Experience from Introducing Unified Modeling Language/Systems Modeling Language at Saab Aerosystems

Henric Andersson,1, * Erik Herzog,1 Gert Johansson,2 and Olof Johansson3

1Saab Aerosystems AB, Brödern Ugglas gata, SE-581 88 Linköping, Sweden
2Combitech AB, SE-580 15 Linköping, Sweden
3Linköping University, Linköping, Sweden

Received 17 September 2008; Revised 23 February 2009; Accepted 31 August 2009, after one or more revisions
Published online 9 November 2009 in Wiley Online Library (wileyonlinelibrary.com)
DOI 10.1002/sys.20156

ABSTRACT

A Unified Modeling Language/Systems Modeling Language (UML/SysML) subset was the modeling notation selected for an aerospace systems engineering project at Saab Aerosystems. In this paper, the rationale for selecting UML/SysML is given, along with a description of the situation at the project planning stage regarding business conditions, method and tools support. The usage of use case, sequence, and activity diagrams are described as well as definition of functional chains with SysML. Furthermore, the connections to system implementation activities including code generation and simulation are discussed. The advantages and disadvantages of using UML/SysML from experience in an industrial context are reported. It is also described how UML/SysML is related to industrial research projects in the Model Based Systems Engineering (MBSE) methods and tools area. Introducing UML/SysML with a methodology and a supporting toolset in an operative organization require a clear strategy, including planning, just-in-time training, and mentor support. Finally, industrial needs for further development of SysML are discussed.


Key words: Systems Modeling Language; Unified Modeling Language; model based systems engineering; unmanned aerial vehicle

1. INTRODUCTION

Unambiguous specification and design has long been the objective in the development of complex heterogeneous systems. Multiple methods and languages have been proposed in the literature and evaluated in industry, as reported in Stevens et al. [1998] and Mar [1991]. In recent years there has been considerable focus on Model Based System Engineering (MBSE) (also referred to as Model Based System Development) as the vehicle for managing complexity and improving specification clarity and consistency [see Oliver, Kelliher, and Keegan, 1997; Alford et al., 1992; Lykins, Friedenthal, and Meilich, 2000; Wymore, 2002; Long and Baker, 1996]. The acronym MBSE has been used in many different contexts with slightly different meanings. For a control engineer, MBSE might refer to model based specification of, for example, controllers in tools like Matlab/Simulink. For a software engineer, MBSE might involve modeling of software in UML. In fact, modeling has become a standard and an accepted method for the design activity in several engineer-
ing domains. This is in contrast to Systems Engineering (SE) of complex heterogeneous systems where, until recently, system modeling was the exception rather than the rule. This has now changed with the introduction of languages such as UML2 and in particular SysML. See, for example, OMG [2006]. Still there are few publications describing experience gained from introducing systems modeling in large projects.

This paper presents the approach taken and lessons learned when introducing systems modeling (from now on referred to as MBSE) using UML/SysML at Saab Aerosystems. The company develops and produces aircraft systems, and its main product is the JAS 39 Gripen lightweight fighter aircraft. 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 now changing in several ways:

- New product development and production programs with shorter turnarounds are being introduced, such as the UAV program (Unmanned Aerial Vehicle) described in this paper.
- The Gripen program is expanding into a multicustomer, 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.
- Market desire to contract for complete systems instead of parts, for example, delivering integrated solutions rather than providing aircrafts and support equipment under separate contracts.

As a consequence of the changes identified above, there is a drive to improve engineering productivity and quality, and the introduction of MBSE is considered to have high potential. This paper reports on experience gained from introducing MBSE and UML/SysML on the Skeldar autonomous helicopter system. Even if, in a strict sense, a UML2 subset was used for modeling, the results are considered applicable for SysML, as the chosen subset and supporting methods align well with SysML projects.

The rest of this paper is outlined as follows: Section 2 presