then alternative structures can be readily identified and the desired behavior can be mapped onto each of them [Oliver 1997].

4.1.1 Architecture modeling techniques

Architectural modeling is also defined as “systems architecting” or “modeling of the system structure”. The architecting activity strives for consistency and balance from requirement to actual product. The number of people working in product creation may vary from a few to several hundred, and the level of concurrency is high. Engineers working on the development of a new product have knowledge of only a (small) subset of the information. Inconsistencies and local optimal (sub-optimal) solutions occur frequently. Architecting is one way to manage this natural degradation of the system quality. Pro-active, through clear and unambiguous requirements and system decomposition, and reactive, by following up the feedback from detailed design, implementation, and test.

During the creation of a product many design decisions are taken and quite often these decisions are made within the scope of a particular moment in time, which means that subsequent decisions may be contradictory. A simple but powerful aid for consistent architecture information is the data dictionary: a centralized repository of information about components, names, meanings, relationships to other components, history, and decisions made. Even if a data dictionary sounds simple, its implementation can be challenging in an environment of both new and legacy components, reuse of both civil and military parts, and with different computer networks for security reasons. Modeling and architecting of the data dictionary itself are therefore recommended.

The architecture model of a system contains at least boundaries, internal parts and interfaces (internal and external). Different views or level of transparency for a part or model may be defined as follows;

- Black-box: Name and interfaces are visible, but no information about internals. This view is used e.g. for specification and verification when a part only has to be treated from an “outside” point-of-view. Is preferably modeled with the SysML “Block Definition Diagram” (bdd).

- Gray-box: Black-box view plus internal part structure, the parts’ connectivity, and the states and parameters of the model (including parts). Is convenient to use to gain knowledge of the internal structure, but not every detail. May be modeled with SysML “Internal Block Diagram” (ibd). It can also be used for “semi-physical” models where equations based on natural laws and parametric black-box models are mixed.

- White-box: Gray-box view plus all system variables, behaviors etc. For a system specification or a system model, all information about the internals is visible. For safety critical software components, white-box specification and testing are required in order to verify that e.g. design rules are met.

- Extended: All information that exists, for example functionality for initiation, execution, state update and simulation enhancement. With this extended view, information related to the model or modeling technique itself is also visible.

Information elements for architecture or structure modeling are described below in section 4.2.1 “Modeling with objects and classes”.

4.1.2 Behavioral modeling techniques

In order to create a description of behavior, a number of modeling elements are required. The necessary set of semantic elements, as defined in [Oliver 1997], includes:

- functions, which accept inputs and transform them to outputs
- inputs and outputs, of various types, and
- control operators, which define the ordering of functions

One example of classification of behavioral models from [Cassandras 1999] that holds for Discrete Event Systems (DES) modeling techniques is shown in Figure 21. The lower part with bold text in the figure is included in the DES definition. Those are defined as event-driven, discrete-state, non-linear, time-invariant, dynamic models.

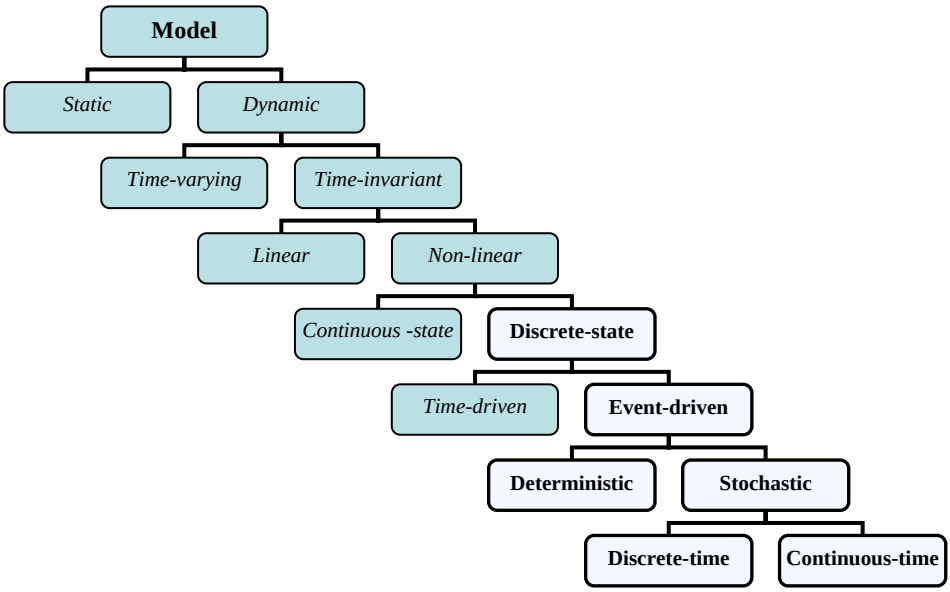


Figure 21. Model classification adapted from [Cassandras 1999]

Here follows a list of the essential properties of models.

Static or dynamic
When relations between the variables of a model are instantaneous, it is static. An ideal resistor is an example, because the electric current is directly proportional to the voltage (Ohm's law). There is no effect from earlier levels of the variables; no history dependency. In the dynamic model, the behavior depends on its history. A dynamic model is built out of differential and/or difference equations.

Time-invariant or time-varying
This classification relates to whether the rules of interaction for a model depend on time. A model is time invariant if the rules of interaction are independent of time. Otherwise, the model is time-varying.

Linear or nonlinear
A model is linear if there are only linear dependencies between input signals, states and output signals. A nonlinear model may be linearized in some point of operation, but then it is valid only in the neighborhood of that point. Linear models are in general easier to analyze.

Discrete-state or continuous-state
This relates to the range sets for model descriptive variables. In a discrete-state model, variables assume a discrete set of values. A discrete-event model is a continuous-time/discrete-state model. In a continuous-state model, variable ranges can be represented by the real numbers (or intervals thereof). A differential equation model is a continuous-time/continuous-state model.

Time-driven or event-driven
In a time-driven model, the time is a central modeling element, whereas in event-driven models, time might not even be defined. This makes integration of those two kinds of models challenging.

Deterministic or stochastic
This classification is based on the cause-effect relationships in the model. A model is deterministic if it only includes exact relations between variables; no random variables appear. A stochastic (probabilistic) model is described by stochastic variables or stochastic processes and contains at least one random variable.

Discrete-time or continuous-time
Models may be classified according to their time base. In a continuous-time model, time is specified to flow continuously - the model clock advances smoothly through the real numbers toward ever-increasing values. In a discrete-time model, time flows in jumps. The model clock advances periodically, jumping from one integer to the next (the integers represent multiples of some specified time unit).

Causal or non causal
Causality is the property of cause-effect. For information flow or models of sensors or a CPU, the system's inputs and outputs decide the causality. For physical systems with energy and mass flows the causality is a question of modeling techniques/tools. In non causal (or a-causal) models, the causality has not to by explicitly stated. There are simulation tools (for example [Dymola]) sorting the equations from a non causal model to simulation code. When creating a causal model, the modeler has to choose what is considered as a component's input and output, and the bond graph modeling technique is a method that aids in transformation from a non causal to a causal model. Bond graph is an energy-based graphical technique for building mathematical models of dynamic systems, see [Cellier 1995].

Synchronous or asynchronous
This refers to communication between sub-systems in a compound system, but can also define the properties of a larger model. Synchronous communication occurs at regular intervals. Asynchronous communication is used to describe communications in which data is transferred intermittently rather than in a steady stream. In simulation systems with a central solver, the models execute synchronously. With decentralized solvers asynchronous behavior is more relevant.

Open or closed
Another category relates the model to its environment. If the environment has no effect on the model (no input), the model is closed (or autonomous). An open or non-autonomous model has input variables whose values are not controlled by the model, but to which it must respond.

4.2 Basic modeling concepts

For every modeling tool or technique there is a modeling concept (or several, combined). The concepts are related to different underlying concept models, which may be founded in mathematics, object orientation or a cause-effect kind of approach. The basic concepts described below are just examples but are relevant for the application area of aircraft and avionics.

4.2.1 Modeling with objects and classes

The object oriented (OO) modeling approach has attracted considerable interest lasting recent decades. In analysis (OOA), design (OOD), and documentation of large systems, the approach is an attractive one as it supports, for example, encapsulation and modularity. The introduction of programming/modeling languages such as ADA 95, C++, Java, and UML supports the object-oriented programming (OOP) paradigm. [SysML], [Mellor 2002] and [Johansson 1996] are used as references for descriptions of the concepts.

Application of object oriented methods

In aircraft and avionics development, OO techniques may be used for several purposes. In this section the descriptions' main focus is on two fairly different application areas:

(i) Aircraft subsystem/software functional development, with dependence to real-time behavior and systems safety aspects.

(ii) Development of tools for Engineering Data Management (EDM) which is partly founded in information theory.

In both cases the identification and classification of objects are central, but they differ, for example, in the respect that (i) includes analysis of physical objects, while (ii) concerns mainly information objects. Both application areas rely on the basic concepts of OO where only the major concepts are presented here.

Class/Object

The concept of classes and objects is fundamental in humans' way of thinking. Philosophers like Plato worked a great deal with this concept. Quine is another philosopher who spent time on the class/object concept and the separation of physical and logical (or information) objects that is made here is partly based on [Quine, 1981]. The conceptual definitions of objects are:

- physical object (a thing); it is unique and exists in a unique place in the four-dimensional time-space
- logical object; an object is an instance of a class (and needs a unique identifier)

Now, definition of a class can be thought of somewhat differently within the two application domains. But the general class definition is "*a collection of objects sharing common attributes*". In SysML the, "block" is used as the same concept as class in other languages, but it has the name block to emphasize the different intended usage. Quotations in table 3 below show the definition of class/block from the references.

Table 3. Quoted definition of the class concept

Reference:	**Definition of Class/Block**
[Johansson 1996]	"A class specifies a classification of object types within the domain that are important enough to have their own name, definition and possibly to be accessed as one unit"
[Mellor 2002]	"A class is an abstraction from a set of conceptual entities in a domain so that all the conceptual entities in the set have the same characteristics, and they all are subject to and behave according to the same rules and policies."
[SysML] V 1.1	"Blocks are modular units of system description. Each block defines a collection of features to describe a system or other element of interest. These may include both structural and behavioral features, such as properties and operations, to represent the state of the system and behavior that the system may exhibit."

A comment on these definitions is that they differ slightly, and in the cases of Johansson and SysML it is significant that they are designed for their intended context or application area, Johansson for (ii) and SysML for (i).

Several names for class in the class concept exist with slightly different semantic meanings depending on the context;

- Category
- Kind of
- Type of
- Sort of
- Object group
- Set of

Even though several definitions of the class concept exists, the modeling activity is not dependent on these conceptual differences as long as there is a defined work-method for the specific modeling tool. The problems usually arise in the area of tool interoperability when transferring data and/or connecting tools.

Attribute

An attribute defines the object's characteristics or properties. It refers to the unique qualities of a specific object or to a class. What range and format an attribute value may have is restricted by its type specification. Examples of attributes of the **`Aircraft`** class are; **`Take_off_weight`**, **`Color`**, **`Maximum_speed`**, and so on. Typically, the value of an object's attributes can change over time. For scale-up and long term consistency in the MBSE context, clear definition/specification of the attributes is necessary, including unambiguous description, units of continuous attributes and discrete values of enumerated attributes.

Abstraction

Abstraction is about simplifying complex reality by making models appropriate to the problem, and working at the most useful abstraction level for a given aspect of the system. In large systems, the concept of abstraction is essential in order to "zoom" in at the right level and be able to focus and communicate about a problem/solution with just enough information. It happens too often that details not relevant to the current problem disturb an engineering discussion.

An example from software specification is implementation details about types and execution rate in a functional model description, as further described in Paper [IV]. Also "level of detail" is used as a similar mechanism, but this will not be further analyzed here. It should be noted that "abstract" could also be interpreted in the sense of "not enough insight" because of knowledge gaps in the specific engineering domain; what is abstract to me could be concrete to you.

Inheritance

Inheritance is a mechanism that enables reuse of general concept descriptions in the modeling domain, or as in the case of object-oriented programming languages, allows reuse of functionality. This provides a means to minimize duplications of definitions. Sub-classes are more specialized versions of a class, which inherit attributes and behaviors from their parent classes.

As a simple example from database design, the class **`Avionics_Equipment`** might have sub-classes called **`Computer`**, **`Sensor`**, and **`Recording_Unit`**. In this case, the specific **`Mission_Computer_1`** would be an instance of the **`Computer`** subclass. Suppose now that the **`Avionics_Equipment`** (super-) class defines some attributes called **`Number_Of_Ports`** and **`Cooling_Air_Required`**. Each of its sub-classes (**`Computer`**, **`Sensor`**, and **`Recording_Unit`**) will inherit those attributes, meaning that the database designer only needs to define them once.

Multiple inheritances are inheritances from more than one super-class, neither of them being a super-class of the other. This can make (ii) EDM development efficient, but is not supported in all systems/programming languages, especially not those designed for (i) aircraft subsystem/software functional development.

Relationships

A relationship is a semantic connection among model elements. There are three major types of relationships, part-of, association and generalization. In [UML] the relationship is defined “between two or more classifiers that specify connections among their instances”. Thus, one UML relation may have more than two “relation ends”, which is not conceptually necessary and adds extra complexity to the UML meta-model. Achieving a ternary (three-fold) relationship can instead simply be solved by replacing the ternary relation with a class connected with tree binary relations with appropriate cardinality constraints.

Example

To illustrate the Class/Object concept to the reader, here is a very near example. Consider a ***class***, this thesis with its ISBN[7] identifier. It represents the set of all copies (***objects***) of the thesis; what I am reading right now is a book belonging to the kind of thesis categorized by the ISBN code 978-91-7393-692-7. The ISBN is an identifier for the class of equal books.

Physical object – each hardcopy of the thesis exist at only one place in the four-dimensional time-space. To emphasize the meaning of individual and unique objects, the hardcopies are numbered, so this copy is identified by Physical Object No: .[8]

[7] International Standard Book Number (ISBN); the unique 9-digit book identifier.

[8] Note: If you are reading an electronic copy of the thesis, the individual numbering is of course not available, and this illustrates the difference that also appears in production of software and hardware artifacts in industry. Software is manageable to copy, but not to uniquely identify, which gives implica-

Logical object – electronic copies on computers or servers can be regarded as logical (or information) objects. These are not identifiable as physical objects. Information is easy to create, move, duplicate and destroy/delete.

4.2.2 Signal flow models

Signal flow modeling is well suited to be done in a graphical editor. In practice it is drag and drop editing from a library with predefined blocks and connections with arrows representing the signals, as illustrated in Figure 22. The library is structured by functionality group: for example, all filters are grouped together. When using a specific block in the model, it is instantiated by choosing parameter values. Besides a signal name, every signal may also be assigned attributes such as unit and a description.

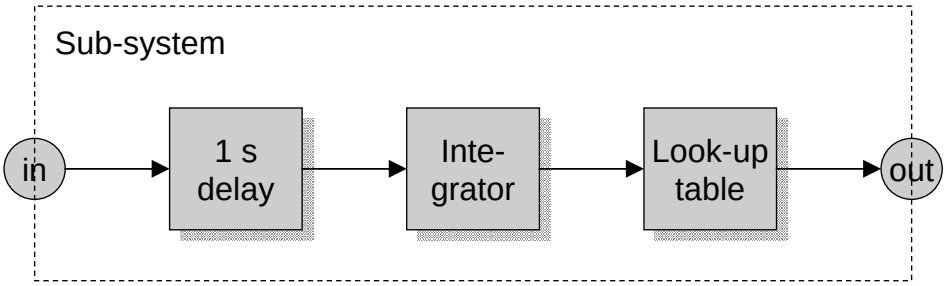


Figure 22. Simple example of a signal flow model

It is possible to define aggregate models with inputs and outputs and to reuse such composite models in any number of hierarchical levels. Connections represent either scalar or vector signals with fixed causality.

4.2.3 Discrete Event models

The Discrete Event System (DES) technique is a large modeling area, with several research groups and new tools emerging for large-scale (or industrial) usage. Here some main elements for DES modeling are defined, and modeling techniques with somewhat different semantics are described.

Basic definitions

Event: An occurrence without extension in time; it is instantaneous. Explicit events are named or predefined, while implicit events are those that occur but are not explicitly defined. An example of an implicit event is the time instant when a condition becomes fulfilled, see Figure 23.

tions to e.g. the differences in PDM (Product Data Management) and SCM (Software Configuration Management) systems.

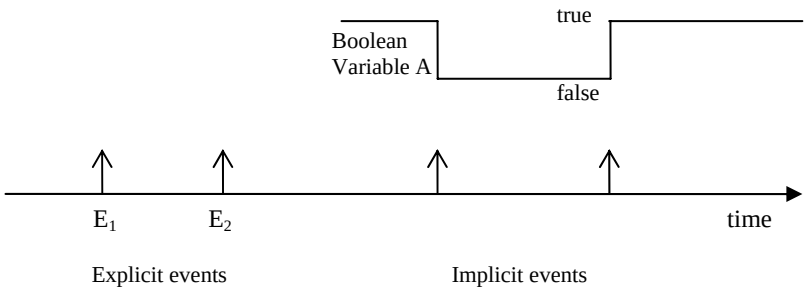


Figure 23. Illustration of explicit and implicit events

Entity: An object that passes through the system, such as messages on a data bus. Often an event (e.g. arrival) is associated with an entity (e.g. message).

Invariant: A property of a system/model that always remains true throughout all operational modes.

Mode: A particular operational condition of a system. Examples: operational phases of the aircraft (take off, cruise flight, landing) and system status (normal / degraded operation).

Predicate: A function that returns true or false.

State: A particular operational condition of a system. A common method of representing the operational functioning of a system is to enumerate all of the possible system states and transitions between them. Representation of the rules for changing states in a particular system can for example be made in statecharts, finite-state machines or Petri nets. Events typically change the state of the system.

Queue: A queue can be a task list, a buffer of messages waiting to be sent, or any place where entities are waiting for something to happen for any reason.

Scheduling: The act of assigning a new future event to an existing entity.

State diagrams, State machines and Statecharts

A state diagram is used to graphically represent the behavior of a system, which is composed of a finite number of states. There are many forms of state diagrams, which differ slightly and have different semantics. A state diagram represents the formal underlying mechanism called finite state machine (FSM). Another possible representation is the state transition table.

There are two kinds of state machines, Moore and Mealy machines, with an important difference:

Moore state machine: The output is a function only of the state of the machine. Consequence; the outputs are always valid except during transitions.

Mealy state machine: The output is a function of the state of the machine and of the inputs. Consequence; the outputs are valid only immediately after a transition.

A Harel statechart is an extension of the simple state diagrams, devised by David Harel. The extensions made are super-states, concurrent states, and activities as part of a state and the informal definition of statecharts is:

> *Statecharts constitute a* ***visual formalism*** *for describing states and transitions in a modular fashion, enabling clustering, orthogonality (i.e., concurrency) and refinement, and encouraging 'zoom' capabilities for moving easily back and forth between levels of abstraction.* [Harel 1987]

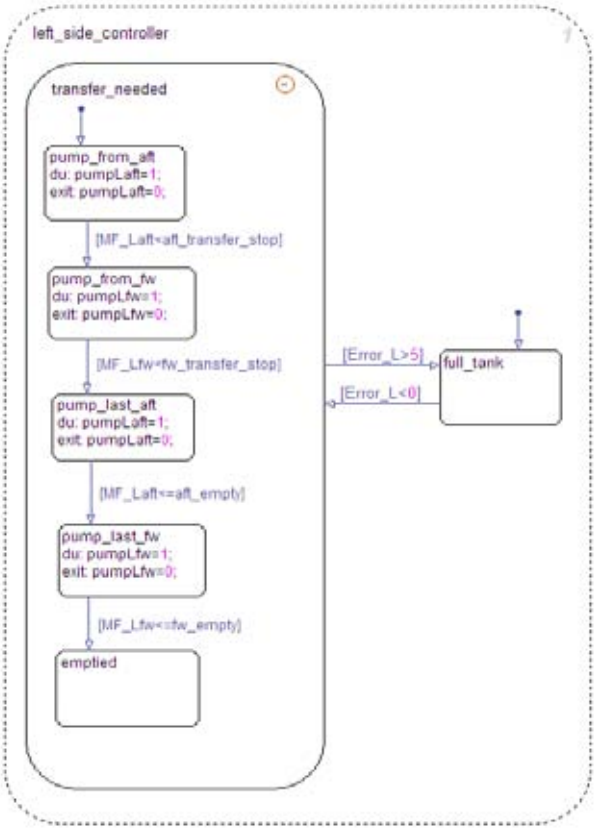


Figure 24. A state diagram of the fuel control model in Stateflow, from Paper [II].

In UML and SysML, the "State Machine Diagram" is used to model state oriented behavior, even thought the notation itself is not strictly formalized.

In tools like [Simulink]/[Stateflow], [SCADE] or [SystemBuild], support for statecharts is fundamental, even though there are limitations and restrictions depending on the tool and how the model is intended to be used. For example, for code generation purposes the limitations enable the systems engineer to model relatively freely within the limits without bothering too much about the production code layout or safety/quality aspects. A state diagram of a fuel control model in Stateflow is shown in Figure 24.

Automata theory

An automaton is the mathematical formulation of finite state machines. It is used to mathematically express, execute and analyze state machines, see chapter 2 of [Cassandras 1999]. Given a set of events as input, it "jumps" through a series of states according to a transition function, (which can also be expressed as a table).

Mathematically a finite state automaton (*M*) is a quintuple (5-tuple) $M=(S, E, T, s_0, F)$ where:

- *S* is a finite set of states
- *E* is a finite set of (input) events
- *T* is a transition function that assigns a state to each state/input pair
- s_0 is an element of *S* called the initial or start state (pump_from_aft in the example above)
- *F* is a subset of *S* called the final states (emptied in the example above)
- *M* is the transition function (Machine)

There are extensions and powerful model variants that build on the basic principles of automata, such as timed and hybrid automata and their "cousin" the Petri net.

Petri net

A Petri net, with reference to chapter 4 of [Cassandras 1999], consists of places, P, transitions, T, and arcs with a given direction. Marked Petri nets have tokens (graphically modeled as dots) assigned to places, as shown in the simple Petri net example in Figure 25. A token is "put in a place" essentially to indicate that the condition described by that place is satisfied. Places may contain one or a set of tokens, and the number of tokens at a place is changed by "firing" the Petri net. The distribution of tokens over the places is called a marking. Arcs run only between places and transitions. The places from which an arc runs to a transition are called the input of the transition; the places to which arcs run from a transition are called the output of the transition.

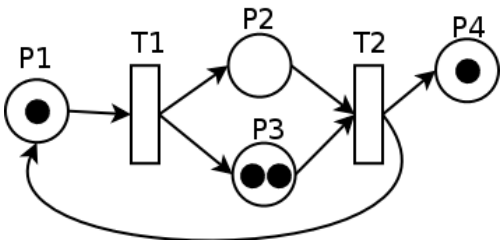


Figure 25. Petri net example

Execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire. If a transition is enabled, it may fire, but it does not have to. Since firing is nondeterministic, and multiple tokens may be present

anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems.

Many Petri net modeling tools exist but they are normally not well integrated to other modeling standards and tools used in avionics. [Modelica] has for example a Petri net extension that works, but is cumbersome to use; the number of input and/or output arcs of a place, P, is predefined and can not be changed in the model element, but the place element has to be replaced to one with the right number of predefined in/outputs.

4.2.4 The differential equation

For continuous oriented problems the differential equation has been a basic means of physical/mathematical modeling ever since the days of Isaac Newton.

In many applications, it is common to use a set of ordinary differential equations (ODE), (1), for which there are several powerful solvers and analysis tools available. A special and simpler kind of representation is the set of linear differential equations, (2). An ODE, (1), may be linearized to form (2) in a specific point of operation, and this enables usage of a broad set of (linear) analysis and synthesis techniques.

$$\begin{cases} \dot{x} = f(u,x) \\ y = g(u,x) \end{cases} \qquad \text{ODE (explicit)} \qquad (1)$$

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} AB \\ CD \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \qquad \text{Linear ODE} \qquad (2)$$

$$0 = f(u,x,\dot{x},y) \qquad \text{DAE} \qquad (3)$$

In other problem areas, in particular in mechanics and electronics where constraints (physical laws) are drivers, one ends up with sets of differential-algebraic equations (DAE), (3), which are more general and more powerful for simulation models than ODEs. Reformulation of DAEs to ODEs is time-consuming, error-prone, and sometimes impossible, while using ODEs in a “DAE tool” is straightforward, because ODEs are just a subset of DAEs.

Another kind of differential equations used in engineering modeling is the partial differential equation (PDE), in which the unknown function is a function of multiple independent variables and their partial derivatives. PDEs are used to solve, for example, heat distribution problems. As a reference to the elementary properties of differential equations and some application examples for physical problems, see [Andrews 1986].

4.2.5 Transmission line

One technique used in simulation aimed modeling of hardware components is transmission lines, a variant of power ports. It makes use of transmission line (bi-directional) data exchange where the nodes correspond to the physical connections.

Using transmission lines and distributed solvers as presented in [Krus 2005], the models become even more "object-oriented" since the component class also encapsulates a solver. Using distributed solvers with transmission lines as connection elements gives a physically motivated partitioning of the system. In this way component models can be numerically independently of each other, which provides highly robust numerical properties. This technique is useful for high speed simulation of systems, and has also been verified as robust for simulation-based optimization, where the system is simulated many times with different settings of model parameters.

The use of transmission line elements for partitioning of systems is a non-exclusive approach. Conventional simulation techniques can still be used within the subsystems. This means that components built for transmission lines can be used to connect simulation models developed in different simulation packages. Using distributed solvers also has the advantage that it allows a model to be assembled from precompiled modules. This can be valuable in collaborative system design, since it does not require disclosure of the source code or knowledge of exactly which solver technique is required, when providing a module to partners.

The introduction of tools for model specification has increased the size of systems that can be modeled efficiently and the increase in hardware performance has also made it possible to simulate and analyze these large models. One problem when dealing with large complex systems, however, is that most simulation packages rely on centralized integration algorithms that scale rather poorly with respect to system size. For large-scale systems it is an advantage if the system can be partitioned in such a way that the parts can be evaluated with only a minimum of interaction.

With the introduction of multi-core desktop computers it is possible to achieve straightforward parallelization architectures. Components with included solvers simply map (are deployed) to cores in the most optimal way.

4.2.6 Physical flow versus signal flow

A good approach that is also attractive to the user when creating simulation models of physical oriented systems is to align the model with the system's structure, or its "topology", as is depicted in the right diagram of Figure 26. It is attractive to connect the components using power ports, which also enables a descriptive specification of equations (DAE form). This means that it is used strictly/simply to define the system equations in a non-causal form and not on how they should be solved. This method is commonly called the "power port" modeling technique.

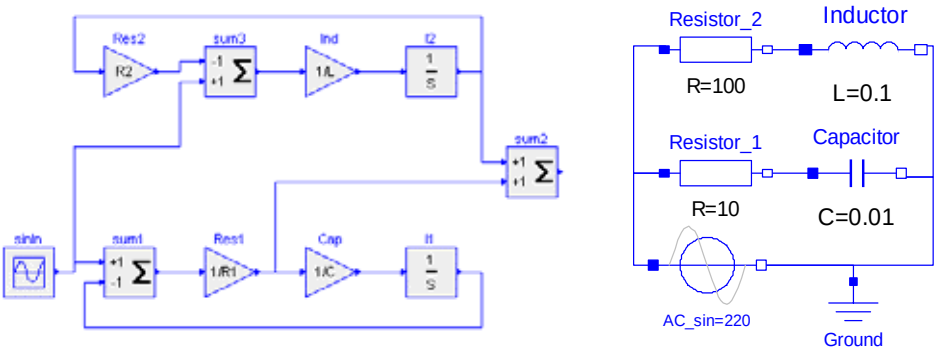


Figure 26. Left: signal flow model. Right: physical flow technique.

The left part of Figure 26 shows the same system modeled with the signal flow technique and an explicit specification of equations (ODE form). When this is used for physically oriented parts of the systems, it easily becomes overcomplicated, with arrows representing information flow in the model rather than real flow paths in the system. For smaller systems this drawback may not be an issue. Further discussion on aspects of these modeling techniques, and on integration of the two, is found in Paper [II].

4.2.7 Object orientation versus signal flow

For software intensive and embedded systems, there are two main approaches that can be viewed as common modeling techniques. One approach is based on the use of the object-oriented paradigm, which has more or less converged into UML and SysML as industry standard languages/notations. The other one is based on the signal flow paradigm that is supported by signal diagrams or data flow diagrams (DFDs) as its main modeling language. Both approaches offer important views of the system at hand, but each of them focuses on certain aspects of the system.

In order to gain understanding and to specify the objectives and capabilities of an application/system its functional requirements are extracted into an initial specification in a domain independent and platform independent manner.

One method, during this functional specification activity, is to integrate and combine the object-oriented and data flow views for analysis and design. The method consists of specification of the system following a functional decomposition, but representing it using the benefits of both object-oriented and data flow views. The argumentation for the value of integrating both views is given in [Fernandes 2004], who states:

> *Unfortunately, it exists a culture of rivalry in the software community with respect to the two major paradigms. Nowadays, the convention is to use either a 'pure' object-oriented approach or a 'pure' functional approach. We prefer to view them as complementary, each one with its own strengths and weaknesses. We think that a proper mixture of the approaches is possible, so that the best of both worlds can be achieved.* [Fernandes 2004]

In an avionics system with partitioned IMA-architecture it is possible to mix methods in the sense that different applications are developed based mainly on one of these paradigms according to certain properties, such as amount of data management, degree of time/event driven, and criticality level.

4.3 Specification techniques

A specification describes *what* the system should do, not *how* to do it. Specification of a system or product can be done in a more or less (semantically) formal[9] way. Here, the traditional (not mathematically formal) method is first briefly described. Because formal methods are more relevant for model based approaches, one example of an emerging specification technique (SPEEDS) is more thorough presented below.

4.3.1 Traditional specifications

The core information type in traditional specifications is *requirement*. The requirements are structured and evaluated e.g. for priority. In the information model, requirements are related to system elements and categorized into functional, performance or other groups of system characteristics. Without any advanced requirements management or object-oriented documentation tool, the information is handled in plain documents or in spreadsheets. When the systems are large, this has in many cases led to difficulties when reviews, updates or distributed input of requirements and related information are performed. A lot of time is spent on manually comparing different versions and merging information from several people into the “master” file.

But with a few strict rules, clearly defined responsibilities, and support from some (database) tools, it is possible to achieve cost-efficient specification even of large systems. At Saab Aerosystems, a requirement and specification process based on the information structure in [MIL-STD-498 1994], aligned with the descriptions in [Hull 2002], and supported by the [DOORS] tool is implemented.

4.3.2 Formal specifications

A formal specification is recognized as a mathematical description of a system that can be used to develop an implementation. This enables formal verification techniques to be used to show (or prove) that a given design is correct with respect to the specification.

Languages for formal specification

Several languages and methods for formal specification have been developed and some are used in industry. Examples of formal specification languages/techniques are:

- Eiffel — Language and method supporting “design by contract”, standardized by ISO
- Esterel — A synchronous languages translatable into finite-state machines

[9] The level of business/contract formality is another aspect that not is discussed here.

- Lustre — A formally defined, declarative, and synchronous dataflow language, which is the core language of the [SCADE] tool.
- Petri nets — See the separate part on Petri nets in section 4.2.3
- VDM — Vienna Development Method is a formal semantics language, enabling proof of model properties. It has an executable subset, so that models also can be analyzed by testing
- Z notation — Specification language based on elementary set theory

As semi-formal languages the following are included:

- UML/SysML — Unified Modeling Language and Systems Modeling Language, standardized by Object Management Group
- AADL — Architecture Analysis and Design Language is an architecture description language standardized by SAE.

Even though the method of formal languages has been defined for several decades and shows promise for "automatically proving" that a given design fulfills a systems specification, it has not (yet) reached large scale usage within the aerospace industry. Semi-formal languages increase however more rapidly in use.

SPEEDS

Here, the component based specification technique SPEEDS[10] will be further penetrated. For background and further documentation, see [Engel 2008] and [SPEEDS]. The SPEEDS methodology is built on a component-based modeling approach. A meta-model, Heterogeneous Rich Component (HRC), has been developed to provide a semantically founded base for the method. The HRC meta-model is extensive and defined to support representation of component models originating from languages and tools such as SysML, Simulink and SCADE. The semantics is defined such that any of these tools could be used as front-ends for creating HRC compliant models.

The core element in HRC is a component. The HRC supports the definition of contracts in a manner similar to the Eiffel method. A component in HRC can have any number of interfaces – each related to functional or non-functional properties of the component. HRC is similar to other block oriented specification methods, e.g. SysML Internal Block Diagram or Simulink block models. Any number of assertions (assumptions or promises) may be associated with each interface, where:

- An assumption is an interface statement on the expected properties (signals, physical or logical properties) fed from the environment through the interface, i.e. properties that are not controlled by the component.

[10] This technique is developed by the SPEEDS consortium, financed through the European Union 6th framework project in embedded systems development. SPEEDS stands for SPEculative and Exploratory Design in Systems Engineering, and the project is executed between 2006 and 2009.

- A promise is an interface statement of the properties guaranteed for a component interface – given that assumptions made on other component interfaces are fulfilled by the environment.

When viewed in isolation, an HRC component and its associated assertions can be presented as shown in Figure 27.

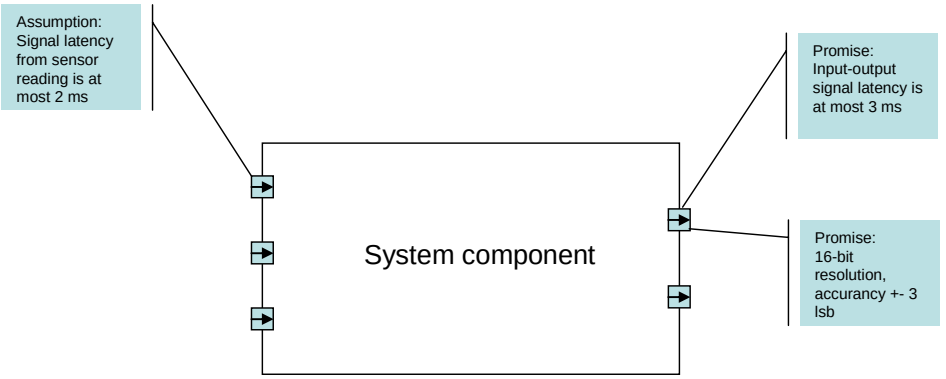


Figure 27. Component model with interfaces and Assumption - Promises

The interfaces are presented as flow interfaces; however, interfaces may be created for any system property that is handled in the engineering process, e.g. component weight or cost.

The next step is to link assumptions and promises from different components to each other to form contracts, indicating that the assumed characteristics of an input interface of a component are indeed satisfied by the output of the component providing the information. Figure 28 illustrates contracts captured between interfaces of two components K1 and K2. The semantics is such that for Contract C1 the assumption A21 must be satisfied by P11.

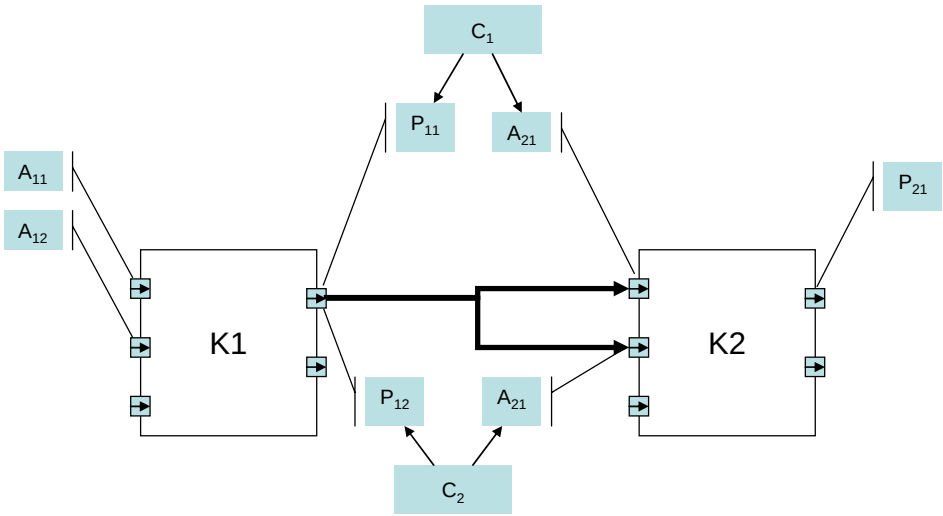


Figure 28. System model; assumptions and promises combined to contracts

Assertions can be specified in plain English, but there is also a formal language, Contract Specification Language (CSL), developed by the project, which allows formal definition of assertions. Specification with CSL enables formal analysis, so contract satisfaction could be established for C1 provided that K1 and K2 are unambiguously specified with CSL. Contracts group assertions and may be formulated between interfaces in:

- components at the same level of decomposition
- components in a parent–child relationship
- components capturing the system at different abstraction level

The SPEEDS technique, including tool support, is not yet sufficiently mature for industrial full-scale use, but it is currently being evaluated for scalability by for example Saab and Airbus.

4.4 Design techniques

A design technique (or method) helps the engineering team to understand, break down, analyze and document the engineering problem and the explored solutions. Of all documented design techniques, a small set relevant for avionics design is described here.

4.4.1 Design matrices

Matrices are a powerful means of systematically analyzing and documenting relations in the design activity. There may be any kind of information on each axis of a matrix,

and two defined matrix types are described here: Axiomatic Design and Dependency Structure Matrix.

Axiomatic Design

Axiomatic Design is a design method based on two rules/axioms, using a matrix visualization to analyze the transformation of customer needs into functional requirements, design parameters, and process variables. The name method is from its use of design principles or design Axioms (i.e. given without proof) governing the analysis and decision making process in developing high quality product or system designs. The method was developed by Dr. N. P. Suh [Suh 2001].

The basis for the theory consists of the two principles - axioms:

The Independence Axiom: Maintain the independence of the functional requirements

The Information Axiom: Minimize the information content in a design

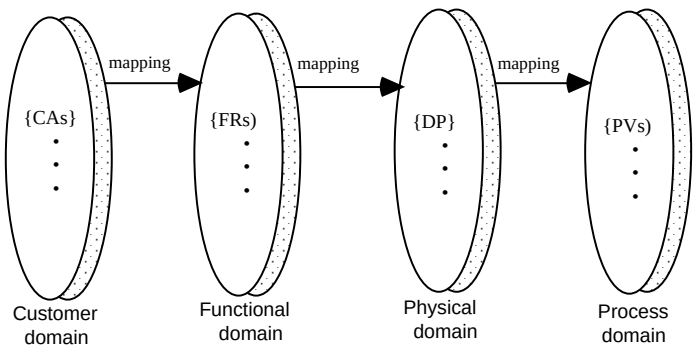


Figure 29. The four domains of the design world. {x} are characteristic vectors of each domain.

There are four domains defined according to Figure 29. The customer domain is characterized by customer needs or the attributes the customer is looking for. In the functional domain, the needs are specified in terms of functional requirements {FR} and constraints {C}. In order to satisfy the specified {FR}, design parameters {DP}, in the physical domain, are used. Finally, to produce the system specified in terms of {DP}, in the process domain one has to develop a process that is characterized by process variables, {PV}. Mapping of i.e. {FR} to {DP} is done linearly by the design matrix [A]. When analyzing [A], and in particular the sparsity of [A], one can determine how much coupling exists in a given design.

$$\begin{Bmatrix} FR_1 \\ FR_2 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \end{Bmatrix}$$

If [A] is triangular, the design is said to be decoupled and if it is diagonal, the design is uncoupled. Otherwise it is coupled. The Independence Axiom states that a design should preferably be uncoupled, and if that is impossible to achieve, it should at least be triangular. The Information Axiom says that the information (complexity) should be kept to a minimum.

Dependency Structure Matrix

A Dependency Structure Matrix (DSM), see Figure 30, is a compact, matrix representation of a system that lists all constituent subsystems or design parameters with a focus on interdependencies and information flow within and between them. DSM analysis provides insights into how to manage complex systems or projects, highlighting information flows, task sequences and iteration. It can help a design team optimize the flow of information between different interdependent components.

Name		1	7	3	4	8	9	5	6	2
Geometry	1	1								
Cooling	7		7							
Size	3	x		3						
Performance	4	x			4	x	x			
Availability	8					8	x			
Maintenance	9				x		9			
Engine	5	x						5		
Electric supply	6		x				x		6	
Weight	2	x	x			x	x	x	x	2

Series block

Coupled block

Figure 30. Example of a Dependency Structure Matrix

DSM analysis can also be used to manage the effects of change. For example, if the specification for a component had to be changed, it would be possible to quickly identify all dependencies on that specification, reducing the risk of missing interface changes.

Another similar technique is the House of Quality (HoQ) method which combines the analysis of function-to-component mapping with component-to-component dependencies. For a more thorough description and a summary of these and other matrix-based methods in aircraft design, see [Gavel 2007].

4.4.2 Function/Means tree

In large, sparse sets of data, matrices become cumbersome and here a function/means tree (or F/M tree) is more appropriate, as described in [Johansson 2006]. This is a method for functional decomposition, allocation to means (components to fulfill the requirements) and concept generation. It supports elaborative development of concepts and enables engineering with large sets of data in a structured way. For a modular avionics system (i.e. IMA) with many software applications, these allocation-based design techniques are valuable for dependency analysis and engineering support.

Function/Means tree modeling

By facilitating reuse of conceptual models of previously well studied products, more time can be spent on developing the parts that contain the edge of a new product generation. Function/Means tree modeling is a method for decomposing a main function into sub-functions with alternative solution elements that can be arranged in a hierarchical structure, see Figure 31. In this way a solution space is specified from which different concepts are generated.

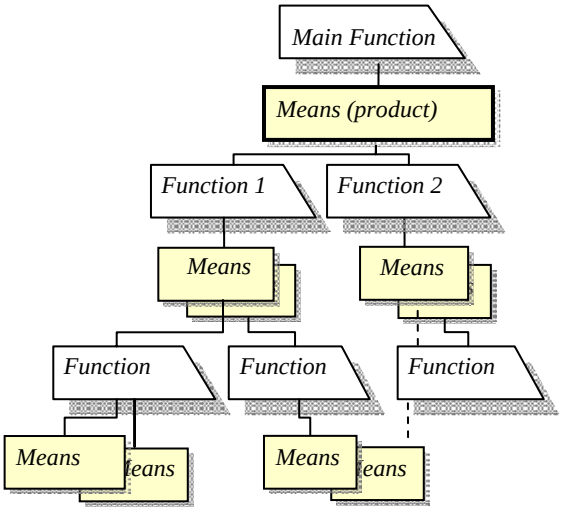


Figure 31. Function/Means tree

In a detailed function-means tree, the same functions re-appear in many places, particularly at lower levels. There is a need to reuse existing sub-trees to avoid information duplication, inconsistency, and extensive maintenance. One way to solve this is to enable an object in the tree, for example a function, to inherit all its information including a sub-tree from another function. Examples of function-means models are found in Paper [III].

Object inheritance technique for large F/M models

Conceptual design for complex products requires a considerable effort since the product models become very large if they are to cover the important aspects. To cope with this overall effort, designers have to rely on legacy designs and reuse, and improve the product concepts incrementally between product generations and variants. In Paper [III] a generalized inheritance mechanism called generic object inheritance that enables quick reuse and modification of conceptual product models at any level in their hierarchical break down structures is described. This enables the modified concepts to be kept in the context of a complete analyzable product model where the impact of changes can be studied without having to maintain multiple copies of the same object structures. The

paper describes how generic object inheritance is used for developing the next generation of a conceptual product model of a small business jet. A new version of the model is created while reusing the essential parts of the previous version with minor modifications to design parameters and substructures.

4.5 Analysis techniques

Models represent different aspects of the system, and those aspects can (partly) be verified by analyzing the model. The analysis activities are thus, performed in order to check or verify the system design. When large or complex models are used to specify the system, some analysis (e.g. static model checks) of the model itself is appropriate, such as checking consistency, semantics or syntax.

An important (and often expensive) activity for analysis/verification at system level is simulation. At component or sub-system levels, desktop simulations in the modeling tool or in specific (mid-scale) simulation software/environments add understanding and confidence to the system's design. For a whole aircraft system, this activity is characterized as large-scale-simulation with specific prerequisites as described in chapter 5.

4.5.1 Fault Tree Analysis

Fault Tree Analysis (FTA) is a top-down analysis technique that is used to identify contributing elements (errors / faults / failures) that could precipitate the system level hazards identified. FTA is a feed-back technique in that one starts with the system level hazards and works backward by identifying all possible causes of the hazards. For reference to FTA, see [ARP 4754 1996] and [Herrmann 1999].

4.5.2 Mid-scale simulation

Tests are run in software-based simulators and in hardware based system simulators (rigs) with product-equivalent computers and other equipment in the loop. There are different types of simulation facilities for mid- to large-scale simulations:

- Desktop simulation tools, cheap and easy to access.
- Handling qualities, software based, simulator with or without pilot in-the-loop.
- Presentation and maneuvering simulator with Human-Machine-Interaction in focus.
- System simulator (rig) with a large amount of target hardware and other product-equivalent equipment present. This type of simulation is defined as large-scale.

A picture of a simpler kind of simulator for mid-scale pilot in-the-loop simulations is shown in Figure 32.

Figure 32. Simple pilot in-the-loop simulator

A software-based simulation model is easy to execute with a range of different user scenarios, and this can also be done in “batch mode”, preferably distributed during “non-work-hours”, to maximize the usage of the company’s computational recourses. A set of scenarios are created with selected values of inputs (e.g. load configuration, fuel content, speed, and altitude). Out of several thousand simulated scenarios/maneuvers, including degraded modes, there is a selection of critical maneuvers to further verify. For each payload combination, a set of the most severe and critical maneuvers are selected for pilot in the loop simulations and maybe flight tests. More details about these kinds of “mid-scale” simulations can be found in Papers [I] and [II].

One drawback of the model based way, relying on simulation tools, is that some aspects are difficult to cover, so a test rig with the system’s real components has to be built anyway. For example, when verifying multi-channel systems, the inter-channel behavior such as redundancy policy, timing, and performance aspects has to be separately tested, in a hardware based rig with target software, in parallel with the model-based functional verification.

Fault injection

Models are needed for different kinds of fault conditions that need to be simulated and analyzed, and for which pilot training should be performed. What faults to model is, of course, dependent on the system concerned, but a rule of thumb is that sensors in a system are error-prone. How to model faulty conditions of a sensor is important and there are many different ways to do it; a few examples are listed below:

- Intermittent fault is a fault that repeatedly occurs and disappears. Example: loose connectors.
- Incipient fault is a fault that gradually develops from no fault to a larger and larger one. Example: A slow degradation of a component.
- Abrupt change is a fault that appears as a very quick change of a variable. Example: Sudden breakdown of a component.

For general handling of faults in a sensor or in its connection, it is convenient to build a fault injection feature into the modeling framework. Every sensor model is then enhanced with extra output settings, see Figure 33.

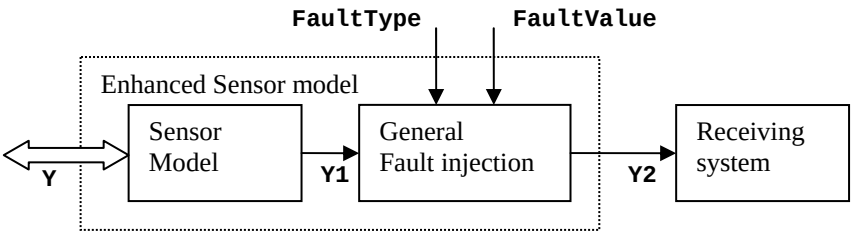


Figure 33. Sensor model enhanced with general fault injection feature

Here is an example of general fault injection settings of sensor signals;

Type	**Name**	**Meaning**
0	no fault	`Y2 = Y1;`
1	zero	`Y2 = 0;`
2	+ hard over	`Y2 = + BigNumber;`
3	- hard over	`Y2 = - BigNumber;`
4	bias	`Y2 = Y1 + FaultValue;`
5	gain	`Y2 = Y1 * FaultValue;`
6	user input	`Y2 = FaultValue;`

This kind of general functionality has in Saab Aerosystems' experience proven to be useful in mid-scale and large-scale modeling and simulation, as it is easy to implement and use. Specific sensor faults are of course needed for certain kinds of analysis or pilot training, and these are preferably built into the sensor model provided by the equipment/sensor supplier.

5 Development Environment

AN ENGINEERING ENVIRONMENT, especially useful in avionics development, comprises a suite of engineering tools designed to work together (i.e. integrated). Well-integrated tools will reduce the time wasted on user-interaction with separate tools and manually moving data between different engineering tools, or, in a worse case, between tools/applications running on different operating systems (Solaris, Windows, Linux etc). The environment also includes process/method descriptions, training and support resources. For scalability reasons the administrative support for handling user accounts, groups, permissions and other security related issues is also important. Flexible access to computational resources for heavy analysis and desktop simulations is mandatory for a modern and efficient environment.

5.1 Basic requirements for efficient team-work

A set of needs and requirements exist for a development environment that will enable efficient team-work. Stating explicit requirements on specific tools or the complete tool chain is difficult, e.g. because engineers are individuals with somewhat different needs, skills, and preferences. Some of them/us prefer text-based user interfaces whereas others favor graphical user interfaces for communication and work with the engineering tools. Table 4 below is a list of “System Requirements” elicited as part of the SPEEDS project, mainly from Saab Aerosystems, but with important input also from Airbus and IAI. The list is not by any means exhaustive, but indicates what kind of requirements actual aerospace projects and companies are concerned with.

Abbreviations used in the list are;

SM – Systems Model

SMIDE – Systems Modeling Integrated Development Environment

Table 4. List of system requirements regarding the development environment

Requirement	Rational / Additional information
Design/modeling related requirements	
D1. Each property shall get a set of predefined attributes when it is created. It shall be possible to add user defined meta information to each component.	To support project management it is beneficial if the user can add for example - estimated time (min, likely, max) to develop a certain object - information about risks This information can then be extracted and analyzed by means of analysis tools.
D2. Interface specification should provide for: - Specification of Input and Output links and data as present in the system model, and - Auxiliary interfaces by which additional links and data needed for analysis of other project aspects.	It should be emphasized that the term "interface" in this context refers just to data items definition and does not include the description of their behaviors. The intended behaviors should be specified by the views.
D3. Input/Output specification should support the following types of data: - Hardware connections (physical flow) - Information flow	Information flow may be of different kinds: - Continuous information flow - Discrete information flow
D4. SMIDE should provide for specification of closed as well as open SMs.	Closed Systems are those that "include" their environment.
D5. In open systems, it shall be possible to denote signals and properties as environment related.	
Simulation related requirements	
S1. It shall be possible to, at any time, store the current state and input of a simulation.	When performing extensive simulation/analysis it is important to be able to initialize the simulation to a specific state. This is useful for batch simulations of extensive flight envelopes.
S2. It shall be possible to initiate a simulation from a specific state independently of which tool stored the state.	A possible scenario is to start a simulation, run the simulation to a specific point, store the current state and then from this point continue to simulate the system in another tool.
S3. It shall be possible to inject faults (single and multiple) into the model to simulate failures.	One difficulty with the implementation of built-in test (BIT) is to define the correct threshold levels for setting an alarm. One way to avoid/minimize this is to simulate and determine the correct threshold using models.

Requirement	Rational / Additional information
S4. SMIDE shall provide capabilities to force steady state solutions.	It is useful to force steady state solutions (for inherently stable models) where the time to get the solution using dynamic simulations would be too long. This is known in aerospace as model "trimming".
Analysis related requirements	
A1. When finally saving an SM, SMIDE must perform a syntactic check, and report whether the SM is syntactically valid or not.	An SM is valid if it contains only data allowed to be stored in a SM. Temporary saving of an incorrect SM should be allowed.
A2. SMIDE shall provide fault analysis capability.	Needed to enable safety and reliability analysis.
A3. Tools in SMIDE shall have at least two roles: Modeling Tool or Analysis Tool.	One tool could play several different roles, i.e. modeling and simulation.
Configuration related requirements	
C1. It shall be possible to store a model, that is model structure and static parameters, and restore it at any time.	If several users share the same model and changes are made to the model's various parameters, one must be able to restore the model.
C2. SMIDE shall support parameterized models.	This can be used to introduce faults during simulation and support failure mode analysis. Example: During simulation of an electronic circuit, the resistance of a resistor can be set to zero to simulate a short circuit. Examples of parameters are: - offset in a sensor signal - arbitrary gains
C3. SMIDE shall support: - Alternative concepts - Different product variants	The basic means for handling concepts and variants may be the same, but handling of concepts should be less formal than handling of variants.
Library and scale-up related requirements	
L1. It shall be possible to store and maintain the following meta information for each object/component: - Owner - Date/time created - Date/time last modified - Modified by - Verified (yes/no, when, by whom)	When working with models in a large scale environment, it is important to verify the model before using it. Each update requires that the model is re-verified. By including meta information regarding changes and verification in the library, the users can easily see the status of the component.

5.2 Examples of tools for industrial use

In this work, a set of tools for modeling, analysis and simulation purposes have been listed for evaluation. Here are some tools with their potential contributions to an industrial development environment listed by functionality.

5.2.1 Requirements handling

Besides the basic management of textual written requirements (that could be handled in any text editor), a modern requirements tool has, for example, capabilities for:

- Versioning
- Base lining
- History
- Traceability
- Prioritizing

Added to this there are basic database functions, such as back-up and logging.

DOORS

The Requirements Management tool [DOORS] from IBM/Telelogic is one of the leading tools for advanced requirements handling within product, systems, and software development. Compared to document based handling, it offers features for large team requirements communication and collaboration. In DOORS the information is structured in “modules” which gives a “document-oriented” look and feel for the end user. It also has filtering and sorting functions for creating views of the modules, enabling a flexible way of presenting and extracting the information, even though it is not as flexible as a true object-oriented or relational-database based tool. The combination of DOORS relative flexibility and the document-oriented meta-model is probably one reason for its popularity.

Focal Point

IBM/Telelogic [Focal Point] is a configurable web-based decision support platform for requirements management, product management and project portfolio management. The key feature is “pair-wise comparison” with underlying algorithms for efficient prioritization of large sets of requirements. It is possible to integrate Focal Point with a DOORS database to more easily retain consistent information in the requirements management process.

5.2.2 Modelica and related tools

[Modelica] is a descriptive, object-oriented modeling language suited to physical based simulations and analysis of behavior and performance. There are model libraries, both open source and commercial ones, typically supporting mechanical, electrical, electronic, hydraulic, thermal, electric power or process-oriented components. An example of an ideal capacitor component is presented in the textual model definition below.

```
model Capacitor
  Pin p;
  Pin n;
  Real v;
  Real i;
  parameter Real C "Capacitance";
equation
  0 = C * der(v) - i;
  0 = p.v - n.v - v;
  0 = p.i + n.i;
  0 = p.i - i;
end Capacitor;
```

In this model the parameter (**C**) is used to instantiate the model with different values of its capacitance. This kind of parameter sized component is well suited for reuse and sharing between users through a model library. As reference to modeling with Modelica, see [Fritzson 2004].

Dymola

Dynamic Modeling Laboratory, [Dymola], is the most mature environment for Modelica modeling and simulation. In Figure 34, a Dymola model is shown, using an instance of the capacitor model defined above, with a capacitance of 2200 µF.

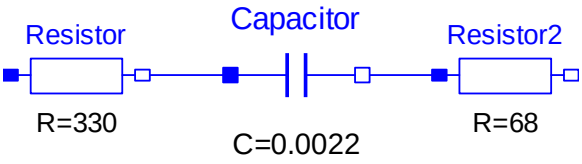


Figure 34. Small Modelica example from the Dymola tool

Dymola is considered, in Saab Aerosystems' experience to have reached the maturity level required for mid-scale modeling and simulation in the industrial engineering environment.

5.2.3 Mathematics based tools

Matlab

Matrix Laboratory, [Matlab], is a tool with the high-level Matlab language and interactive environment that enables computational prototyping in an interpreting fashion. For several years Matlab has been chosen by engineers and scientists for its data handling capabilities i.e. transformation and visualization/plotting of data. In the field of control theory there are several toolboxes for specific design and analysis techniques.

5.2.4 Signal flow modeling tools

These tools expands their functionality successively, with e.g. discrete states diagrams, and support for physical components and power port technique.

SCADE

[SCADE] stands for Safety Critical Application Development Environment and is a commercial product from Esterel Technologies. It is based upon the formal, synchronous and data-flow oriented Lustre programming language. The toolset may generate C or Ada and it is, according to its vendor, [Esterel], qualified as a development tool for DO-178B up to level A. As a result, its main application fields are aerospace and avionics, but it is used also in other industries (e.g. automotive, rail transportation and nuclear power plants).

Simulink

[Simulink] is a platform built on Matlab for multi-domain simulation and Model Based Design for dynamic systems. It is time domain based and provides an interactive graphical environment and a customizable set of block libraries that can be designed for specialized applications. Simulink has block-sets containing blocks for both continuous and discrete models. In Figure 35, two different Simulink representations of the same model as in Figure 22 is shown, a time-continuous and a time-discrete model.

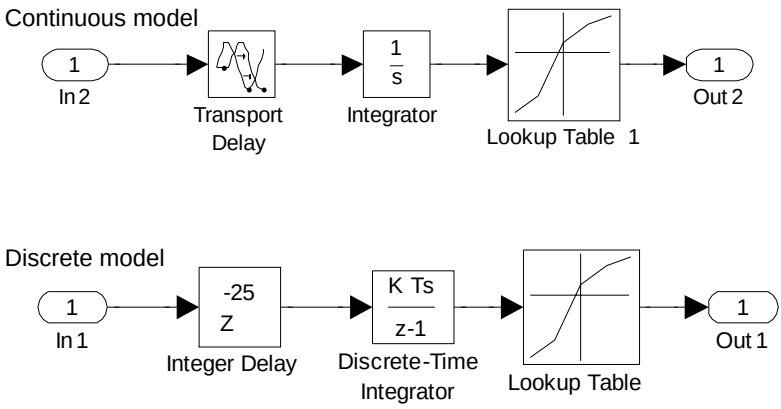


Figure 35. Simulink example models

The Simulink simulation engine includes a set of integration algorithms, or solvers, which are based on ordinary differential equations (ODE). A sophisticated ODE solver uses variable time-steps that adaptively select a time-step tuned to the smallest time constant of the system, and also back-tracks whenever the truncation error exceeds a limit set by the user. From release R2008b the Simulink team has implemented a model-

ing language based on Matlab (a Matlab dialect), called [Simscape], for multiphysics non-causal modeling and simulation, similar to Modelica.

SystemBuild

[SystemBuild] is part of the MATRIXx product family and is a graphical environment for model development and simulation suited for management of large models. With the hierarchical organization, the models may be segmented at different levels. Reuse or/and share of models is simplified with its structuring principles. It has simulation and analysis capabilities for system verification and model validation.

SystemBuild is very similar to Simulink, but it has not as large market share and is, in resent years, not as heavily developed with new features and toolboxes, as Simulink. Further description of SystemBuild is found in Paper [I] and comparison to Simulink is found in Paper [IV].

5.2.5 Tools for UML and SysML

The standardization organization Object Management Group (OMG) has released specifications for Unified Modeling Language [UML 2007], and Systems Modeling Language [SysML 2008]. Both are general-purpose object-oriented graphical modeling languages for specifying, analyzing, designing, and verifying complex systems.

UML provides graphical notation with a semantic foundation for modeling behavior and structure. SysML represents a subset of UML with extensions for requirements and parametrics (basic mathematical support) needed for Systems Engineering. Both have weak support for building simulation models, but most tools have code generation engines, enabling compilation and execution. The defined SysML diagram types are shown in Figure 36.

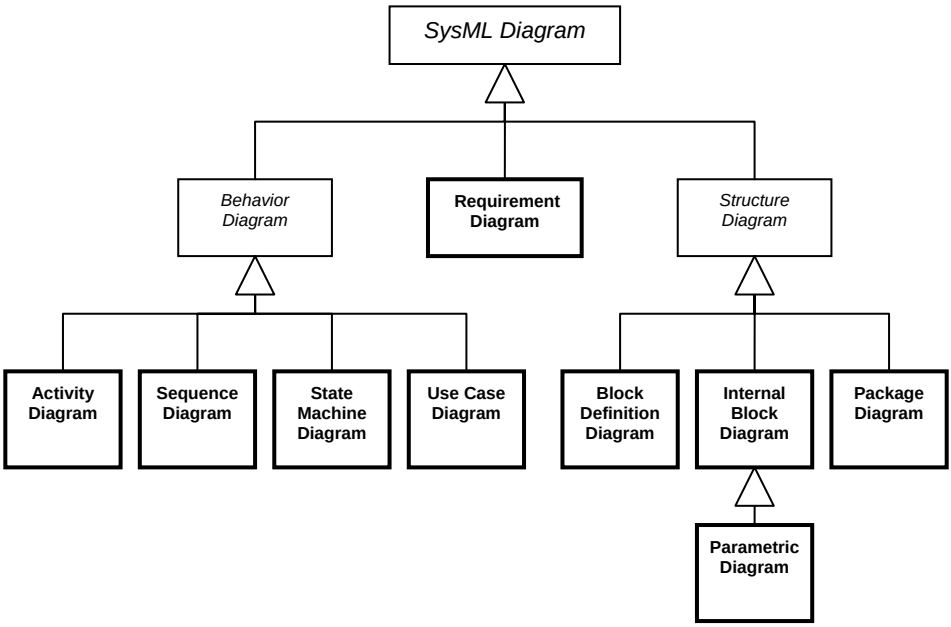


Figure 36. Diagram types defined in SysML 1.1

It is convenient for a specific project to reduce the set of UML/SysML diagrams used, as there is some overlap between the diagram types. A limited set also simplifies the introduction of UML/SysML modeling including guidelines, training and tool set-up. As described in Paper [V], an appropriate set of diagrams to start with in avionics system development is:

- Use Case and Activity Diagrams for analysis.
- Class/Block Definition, Sequence, State Machine, and Deployment Diagrams for design, implementation, and test.

SysML also has built-in definitions of dimensions and SI-units (e.g. "**`dimension; Frequency`**, **`unit; Hertz`**" or "**`dimension; Power`**, **`unit; Watt`**"). It is possible to add "user-defined" units, which is necessary in avionics/aviation specification and design. Handling of, for example, flight speed and altitude is, in the aviation community, done in the non-SI-units **`Knot`** and **`Foot`**.

For further details on the SysML language, see for example [Herzog 2005], [Friedenthal 2008], and [Weilkiens 2008]. Here follow some examples of commercial tools capable of both UML and SysML modeling. In addition there exist a range of freeware tools, in particular for UML.

Magic Draw

[Magic Draw] is a newly developed tool with a modern look and feel and is profiled as the UML tool with best alignment to the XMI storage format. There is an add-on for the SysML version of Magic Draw, connecting parametrics in SysML with mathematic/analysis tools, called “Paramagic”. The rationale and concepts behind this integration can be found in [Peak 2007].

Rhapsody

IBM/Telelogic [Rhapsody] is an environment for model-driven development of software intensive systems and is used for specification, design, and for test. Rhapsody specifies systems and software design graphically using the UML and SysML notations and enables systems engineers and software developers to simulate and verify software.

Even though Rhapsody has its background and main field of application in software development, and traditionally weaker support for systems engineering, the SysML add-on is beginning to mature. An example from Rhapsody of visualization of the user-defined units **`Knot`** and **`Foot`** is shown in Figure 37.

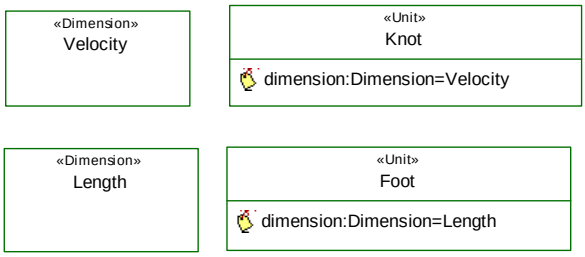


Figure 37. User-defined units in Rhapsody

In Paper [V] an application example using UML/SysML from Saab Aerosystems and the Skeldar project is described.

5.2.6 State oriented modeling tools

State oriented modeling with the Statechart formalism gained an industrial breakthrough with the introduction of diagram hierarchy [Harel 1987] and implementation in the tool “StateMate”.

Stateflow

[Stateflow] is an add-on tool to Matlab/Simulink and the essential difference from Statecharts in the StateMate implementation, is in the action language. The Stateflow action language has been extended primarily to reference Matlab functions and Matlab

workspace variables. Further, the concept of condition action has been added to the transition expression.

BridgePoint

In [BridgePoint], the UML subset “executable UML” (xUML) is used as modeling notation. This technique has its roots in the Shlaer-Mellor method, see [Starr 1996], and makes usage of the MDA principles as described at the end of section 3.3.3. To generate code from a BridgePoint system/software model, the rules and mappings are defined by a meta-model in the same fashion as the system/software model. This makes it possible to create user-tunable structure and code layout in a more flexible way than many other tools. In, for example, Simulink/Stateflow or Rhapsody, a language different from the system modeling language is used to control the code layout, requiring one a further competence.

5.2.7 Tools for event based simulation and analysis

For event based simulations of large models or systems, the tool needs a number of basic functionality or modeling entities such as clocks, events lists, random-number generator, and statistics reporting.

SimEvents

As a block-set for Simulink/Stateflow, the [SimEvents] add-on has capabilities to simulate and create graphical models of event based systems. SimEvents works with Stateflow to represent systems containing detailed state-transition charts that may produce or be controlled by discrete events. It is simulation oriented with statistics measure capabilities and has no actual analysis engine based on formal methods.

UPPAAL

UPPAAL is a university developed tool created in cooperation between Uppsala and Aalborg universities and is made for modeling, simulation, and verification of real-time systems, see [UPPAAL 2008] and [Behrmann 2006]. It is useful for systems appropriately modeled as collections of non-deterministic processes with finite control structure and real-valued clocks (i.e. timed automata), communicating through channels or shared data structures. Typical applications include real-time controllers, communication protocols, and other systems where timing aspects are critical.

The UPPAAL tool has both a graphical user interface for modeling and a command line interface for e.g. batch verifications. Studies in large scale usage are reported by for example [Braspenning 2008], but UPPAAL is nonetheless a research oriented tool. Even if the algorithms for analysis of larger models have been improved, the problem of state explosion is still an issue for further research.

5.2.8 Tools summary

Table 5 summarizes some tools, listed by modeling domains and divided into the specification and analysis classification according to section 3.4.2.

Table 5. Examples of tools used for specification or analysis, listed by modeling domain

	Specification		**Analysis**	
	Method	**Tool example**	**Aspect/Method**	**Tool example**
Structure	Spatial & Solid modeling (CAD) Assembly, production (CAM)	CATIA Delmia	Stress, FEA Flow, CFD	Nastran, Abacus StarCD, Edge, Ensight, Matlab, WIND
Physical systems	Architecture: • What components? Dimensioning: • Size and number of components	SysML Power-Port tools Modelica/Dymola CATIA	Simulation FTA, FMEA System safety Availability Optimization	Easy5 Dymola Fault Tree Plus (FT+)
Electronics/ Optronics	Architecture: • What components? Dimensioning: • Size and number of components	SysML pSpice	Simulation FTA, FMEA System safety Availability Heat & cooling	Rodon
Control	(Synthesis) Filter-design Control loops LQ	SystemBuild Simulink SCADE	Robustness Simulation Analyze in freq. domain	MatrixX, Matlab Ares, Styrsim
Logic/ Data	Object-Oriented Design Information flow State Machines Mode Logic	SysML/UML Stateflow SCADE StateMate	Object-Oriented Analysis Req. traceability	NP Tools PolySpace UPPAAL SimEvents
MMI/ Virtual surrounding	Environment, Graphical layout	VAPS Qt	Usability	

This section and the table exemplify tools related to the defined modeling domains. There are, however, other important engineering methods/tools for avionics design and aircraft simulation not listed in this thesis.

5.3 Integration

Advances in domain specific modeling have created powerful tools for their main purpose, as exemplified in Table 5. Many tools have, in the big view, a limited area of modeling or analysis, which has highlighted the need to integrate information/models from the different modeling domains. For the purpose of analysis or simulation of a larger scope (e.g. aircraft level), models and tools based on different modeling techniques needs to be integrated. An example of a desirable condition would be a system model integrated from these three kinds of (sub) models:

- models of hardware (such as resistors and capacitors in electricity, or pipes and nozzles for fluids) for performance evaluation and dimensioning
- models of the embedded software for control and monitoring of the hardware
- models of the environment of the system, e.g. other subsystems

Currently, text documents and spreadsheets in proprietary data formats predominate for information exchange in the systems engineering activities, but standards for data exchange are emerging, such as:

- AP233 - ISO 10303, Standard for exchanging systems engineering data, see [AP233]
- XMI - XML Metadata Interchange defined for the UML framework, see [XMI 2002]
- RIF - Requirements Interchange Format, see [RIF 2008]

In the remaining part of this section, some integration strategies are described and an example from the Gripen avionics demonstrator is provided.

5.3.1 Integration strategy

Because of the hierarchical building-block based product structure, information and models from different integration levels but also from different modeling domains often have to be integrated to be part of analysis and/or verification activities. [Carloni] argues that a single environment cannot offer a complete solution to the needs of designers who use hybrid/integrated models to represent the system under development. One example is simulation of a physical system together with the control and monitoring algorithms implemented as software. The Hosted Simulation technique for this integration purpose is described in Paper [II].

In Saab Aerosystems' experiences a "loose integration" strategy should be chosen in order to avoid lock-in effects and costly long-term maintenance of the tool integrations (the "tool glue"). This requires formats and interfaces to be clearly defined or standardized in order to integrate the tools, and maintain this integration. Tight integration, on the other hand, relies on the tools being connected and running in parallel with ex-

change of data when performing an analysis or a simulation. The loose integration concept reduces the dependency of two or several tools available simultaneously.

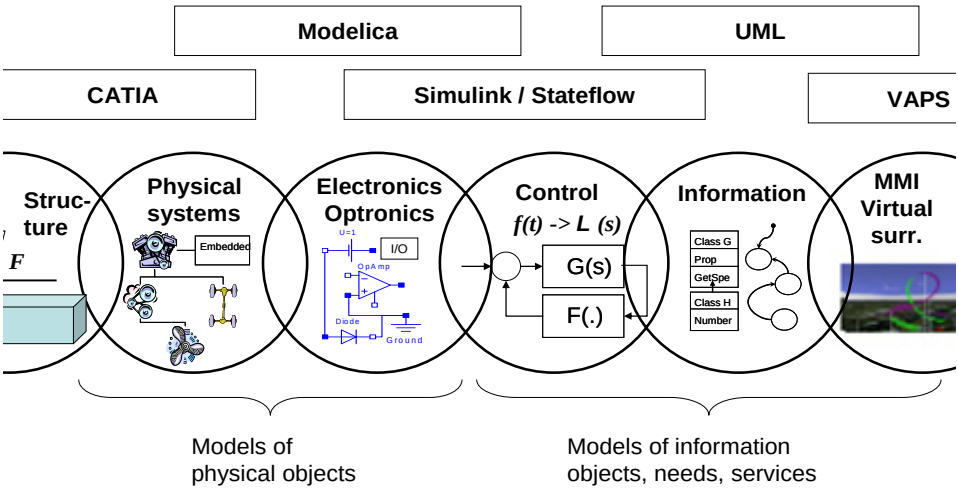


Figure 38. Integration of modeling domains and examples of tools.

There are several interfaces between modeling domains that need to be integrated. In Figure 38 the main pattern of integration needs is shown, going from a “heavy” hardware/structure to the left, through equipment, electronics, control, and information, to a “soft” graphical layout to the right. When analyzing features of development tools it is interesting to notice that many tool vendors try to cover larger and larger parts of this domain map by adding features and functions to their tools. Three examples, with reference to Figure 38, are;

- Modelica is announced to be integrated into a coming release of CATIA, extending its capabilities for dynamic simulation (extension of CATIA to the right in the figure), see [Dassault 2006].
- In the Modelica language, libraries have been developed for Petri nets and statecharts to extend Modelica (to the right in the figure), see [Modelica].
- Simulink is integrated with Stateflow (extension to the right in the figure) and enhanced with Simscape (extension to the left in the figure), see [Simulink], [Stateflow], and [Simscape].

Tools with functionality that support multiple modeling domains are of course good; it gives the engineers and the team a possibility to choose among several tools for a specific engineering task. In large scale projects, however, with several teams established over a period of time, focusing each one on specific engineering tasks, and with some freedom of choice within the teams regarding the tool chain, it might give an overall diversity of tools and the way they are used. This may lead to a sub-optimized imple-

mentation of methods/tools and inefficiency in the long term, especially regarding specification modeling.

Concerning modeling for simulation, connection or integration of the modeling domains is done at different levels. At component and subsystem levels, integrated simulation can be done using co-simulation or hosted simulation techniques, performed in desktop tools. At higher levels of integration more execution efficient techniques appropriate for large-scale simulation have to be used.

5.3.2 Behavioral modeling for software design

Behavior modeling in early stages is done in a conceptual way, focusing on system functions with related driving requirements and usually by simplified views, using for example basic activity or sequence diagrams. This will not be further discussed here. For later stages, a component approach is preferable, where the architecture is set, components are defined, and detailed behavior of each component is modeled using UML, Simulink or some similar technique. With reference to Papers [I] and [IV], some aspects of behavior modeling for software design are discussed below.

In Paper [IV] three different approaches of Simulink usage for detailed software design are presented:

1) A functional oriented systems modeling and simulation approach where the function is in focus; complete enough to be simulatable, but abstract from an implementation point of view.

2) An implementation oriented specification approach that is based on a modeling framework with predefined system architecture, scheduling, data types and rules for discretization. The resulting embedded software is hand coded using the model as specification.

3) Similar to approach two, but here the embedded software is automatically generated using a high quality code generator.

Mapping of the different approaches to the development process is shown in Figure 39.

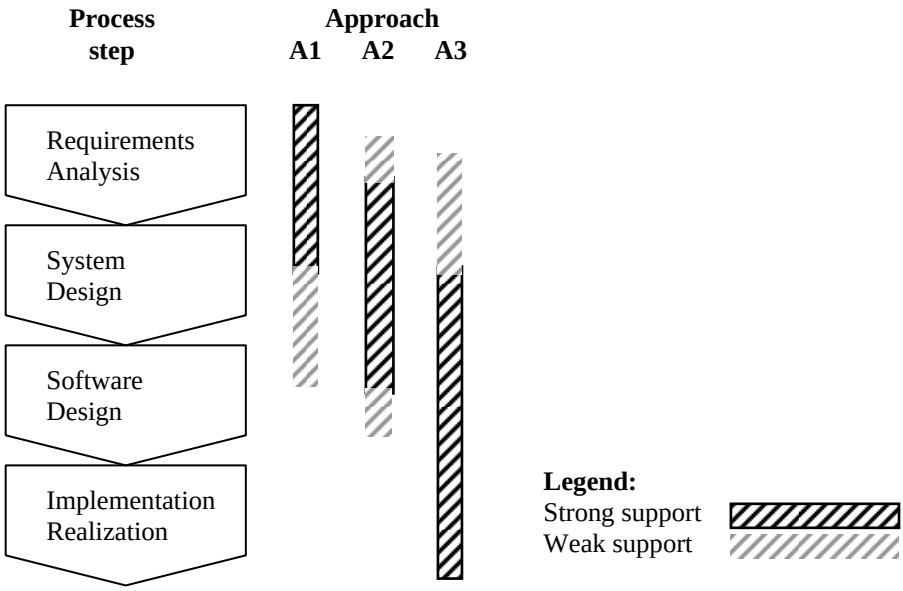


Figure 39. Mapping of modeling approach to development process activities

Further details and a small example of models according to the three approaches made with Simulink can be found in Paper [IV]. In Paper [I], which describes model based development of the Gripen flight control system, the modeling tool is System-Build and the approach used is the intermediate one (number two) described above.

5.3.3 Integrated simulation techniques

Several commercial and in-house developed tools exist for connecting different simulation models. It is a growing challenge to use and integrate simulation models from different domains, as an increasing part of the end system verification relies on results from simulation models rather than expensive testing in flight tests. The development of computer performance and modeling-and-simulation tools enables simulation on larger and larger scales. Consequently, the need for integrated models, and their validation is growing. It is then a challenge when models are based on different modeling techniques/tools. Simulation (sub)-models for aircraft systems can be organized into the following major categories:

- Equipment models (e.g. resistors and capacitors in electricity, pipes and nozzles in hydraulics) for performance evaluation and dimensioning;
- Models of the embedded software for control of system functions and for monitoring of functions and of the equipment/hardware;
- Models of the environment of the system.

There are different ways of performing integrated simulation. Here, two methods for desk-top simulation (up to mid-scale size) are described.

Co-simulation

The concept of co-simulation is to integrate models combined from the different simulation engines and execute them concurrently. This approach is supported by several tool vendors, who provide add-ons or packages for connecting tools together for simulation set-up. Co-simulation is very suitable for a system of ECUs connected by a data bus, but less useful when connected models have physical interactions, requiring power port modeling techniques and a central equation solver.

Hosted simulation

Another method for combining simulation models is Hosted Simulation (HS), where the simulation engine from one of the modeling and simulation tools is used to "host" other models during simulation. Naturally, models are developed with different tools relying on different modeling techniques – each focused on supporting a specific engineering discipline. When the models are combined for simulation purposes, the scope of the integrated model might be too broad for any of the modeling tools. The method of hosted simulation is enabled through code generation. A model created in one tool is simply generated to executable code and imported (hosted) in another tool to perform the simulation.

The principle and definitions are brought from the [SPEEDS] project. With hosted simulation, the engineering team can combine several types of systems such as mechanical, hydraulic, or electrical with systems such as sensor, control, and software. The resulting model structure is a heterogeneous engineering system and an example of how to cope with the complexity of aircraft system development.

Evaluation of integrated simulation techniques

Within the work done for Paper [II], two different approaches for hosted simulation were evaluated: one with a power port tool (Dymola) as hosting tool, and one with a signal flow tool (Simulink/Stateflow) as host. The main conclusion was that Dymola is preferred as host if the system to be analyzed is equipment (hardware) intensive and vice versa. For systems with considerable content of both software and hardware, a process relying on both approaches is recommended, because the combination of analysis results then will contribute substantially as a basis for system validation and verification.

It was further noted that the hosted simulation technique has the following advantages compared with co-simulation:

- The user is not required to have all the tools to execute the simulation. Only the simulation tool is needed.
- All the simulation results are available in a single tool.

- It gives better performance: no overhead in using a message bus and coordination between different simulators.
- It has the ability to simulate interaction with a component even if the design tool has no simulation capabilities, which is the case with many UML tools.

The main drawback is that one cannot view the internal behavior of hosted models - only monitor their interactions. Application of hosted simulation is mainly restricted to desktop and software based simulation systems and is considered to be of mid-scale size. More details of hosted simulation can be found in Paper [II].

5.4 Examples

The appended papers provide some examples on application of MBSE/MBD and development environments/tools used:

- Design and development through modeling, and analysis by extensive batch simulations of Gripen flight control laws with the “phase compensated rate limiter”, in Paper [I].
- Design of “Extended Kalman filter” for the “Synthetic Attitude and Heading Reference System (SAHRS)” function in the Gripen product, in Paper [I].
- An UAV (unmanned aerial vehicle) fuel system model, modeled in both the power port tool Dymola and the signal flow tool Simulink, and then analyzed through hosted simulation. See Paper [II].
- A conceptual design model of the fictitious small business jet RAVEN, through function/means modeling with the FMDesign tool, in Paper [III].
- Systems modeling and specification with UML/SysML and the Rhapsody tool during development of SKELDAR V150 – a short to medium range mobile UAV system, in Paper [V].

One further example is taken from the Gripen-Avionics-Demo project at Saab Aerosystems, and shows how the new emergency mode implementation for Gripen was developed. Function prototyping, including architecture, signal flow and display layout, as shown in Figure 40, was performed in a model consisting of the following parts:

- An architecture model in SysML
- A display layout model in VAPS XT
- A Simulink model with scaling, filtering and logical functions

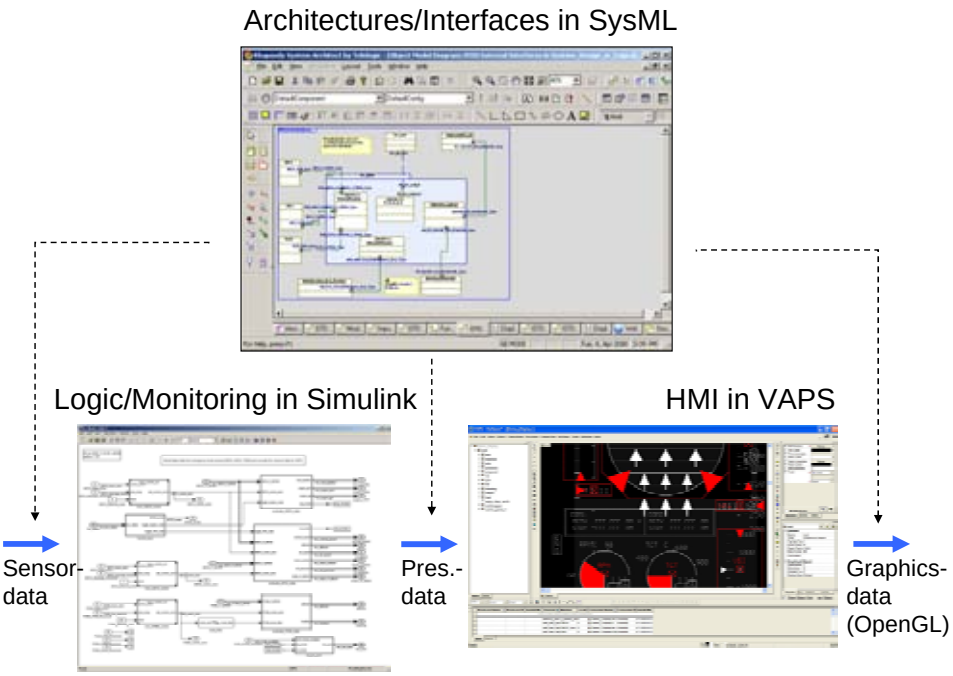


Figure 40. Gripen emergency mode model(s) for presentation of critical flight data

When the preliminary design was reviewed and agreed it moved from prototype or concept phase into development phase. It is important here to change the view of the model(s) itself. In the concept phase it should be easy to change, test new variants, simulate without all details, and so on.

In the transition from concept to development, the model undergoes a formalization conversion. All parts not following the modeling rules and guidelines are replaced, for example, computation rate is specified and time-continuous blocks are replaced with their dime-discrete equivalents. Additional model annotations and documentation are reviewed to enable efficient updates and maintenance of the function in the long-term.

6 Model management

THIS SECTION OUTLINES how the presented model based methods and tools may fit into an existing typical development environment for aircraft or avionics embedded systems. Models, modeling techniques and the supporting tools have to be managed during the product's whole life-cycle. Every update of a model has to be validated for wider use. Configuration support is somewhat different for models than for the end product because one model may, and often does, represent several variants of the end product. Evaluation of existing and new methodology concepts, installation of tool updates and education is also included in management of the modeling framework.

6.1 Model management strategy

For a complex product, when shifting from a traditional document centric methodology to a model based, strategies for how to manage and scale-up the set of models have to be elaborated and planned. Strategies for formalism, configuration management, and if/how to get a specific model accredited in the certification process are included. This includes, for example, how to:

- Distinguish between a system model and models of respective components in the system
- Differentiate management of software models versus models of hardware
- Reuse functions/parts and store reusable ones in a library
- Use different levels of formalism; which models are formal and require validation and a more stringent lifecycle control
- Manage specification models versus models for analysis purposes

A deeper discussion of the aspects of separation, reuse and validation can be found in the remaining part of this section.

6.1.1 Separation of system levels

To be able to work in parallel in teams, the model partition and the size of model's components sets the boundary. In Figure 41 one way of separation applicable for system/software development is shown, with examples of tools to use for the different models. At the top, a system model is kept relatively independent of other models. In the system model, component names, interfaces, and common definitions are defined.

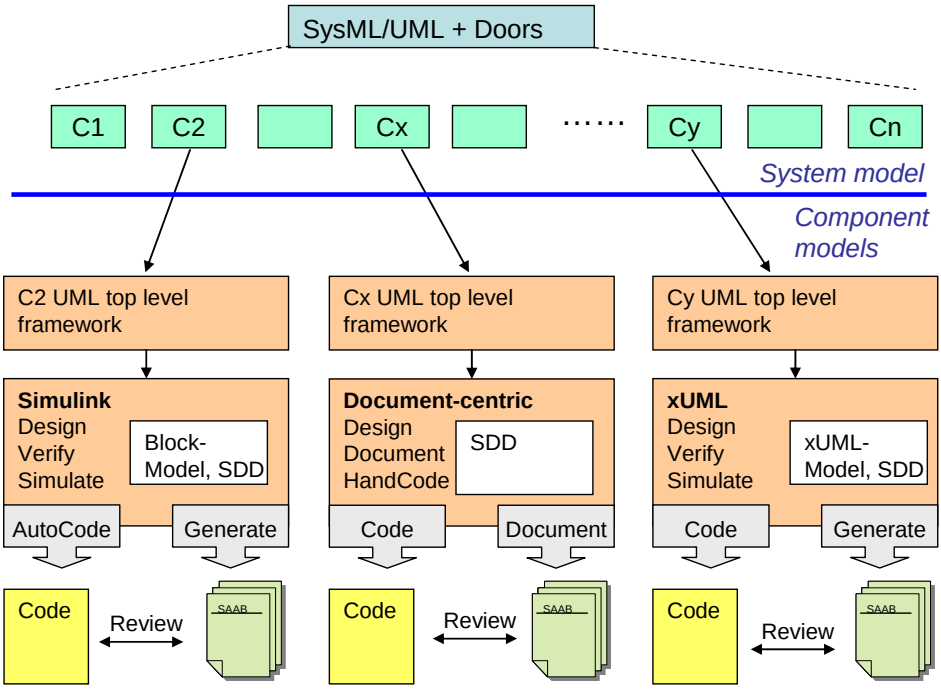


Figure 41. Separation of system model from component models.

Below the line, the components are developed as separate models, each with different methodology: Simulink-based, document-centric and xUML-based methods. The rule is to allow dependencies from component models to the system model, but make the system model independent of all component models. This implies that all libraries of reusable entities (for example type definitions, definition of units and filter function blocks) belong to the system model. When a component model is developed in the same (UML/SysML) tool as the system model, the reference mechanism is used to get access to the definitions from the component model, so they need not to be duplicated. For each component there is also a top level framework model for the component architecture and "local" definitions.

6.1.2 Library support

Another important scale-up and management issue is support or libraries for reuse of engineering components, containing "legacy" information such as specification, design or implementation. In the concept phase information of available technology and equipment (datasheets) is needed, and later on the needs are more for design patterns or verified algorithms/functions. Support for large-scale engineering data management (EDM) for conceptual development is described in Paper [III].

For aerospace systems the verification aspect is important and should be handled within libraries for those reusable components employed during detailed design. In design environments for system design and analysis like Simulink or Modelica, block libraries for reusable blocks are fundamental. These two kinds of user-defined and in-house developed component (block) libraries need to be available and easily accessible in the engineering environment when building behavioral models:

- Models of physical components/equipment mainly for desktop simulation, e.g. pumps, valves & turbines *("Component models")*
- Reusable complex functions for usage in the real-time embedded software, e.g. filters, latches & coordinate transformations, *("Model components")*

For "component models", the Modelica language is appropriate as the modeling means. "Model components" may be implemented in Simulink in several ways: as S-functions, model references, or as Embedded Matlab blocks. To create/modify and assure the quality of the block libraries a simple process is defined, see Figure 42.

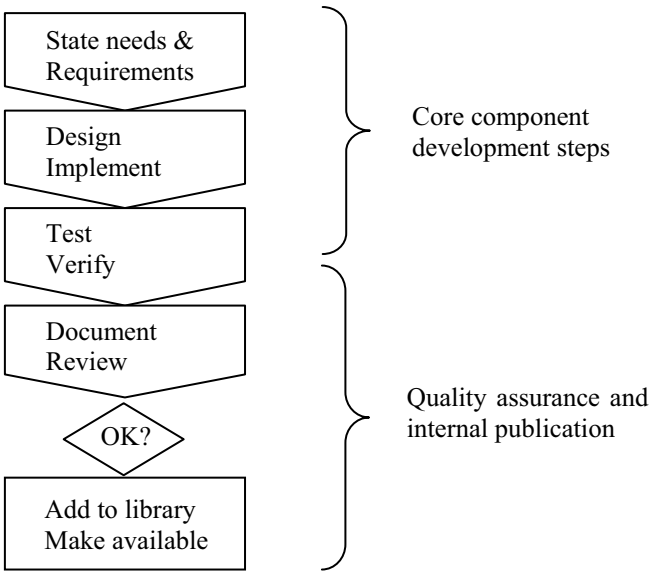


Figure 42. Process for creation/modification and "publication" of library blocks

The same basic process is used to develop both kinds of components.

There are many ways to hierarchically organize a design library so that the reusable components are easy to find during the creative interactive work. Usually there is no single hierarchical structure that is suitable for all types of interactive search tasks. Reusable objects can be categorized according to different aspects and made reachable through many parallel categorization trees. A description in Paper [III] shows one technique for building multiple parallel categorization trees.

6.1.3 Model validation

Similar to validation of the system, model validation should answer the question "*Did we build the right model?*" Adopted from [Sargent 2007], Figure 43 describes verification and validation in the modeling process.

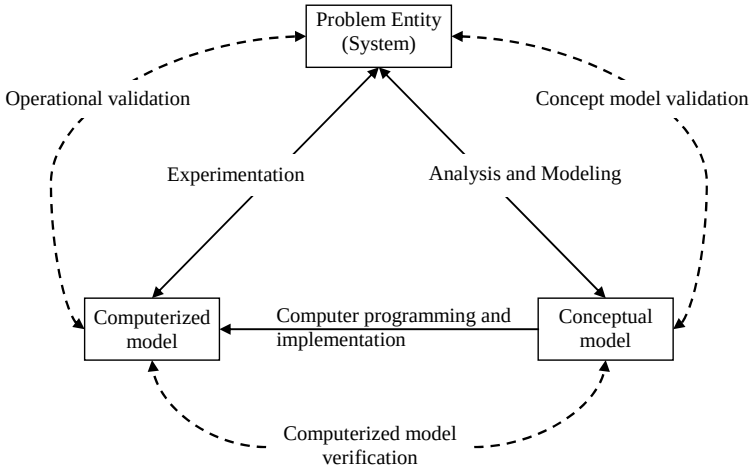


Figure 43. Verification and validation in the modeling process

The problem entity is the system subject to be modeled. The conceptual model is a mathematical/logical representation of the system, written in a model specification. The computerized model is the conceptual model implemented in a computer. The definition of activities in Figure 43 is according to [Sargent 2007]:

"*Conceptual model validation is defined as determining that the theories and assumptions underlying the conceptual model are correct and that the model representation of the problem entity is "reasonable" for the intended purpose of the model.*

Computerized model verification is defined as assuring that the computer programming and implementation of the conceptual model is correct.

Operational validation is defined as determining that the model's output behavior has sufficient accuracy for the model's intended purpose over the domain of the model's intended applicability."

With modeling and simulation tools, such as Dymola or Simulink, (for example reference, see further Paper [II]), the number of programming and implementation errors may be reduced. The activity *computerized model verification* then primarily ensures that an error-free simulation tool has been used, that the simulation language has been properly implemented on the computer, and that the model has been programmed correctly in the simulation language. The activity *operational validation* can not be fully performed in early development phases such as the concept phase due to the need for measured data from the real system. A sensitivity analysis can still be done, however, in order to point out model component parameters that have strong influence on the overall simulation results.

6.2 Simulators and simulation models

The main objective for simulation in aerospace is to reduce risk and cost. In the early stages, risk and cost are reduced by gaining a better understanding of how to specify the system/product and what the tough constraints are. In later stages, simulation generates "flight-hours" providing both engineers and pilots with knowledge about the system for verification and usage. In this section some aspects of simulation related to the modeling/development environment are discussed.

6.2.1 System verification

Verification answers the question; *"Does the system fulfill the specified requirements?"* or to put it briefly *"Did we build the thing right?"* and can be performed using multiple verification methods including test, inspection, demonstration, and analysis, even thought test usually becomes the main activity for system verification. When a model of the system is available (and has been validated), the model serves as a substitute for the real system during the verification activity.

Verification activities are performed at different levels according to the V-model:

- Functional verification
- Software test/verification
- System test/verification

6.2.2 Large scale simulation

When integrating several ECUs, data buses, displays, and controls, simulation has in Saab Aerosystems' long-term experience, proven to be an efficient method and is the most used means of system verification of large, safety-critical avionics systems. A simulation system consists of both hardware and software, but the trend is to rely more on software models of equipment that was earlier either a prototype of the production set or a simplified (hardware) equivalent. The introduction of software based simulation

systems enables more flexible and versatile usage, such as analysis through batch simulations. But new challenges in terms of for example configuration management also appear, as described below.

Batch mode simulation

Batch analyses are performed by running the simulation in a specified set of operating points (automatically by scripts). A set of scenarios are created in the same way as described for mid-scale simulations. It is also possible to introduce failures in the steady state, or during the dynamic simulation. The efficiency depends on the time taken to get the stationary operating point in advance of each simulation. One way of increasing the efficiency is to calculate and save steady-state solutions to a library in advance and use these to initiate the simulation model during the batch run. For efficient batch mode simulation, requirements S1, S2 and S4 in Table 4 are the drivers.

Scale up issues

Development programs generally try to minimize the number of models of a system due to the significant administration effort that is needed and the risk of needing to handle mistakes following from multi-model handling. The challenge is to reuse and combine component models in a system-of-systems context, without requiring too many parameter settings and consequently too long execution times. So on the one hand there should be reusable and configurable (detailed) models, appropriate for both small and large scale usage, and on the other a sufficiently detailed, or dedicated model that makes it easier for the user to set up and execute the simulation at a certain level.

One way to avoid several different models is to use the Multi Level Approach reported in [Kuhn 2008]. With the Multi Level Approach, switching between model levels can be done, from example:

- a simple and fast model for energy consumption design
- a detailed model for fast network stability analysis
- a detailed model for network quality assessment by increasing of the equation complexity in the model components

There are of course constraints to using the Multi Level Approach: for instance, the interfaces definitions for a model must be the same at all levels of detail (viewed as a black box).

Configuration principles

Challenges concerning configuration management are somewhat different and in some ways more difficult for a simulation system than for the product the simulation model represents. This is due to the possibilities to let every sub-model be configurable to represent several variants of the real component. Example: In the Gripen case, a module for calculation of weight, center of gravity and inertia has to manage data for both the one-seater and two-seater versions, as well as all possible payload configurations.

Configuration is managed in several stages and there are typically some mechanisms for (variants) configuration in large-scale simulation environment of models and simulator platforms. Both hardware and software components are part of the simulator configuration activity and predefined rules must exist to support variant handling and on-board configuration of the system software according to the type of equipment installed.

The need is resolved by switching techniques; model-time switch, compile-time switch or run-time switches. Switching during model-time creates different source code variants, and switching during compile-time creates different object codes. Run-time switching is the most flexible alternative and enables the simulation engineer to change configuration "during flight". Which type of switching technique to use for a particular selection case depends, for example, on safety, security, and maintainability factors. For all those techniques, structured configuration tables similar to those described in IMA section improve the efficiency and quality of scalability in the simulation/development process.

6.3 Tool management strategy

There is a trade-off between, on the one hand, tightly integrated method-tool-chains, and on the other separate tools that each provide state-of-the-art engineering support in its discipline. The integrations may be of the peer-to-peer kind, connecting one tool to another for one or two-way information transfer. Or it may be more of a platform type where several tools are plugged in to a predefined framework, such as the open development platform [Eclipse]. In this section some strategic choices are discussed; how tight to integrate, in what way, and what technology to use, because this sets the possibilities, limitations and cost levels for long time ahead. One solution is to rely on mature standardized interfaces rather than tool vendor specific interfaces and add-ons which are more likely to change over time.

6.3.1 Tool selection criteria

The following parameters form a basis for the decision on selecting a method for model based systems engineering. Ideally, the modeling method shall:

- Have a wide market penetration and be standards-based.
- Provide capabilities for seamless transition from systems engineering to specialty engineering.
- Have multiple commercial tools supporting the language/notation.

In [Friedenthal 2008] there is a list of selection criteria for a SysML tool containing for example:

- Conformance to SysML specification
- Document generation and export capability
- Conformance to XMI and AP233
- Integration with other engineering tools (including legacy tools within an existing system development environment)

    - Requirements and Configuration Managements tools
    - Engineering analysis and performance simulation tools
    - Software, electrical modeling and mechanical CAD tools
    - Testing and verification tool
    - Project management tool
- Performance (number of users, model size)
- Model checking
- Training, online help, and support
- Availability of model libraries (e.g. SI units)
- Life-cycle cost (acquisition, training, support)
- Vendor viability

Similar criteria from Saab and the Skeldar project are described in Paper [V]. IBM/Telelogic Rhapsody was the selected tool for the Skeldar project a decision that was partly based on experienc from an earlier avionics program at Saab Aerosystems with software design and documentation using Rhapsody and UML. The introduction of UML/SysML and selection of Rhapsody in the Skeldar project was based on an analysis of:

- Need for systems engineering support in the product/project (e.g. modularity, variants handling)
- Existing best practice and tool providers/vendors relations in previous projects
- Benchmark of existing state-of-the-art systems engineering support tools and methods
- Provision of systems engineering support in a longer term, throughout the system life-cycle
- The strategy for future partners and consultants

The tool selection criteria should not be based too much on technical aspects (e.g. export capability or model checking) because these functions/features may be built into the tool in future releases. More important is that it has a sound underlying structure and open interfaces that conform to standards.

6.3.2 Long term considerations

A fighter aircraft may be operational for 30-40 years with sufficient upgrades and civil aviation programs plan for an even longer lifetime. In a product with such a long lifetime the tools may add unwanted extra costs to the system's overall life-cycle costs due to support and extra integration work. For example, updating a specification tool to a new version may lead to different (not exactly the same) models or reports being generated. If the tool is used for development of safety-critical airborne systems, the tool update will require extra review, verification, and perhaps also system test. In large systems, these activities can lead to considerable costs.

Not updating the tools as new versions are released will also add cost in the long term. Examples are when an operating system (OS) (or a version thereof) is obsolete and needs to be upgraded, but some tool does not work in the new OS version. One way to solve such an incompatibly problem is to emulate (to run virtually) the old OS in a modern OS. An old VAX computer with a VMS operating system can, for example, be emulated, which solves the problem technically but adds extra cost for procurement, maintenance, and competence for yet another software component.

Paper [IV] contains a study of three different approaches for model based software development with Simulink. The study indicates that a strategy with Simulink models detailed enough to generate high quality software is costly in the long term. This is valid if the tool is updated in the engineering environment with every release from the vendor. Consequently, a method that relies on a tool for generating code is vulnerable to tool changes.

6.4 Organization & responsibilities

As an aircraft integrator or OEM (Original Equipment Manufacturer) it is always convenient to have tier-one suppliers capable of supplying models with sufficient level of detail at the right time for usage in the project. When this is not the case, a company-internal or third party modeling and simulation team has to provide mock-ups, environment models, simulators, and test rigs with models developed “in-house”, but with knowledge from the supplier, through specifications/design documents and preferably also through close cooperation.

In a specific project there are a number of choices to make regarding the formalism to be achieved in the different engineering phases. These include;

- Baselines and change control
- Review level of interfaces, models and documents.
- Review format; how much of the model information is to be exported to documents

Explicitly appointed roles/responsibilities needed within simulation model development, integration and management are the following, with reference to Figure 44:

- Model specification; a role that specifies the requirements, interfaces, and behavior of a simulation model based on the simulation needs and the system specification.
- Model implementation; this role is responsible for implementation of a specified model into a specific simulation platform
- Simulator platform management; for every simulator, the simulation facilities and functions, model configuration, and related documentation are baselined, verified, and released for wide usage in the development project.

- Simulators general manager; responsible for planning of updates as well as verification and release procedures for the set of simulator platforms (Systems Engineering activities for the Enabling Products simulators). One way to gain control of this rather large area is to head the planning and control meetings with the other roles present.

For smaller models and simulators it is convenient to combine the model specification and model implementation roles as deemed practical.

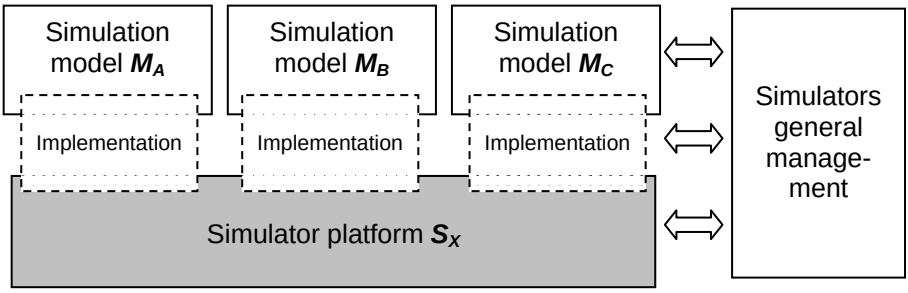


Figure 44. Simulation models and simulator platform responsibilities

This way of organizing the work supports reuse of models between simulator platforms and increases the possibilities of consistent model specifications and models with good quality.

Where possible, every simulation model should be developed to fulfill requirements from all the simulation platforms where the model is implemented. There are, however, situations where the platform's specific requirements are very different, so different models of the same system component have to be managed. Variants and revisions of all models for each simulator platform have to be kept under control in a Product Lifecycle Management (PLM) system, as an enabling product, in alignment with the end product.

7
Discussion and Conclusions

IN THE ACADEMIC world there are a vast number of different modeling methods with supporting tools which are evaluated by using comparably small examples. These are usually applied to real engineering problems in order to evaluate their practical use, but are normally demonstrated by relatively small or well suited examples (compared to the industrial cases) and they are not included in a context or environment of several methods and means. In industry, the terms tool set, tool chain, and methods chain are nowadays frequently used to define an engineering environment with some methodology, integrated tools supported by usage descriptions. The discussion includes scalability of some of the studied modeling techniques and how far it is reasonable, in the coming decade, to plan for MBSE and MBD implementation in an aircraft and avionics development company.

7.1 Discussion

For an industrial Systems Engineering focus on MBD, domain specific modeling techniques and means as such are not central, but the overall workflow, as indicated by this quotation:

> *Typically, there is no single tool that addresses all these issues, and, therefore, a suite of tools is used throughout the design process. Because these tools hardly ever are compatible, the sharing and coordinating of information flow between project teams inevitably leads to a lot of overhead in terms of collaboration, and is very error prone, inefficient, and expensive. Moreover, similar tasks may be carried out multiple times and even simultaneously.* [Vangheluwe 2000]

It is the integration, collaboration and scale-up aspects of MBD that add most value to aircraft and avionics development projects, and these are also discussed below.

7.1.1 Reuse and libraries

To reuse things between products or teams is obviously efficient and valuable for reducing uncertainty/risk and cost. Reuse of software is a growing trend with all kinds of open source communities, but also reuse of embedded functions or parts of code (legacy) in the aerospace and automotive industries is in focus. Even more important here is to reuse the designs: the design is needed to enable the engineers to understand functionality and maintain systems in the long run. This is a strong driver for model based design/development which forces the design to be captured in a structured and maintainable way. For efficient reuse the process must encourage storage of the good, well-verified, and documented (reusable) components in a functioning, well-known storage place; the design library. A shift of focus, from libraries of implementations, to libraries of designs is therefore recommended.

One drawback with reuse is that a reused component may not fit the purpose exactly. It usually has extra/more functionality built in, but which is not needed in the specific instantiation. One may say it adds “dead functionality”, which necessitates extra verification activities. For reuse there is generally thus a trade-off between savings in terms of reuse and cost in terms of verification.

7.1.2 Hardware and software in MBSE

The Hosted Simulation technique was evaluated and is generally a very promising approach, as shown in Paper [II]. For specifying an ordinary aircraft subsystem model containing the hardware – sensor – software – actuator loop, SysML is an appropriate language. Experience, however, shows that there is lack of support for physical flows. In many situations it is not visually clear in a SysML diagram if a flow is an actual physical flow (e.g. fuel flow) or information in the system *about* the flow.

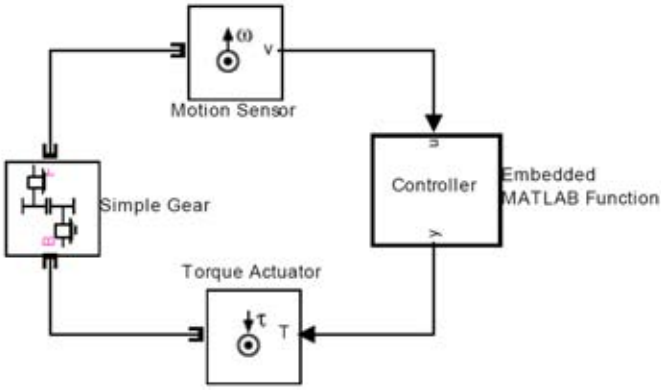


Figure 45. Connectors for physical flows (to the left) and information flow (to the right)

Distinction between physical and information flow connectors such as those in Modelica or Simscape based diagrams, for example, as shown in Figure 45, is on the wish list for future SysML extension.

One proposal to integrate SysML with simulation oriented modeling in Modelica, including equation and simulation diagrams, called ModelicaML is reported in [Akhvlediani 2006]. From an industrial large scale MBD perspective, the value of integrating these two views is tempting. In practice it might evolve tools that support both SysML and Modelica, which is good, but will probably be hard to scale up, because it does not support the separation of concerns gained with the layered approach and building block structure. It looks tempting to use only one model capable of handling requirements, use cases and system structure as well as equations, variables and simulation set-up, as shown in Figure 46.

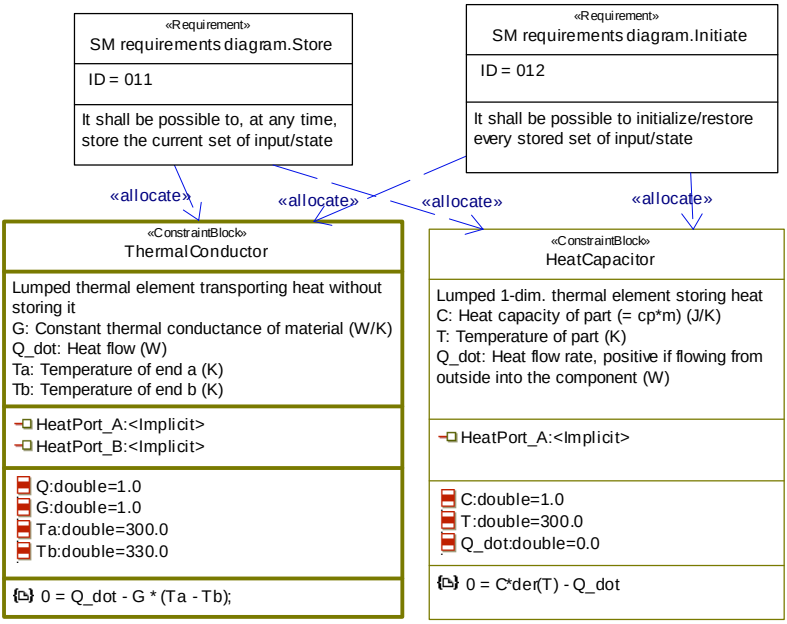


Figure 46. A SysML Block Definition Diagram example of two reusable simulation components with allocated requirements.

The figure shows requirements and two blocks defining simulation components in SysML. Lessons learned from software development show that a separation strategy is preferable when the system and complexity scales up. Storing the requirements in a model/tool appropriate for requirements and behavior models in an execution/simulation oriented tool is to be preferred in the long run, to support all phases of the product family lifecycle. Traceability between the model objects and the requirements needs also tool support and a way of implementing the traceability links, for ex-

ample, storing the unique requirement identifiers in trace/reference (alternatively in comment or annotation) attributes of related model components.

One stage where an integrated approach might be of interest is in the early concept evaluation phase, where the number of advanced tools should be minimized, and it could be a complement to the normal “spread-sheet technology” used today.

7.1.3 Long term effects

Advanced engineering environments are threatened by tool (or vendor) Lock-in, i.e. situations where tool vendors provide few or no means for information exchange with tools not provided by the vendor. Although, for example, MDA was conceived as an approach for achieving (technical) platform independence, current MDA vendors have been reluctant to engineer their MDA toolsets to be interoperable. Such an outcome could result in vendor lock-in for those pursuing an MDA approach. The same applies for other modeling domains such as multiphysics modeling and simulation, where development of the Modelica language is a driver for openness and tool interoperability. Vendors with a considerable market share in its segment, however, choose a proprietary policy such as The MathWorks with its Simscape language, which is not Modelica compatible.

Based on some of the trade-off problems/relations within development of complex products, as described in this thesis, the “paradox of model based development” may be defined as:

1. To solve the problem of complexity, introduce a model to be able to (by analyzing the model) better understand the needs, constraints, and design effects.
2. One model is not enough, because different aspects need to be analyzed. This requires different analysis methods, based on different modeling techniques (or meta-models) and also supported by powerful tools.
3. For each kind of tool, a method, a support team, user guidelines, and education (and maybe a process) are needed.
4. The introduced set of methods, tools, and required skills add more complexity to the development work.

This paradox clearly shows that there is (in general terms) an optimal level of MBD. For some modeling domains the modeling paradigm is more mature (e.g. CAx or control engineering) and it is natural to choose a method that relies to a high degree on models. Nevertheless, the lock-in effects and complicated integrations are still a long term issue, and should be carefully considered in every project’s start-up phase.

7.1.4 Large scale modeling and simulation

When scaling up an advanced work method with connected tools and engineering systems the total process quickly becomes complicated, and it has to be thoroughly analyzed, understood, and communicated. The model based approach has been in focus as a

development means for an aircraft with its components, but not (in this thesis or in industry) to develop the engineering systems (the enabling products). But there is also a great potential in modeling of the (modeling) process, for obtaining common nomenclature, clarifying roles, responsibilities, hand-over between disciplines and sequences in the process. The modeling should focus on the change, so that the current ("as is") process may be analyzed and improved to a future one ("to be"). A small example of a process model is shown in Figure 47. This kind of modeling may be referred to as Business Process Modeling (BPM) and has its own community, notation, and tools as summarized in [Hommes 2004]. Using SysML for process modeling in an engineering environment is considered to give value in terms of both process analysis and insight, but also in the fact that engineers and managers get practice in the concepts of modeling and the UML/SysML notations at the same time.

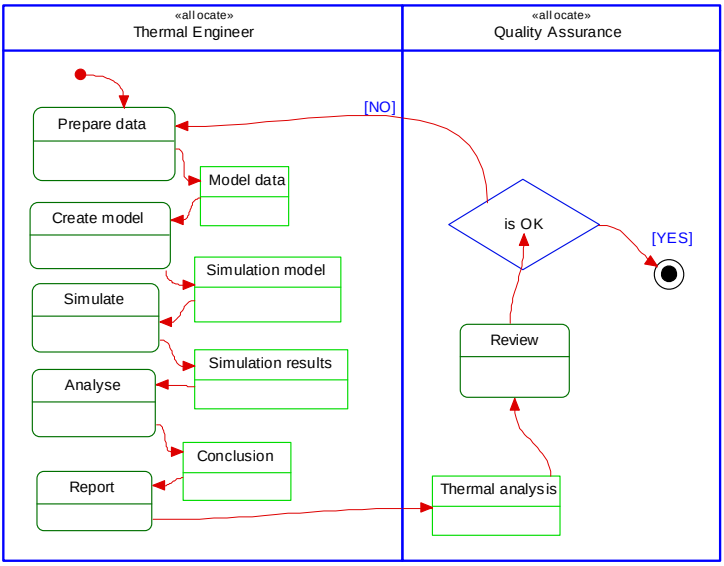


Figure 47. An example of a process model in a SysML Activity Diagram, showing a simulation and analysis task with activities, artifacts and actors involved.

Other aspects and emerging techniques requiring thorough analysis and preferably business process models are:

- Modularization (e.g. IMA and ARINC-653) and configuration of the product family approach.
- Distributed development including distribution of simulation models and modeling guidelines to suppliers and partners.
- Software based simulators, enabling large-scale batch runs during nights, requiring evaluation techniques and tools for large sets of data.

- Multi-core techniques for increased performance of simulation execution, eventually affecting the modeling activity for simulation models and/or deployment of simulation threads/processes on computational resources.

An interesting technique not yet evaluated for large-scale modeling and simulation is the bi-directional transmission line data exchange described in section 4.2.5. With this approach relaying on local solvers, requirements regarding both modularization and increased performance via a simplified multi-core deployment task could be supported.

7.1.5 Human factors

There are several human or social factors not covered in this thesis, but these are most relevant for an industrial scale MBSE and MBD introduction. They include organization of projects and of the competences in the company, and also the (built-in) resistance to organizational change efforts. At a personal level, different engineers have different relations to, for example, abstract/concrete types of development methods. The object-oriented way of thinking appeals some individuals, but far from all. Some prefer graphical representations and other prefer a textual description. In a large-scale environment the chosen methods should not be too “extreme” in any aspect, in order to be understandable and accepted by a majority of engineers.

The question is how to gain acceptance for introducing a different way of working in a (changing) environment which is already complex enough through all the different support systems, databases, operating systems, and log-in/access rights. The overload and stress level for engineers may become unnecessarily high, but with the right level of abstraction and adequate, mature tools supported by a stepwise introduction strategy, I am convinced that the model based approach is the way of the future; because through models, one gains knowledge and a sense of having control of the complexity.

7.2 Conclusion

In this thesis a survey of methods, tools and emerging modeling techniques is described. Some experiences made when introducing MBSE/MBD to support systems development at Saab Aerosystems is also given through the appended papers.

For coming complex products, such as fighter aircraft, civil aviation, or UAV systems, major parts of the development effort will be made in systems and software. Systems Engineering has become increasingly complex, with need to support diverse application fields, different markets and engineering disciplines. For avionics projects the usage of the SysML language has just begun and so far looks promising. But there are several areas where method and tool support must be improved, such as language and tool complexity as well as configuration management support before MBSE with SysML will be the natural method to select in a project developing a complex system.

As engineering to a great extent is the act of balancing (or making trade-offs) between conflicting goals, the design of an engineering methods/tool chain is also a bal-

ancing act with many dimensions. One central tool will not solve all needs, but too many tools tend to add complexity to the development challenge. The trade-off shows that a balanced choice is to restrict the design to one method/tool for specification per modeling domain. For analysis activities there are reasons to provide the engineering teams with powerful analysis means for the specific purpose, but avoiding a too "hard-wired" process integration to prevent lock-in effects.

In summary, the conclusion is that:

- Emerging modularization standards/techniques for avionics systems enables differentiation in design assurance level, development methods, and increased overall flexibility.
- Both method/tools and standards have overlap, which gives freedom in the design of an engineering environment.
- To avoid lock-in effects, integration should rely on standardized interfaces and formats, not on vendor-dependent peer-to-peer tool connections.
- Increased computer performance, matured meta-modeling and database techniques, enables large set of data to be modeled, simulated, and analyzed.
- There are standards for many aspects of avionics development, but there is need for improvements in the area of large-scale simulations. One need is the framework for configuration support from both software configuration management (SCM) and product data management (PDM).
- The semantics of UML and SysML lack the means to clearly differentiate physical objects/flows from information objects/flows.
- Human factors have to be considered both in the design of a method-tool-chain and in the organizational changes. Also, different people/engineers have different relations to, for example, graphical/textual and abstract/concrete types of development methods.
- Process modeling is not in such focus as product modeling, even though the "engineering system" is complex enough to be the subject of MBSE.

7.3 Future work

In the broad area covered in this thesis there are several questions that could be the subject of further research. The natural extension would be to further develop the methods (and standards) for "integrated" design of applications based on modeling approaches, where the term integrated refers to models that can be used to generate models for large-scale simulation purposes as well as for implementation in the real avionics system.

The primary focal areas for future work are outlined below.

7.3.1 Standardization

Examples of needs related to avionics design are definition and standardization of application components. A recently developed framework with definitions and patterns for component development is the automotive standard [AUTOSAR], where parts also apply to avionics. In the area of standards AUTOSAR provides for example;

- Different application interfaces (API) to separate different software layers
- Definition of software data types to be AUTOSAR compliant
- Identify basic software modules with standardized interfaces

Another area where standardization is needed is the fault injection pattern for simulatable sensor components as shown in Figure 33, and in this area there is clearly further work to do within the aerospace and avionics community. SysML, Modelica, and Simulink are equally suited for implementation of fault injection patters.

Definition of other parts of a simulation framework including clear definitions, usage scenarios, predefined patterns and interfaces is planned within the research project CRESCENDO[11]. One aim of the project is to develop the foundations of the Behavioral Digital Aircraft (BDA) paradigm. BDA is expected to become a platform of the simulation domain, covering the entire design lifecycle, from the conceptual design phase to test and verification. From the work in this thesis it is planned to continue in the area of large scale simulation and it is plausible to contribute to the BDA in the area of:

- The Multi Level Approach reported in [Kuhn 2008].
- Interfaces and tool support for the hosted simulation technique.

7.3.2 Configuration management

Current modeling and simulation tools typically provide interfaces for integration with standard software configuration management tools. Such tools are inherently strong in version management, but lack the integrated support found in product data management tools and standards needed for the evolution and maintenance of a set of realized products and systems. Configuration management principles need to be specified for large-scale simulation set-ups within a product family, aligned with principles and concepts from both the software configuration management (SCM) and product data management (PDM) disciplines.

7.3.3 Distinction of information and physical flows

Experiences show lack of support for explicit and clear specification of both information and physical flows in the same diagram. Simulation oriented tools as Dymola and Simulink/Simscape with power port connectors has this support, but in a system-level specification diagram in SysML it is not visually clear if a flow is an actual physical flow or information in the system *about* the flow. SysML needs, clearly in my view, to be complemented with the semantics and graphical power of power ports.

[11] CRESCENDO is a research project within the European Commission 7th Framework Program

References

Akhvlediani, D., (2006). “Design and implementation of a ‘UML profile for Modelica’/SysML”, *Department of Computer and Information Science, IDA, Linköpings universitet*, ISRN: LITH-IDA-EX-06/061-SE, Linköping, Sweden

Alford, M. et al. (1992). “Improving the Practise in Computer-Based Systems Engineering”, In *Proceedings of the 2nd annual Symposium of the National Council on Systems Engineering*, INCOSE

Andrews, L.C., (1986). *Elementary partial differential equations with boundary value problems*, Academic Press Inc., Orlando, Florida, USA

ANSI/EIA-632, (1999). *Processes for Engineering a System.* American National Standards Institute, ANSI.

AP233, *Industrial automation systems and integration - Product data representation and exchange - Part 233: Systems engineering data representation*, ISO/CD 10303-233, International Organization for Standardization.

ARINC 653, (2006). “Avionics Application Software Standard Interface”, *ARINC 653, Specification, Part 1-3, Aeronautical Radio Incorporated*, Annapolis, Maryland, USA

ARP 4754 (1996). “Certification Considerations for Highly-Integrated or Complex Aircraft Systems”, *SAE International*

AUTOSAR, *Automotive open system architecture*, www.autosar.org, Retrieved on (2009-02-05)

Backlund, G., (2000). *The effects of modeling requirements in early phases of buyer-supplier relations*, Thesis, ISBN 91-7219-676-9, Linköpings universitet, Sweden

Beck, K., (2000). *Extreme Programming explained: embrace change.* Addison-Wesley, Reading, Mass. USA

Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., and Hendriks, M., (2006). “UPPAAL 4.0”, In *Proceedings of the 3rd international Conference on the Quantitative Evaluation of Systems*, IEEE Computer Society, Pages 125-126, Washington DC, USA

Blackenfelt, M. (2001). *Managing complexity by product modularisationmodularization*,. Dissertation,Doctoral Thesis. Department of machine design, Royal Institute of Technology,. Stockholm,. Sweden.

Braspenning, N.C., Bortnik, E.M., van de Mortel-Fronczak, J.M., and Rooda, J.E. (2008). ”Model-based system analysis using Chi and Uppaal: An industrial case study.” *Computers in Industry*, Volume 59 , Issue 1 Pages 41-54, ISSN:0166-3615

BridgePoint, *Mentor Graphics*, www.mentor.com/products/sm/model_development/bridgepoint, Retrieved on (2009-02-16)

Carloni, L.P., Passerone, R., Pinto, A. and Sangiovanni-Vincentelli, A.L., (2006). “Languages and Tools for Hybrid Systems Design” *Foundations and Trends in Electronic Design Automation*. Vol. 1, No. 1-2, pages 1-193, Now Publishers Inc.

Cassandras, C.G. and Lafortune, S., (1999). *Introduction to Discrete Event Systems*, Springer, New York, USA

Cellier, F.E., Otter, M. and Elmqvist, H., (1995). “Bond Graph Modeling of Variable Structure Systems”, in *Proceedings of Second International Conference on Bond Graph Modeling and Simulation ICBGM'95*, pages 49-55, Las Vegas, USA

Dassault Systems (2006). www.3ds.com/cn/news-events/press-room/release/1220/1/, *Dassault Systems Press release*, Paris, France, Retrieved on (2009-02-12)

DOORS, *IBM/Telelogic*, www.telelogic.com/products/doors, Retrieved on (2009-02-08)

Douglass, B. P., (2007). *Real-time UML workshop for embedded systems*, Elsevier, Boston, USA

Dymola, *Dynasim AB*, www.dynasim.se, Retrieved on (2009-02-08)

Eclipse, *The Eclipse Foundation*, www.eclipse.org/org, Retrieved on (2009-02-01)

Engel, A., Winokur M., Döhmen, G. and Enzmann, M., (2008). “Assumptions / Promises - Shifting the Paradigm in Systems-Engineering”. In *Proceedings of the 18th annual International Symposium of the International Council on Systems Engineering*, INCOSE, Utrecht, The Netherlands

Esterel, www.esterel-technologies.com, Retrieved on (2009-02-08)

Fernandes, J.M. and Lilius, J., (2004). ”Functional and object-oriented views in embedded software modeling”, *Engineering of Computer-Based Systems, Proceedings of 11th IEEE International Conference and Workshop*, Pages: 378 – 387

Focal Point, *IBM/Telelogic*, www.telelogic.com/products/focalpoint, Retrieved on (2009-02-08)

Fredriksson, B., Holmberg, G. and Lilliecreutz, J (2002). “Collaborative Product Development Supported by Modelling and Simulation”, *NATO Applied Vehicle Technology Symposium on “Reduction of Military Vehicle Acquisition Time and Cost through Advanced Modelling and Virtual Simulation”*, Paris, France

Friedenthal, S. et al., (2000). “Adapting UML for an Object Oriented Systems Engineering Method”. In *Proceedings of the tenth annual International Symposium of the International Council on Systems Engineering*, INCOSE

Friedenthal, S., Moore A. and Steiner R., (2008). *A Practical Guide to SysML: The Systems Modeling Language*, Morgan Kaufmann Publications, USA

Fritzson, P., (2004). *Principles of object-oriented modeling and simulation with Modelica 2.1*. IEEE Press/Wiley, New York, USA

Gavel, H., (2007). *On aircraft fuel systems: conceptual design and modeling.* Dissertation, Linköpings universitet, Sweden

Harel, D., (1987). "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, Pages: 231-274

Herrmann, D.S., (1999). *Software Safety and Reliability: techniques, approaches, and standards of key industrial sectors*, IEEE Computer Society, Los Alamitos, California, USA

Herzog, E. and Pandikow A., (2005). "SysML - an Assessment", *Proceedings of the 15th annual International Symposium of the International Council on Systems Engineering*, INCOSE

Hilderman V. and Baghai T., (2007). *Avionics Certification: A Complete Guide to DO-178 (Software), DO-254 (Hardware)*, ISBN 978-1-885544-25-4

Hommes, L. J., (2004). *The Evaluation of Business Process Modeling Techniques*. Dissertation-Doctoral thesis., Technische Universiteit Delft, The Netherlands.

Hull, E., Jackson, K. and Dick, J. (2002). *Requirements engineering*, Springer, London

INCOSE (2007). *Systems Engineering Handbook - A Guide for System Life Cycle Processes and Activities, version 3.1,* International Council on Systems Engineering.

Johansson, O. and Krus, P., (2006). "Configurable Design Matrixes for Systems Engineering Applications", In *Proc of IDETC/CIE ASME 2006 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, USA

Johansson, O., (1996). *"Development environments for complex product models"*, Dissertation, Linköping studies in science and technology. ISBN: 91-7871-855-4, Linköping University, Sweden

Krus, P., (2005). "Robust System Modelling Using Bi-lateral Delay Lines", *Simsafe 05*, Linköping University, Sweden

Kuhn, R. (2008). "A multi level approach for aircraft electrical systems design." *Sixth International Modelica Conference*, Vol. 1, pp 95-101, Bielefeld, Germany

Larses, O., (2005). *Architecting and Modeling Automotive Embedded Systems*, Dissertation-Doctoral Thesis, Department of Machine Design, Royal Institute of Technology, Stockholm, Sweden

Lavi, J.Z. and Kudish J., (2004). *Systems Modeling and Requirements Specification Using ECSAM: An Analysis Method for Embedded and Computer-Based Systems*, IEEE Computer Society, Washington DC, USA

Lichtenberg, S., (2000). *Proactive management of uncertainty using the Successive Principle*, Polyteknisk Press, Copenhagen, Denmark

Lilliecreutz, J. and Holmberg, G., (eds.) (2000). *Constructive Lean - A Systems Integrators Perspective. The Relationships of Tomorrow Between Systems Integrators, Customers and Their Major Suppliers In the Aerospace Industry Findings from LARP part I.* Research Report from Linköping University and Saab AB, Linköping, Sweden

Long, J, Baker L Jr. (1996). "Specifying A System Using ERA Information Models". In *Proceedings of the sixth annual International Symposium of the International Council on Systems Engineering*, INCOSE

Magic Draw, *No Magic, Inc.* www.magicdraw.com, Retrieved on (2009-02-16)

Mar, B., (1991). "Back to basics", In *Proceedings of the second annual international symposium of the National Council on Systems Engineering*, INCOSE

Matharu, J., (2006). "Reusing safety-critical software in aviation", *Electronics systems and software*, vol. 4, issue 1, pg 32 -35

Matlab, *The MathWorks*, www.mathworks.com/products/matlab, Retrieved on (2009-02-08)

Mellor, S,J. and Balcer, M. J. (2002). *Executable UML: a foundation for model-driven architecture.* Addison-Wesley, Boston, USA

MIL-STD-498, (1994). "Military Standard, Software development and documentation", *Department of Defense*, USA, Retrieved from www.everyspec.com/MIL-STD (2009-01-22)

Modelica, *The Modelica Association*, www.modelica.org, Retrieved on (2009-02-08)

Moir, I. and Seabridge, A., (2004). *Design and development of aircraft systems: an introduction.* Professional Engineering Publishing, London, UK

Muller, G. J., (2004). *CAFCR: A Multi-view Method for Embedded Systems Architecting; Balancing Genericity and Specificity*, Dissertation. ISBN 90-5639-120-2, Amsterdam, The Netherlands

NASA Systems Engineering Handbook, (1995). *National Aeronautics and Space Administration*, USA

Oliver, D. W., Kelliher, T. P. and Keegan, J. G. (1997). *Engineering Complex Systems – with models and objects*,. McGraw-Hill, New York, USA

OMG (2005). "Semantics of a Foundational Subset for Executable UML Models", *Request For Proposal Object Management Group*, OMG Document: ad/2005-04-02

Peak, RS, Burkhart, RM, Friedenthal, SA, Wilson, MW, Bajaj, M, Kim I, (2007). *Simulation-Based Design Using SysML* - Part 1: A Parametrics Primer. INCOSE International Symposium, San Diego, USA

Quine, W.V.O., (1981). *Theories and things.* Harvard U.P. Cambridge, Mass. USA

Rhapsody, *IBM/Telelogic*, www.telelogic.com/products/rhapsody, Retrieved on (2009-02-08).

Rieder, M., Steiner, R., Berthouzoz, C., Corthay, F., and Sterren, T., (2006). "Synthesized UML, a practical approach to map UML to VHDL", *Second International Workshop on Rapid Integration of Software Engineering Techniques (RISE)*, Heraklion Crete, Greece

RIF (2008). "Recommendation - Requirements Interchange Format (RIF), PSI 6, Version 1.2", *ProSTEP iViP Association*, Darmstadt, Germany

RTCA (2009). *Radio Technical Commission for Aeronautics* www.rtca.org Retrieved on (2009-01-18)

RTCA/DO-178B (1992). *Software Considerations in Airborne Systems and Equipment Certification*. Radio Technical Commission for Aeronautics, RTCA, Washington DC, USA

RTCA/DO-254 (2000). *Design Assurance Guidance for Airborne Electronic Hardware*, Radio Technical Commission for Aeronautics, RTCA, Washington DC, USA

RTCA/DO-279 (2005). *Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*, Radio Technical Commission for Aeronautics, RTCA, Washington DC, USA

Sargent, R. G., (2007). "Verification and validation of simulation models." *2007 Winter Simulation Conference*, pp 124-37, Washington DC, USA

SCADE, www.esterel-technologies.com/products/scade-suite, Retrieved on (2009-02-08)

Shlaer, S. and Mellor, S., (1988). *Object-Oriented Systems Analysis: Modeling the World in Data*, Yourdon Press, Englewood Cliffs, N.J, USA

SimElectronics, *The MathWorks*, www.mathworks.com/products/simelectronic, Retrieved on (2009-02-08)

SimEvents, *The MathWorks*, www.mathworks.com/products/simevents, Retrieved on (2009-02-08)

Simscape, *The MathWorks*, www.mathworks.com/products/simscape, Retrieved on (2009-02-14)

Simulink, *The MathWorks*, www.mathworks.com/products/simulink, Retrieved on (2008-10-26)

SPEEDS, *The SPEEDS consortium*, www.speeds.eu.com, Retrieved on (2009-01-14)

Spitzer, C.R. (red.), (2001). *The avionics handbook*, AvioniCon, Inc. CRC Press Williamsburg, Virginia, USA

Starr, L., (1996). *How to Build Shlaer-Mellor Object Models*, Yourdon Press, Upper Saddle River, N.J., USA

Stateflow, *The MathWorks*, www.mathworks.com/products/stateflow, Retrieved on (2009-02-08)

Stevens, R. et al. (1998). *Systems Engineering: coping with complexity*, Prentice Hall, Europe

Suh, N. P., (2001). *Axiomatic Design Advances and Applications*, Oxford University Press, New York, USA

SysML (2008). "OMG Systems Modeling Language (OMG SysML™), Version 1.1", *Object Management Group*, www.omg.org/spec/SysML/1.1, Retrieved on (2009-01-18)

SystemBuild, *National Instruments, NI MATRIXx suite*, www.ni.com/matrixx, Retrieved on (2009-02-08)

Tooley, M., (2007). *Aircraft Digital Electronic and Computer Systems: Principles, Operation and Maintenance*, Elsevier / Butterworth-Heinemann, Oxford, UK

Ullrich, K.T. and Eppinger S.D., (2000). *Product Design and Development, 2nd Edition,* McGraw-Hill, Inc., New York, USA

UML (2007). "OMG Unified Modeling Language (OMG UML), Infrastructure/Superstructure, V2.1.2" *Object Management Group*, from www.omg.org/spec/UML/2.1.2 Retrieved on (2009-01-18)

UPPAAL, *Uppsala University and Aalborg University*, www.uppaal.com, Retrieved on (2008-12-04)

Valerdi, R., (2008). *The Constructive Systems Engineering Cost Model (COSYSMO): Quantifying the Costs of Systems Engineering Effort in Complex Systems*, VDM Verlag

Vangheluwe, H.L., (2000). *Multi-Formalism Modelling and Simulation*, DissertationDoctoral Thesis, Faculty of Sciences, Ghent University, Belgium

Watkins, C. B. and Walter, R., (2007). "Transitioning from federated avionics architectures to Integrated Modular Avionics," *26th Digital Avionics Systems Conference. DASC '07. IEEE/AIAA*

Weilkiens, T., (2008). *Systems engineering with SysML/UML: modeling, analysis, design,* Morgan Kaufmann OMG Press/Elsevier, Amsterdam, The Netherlands

Wymore, W., (2002). "A System Theoretical Framework for V & V", In *Proceedings of the twelfth annual International Symposium of the International Council on Systems Engineering*, INCOSE

XMI (2002). "OMG XML Metadata Interchange (XMI) Specification, V1.2", *Object Management Group*, www.omg.org/docs/formal/02-01-01.pdf, Retrieved on (2009-02-14)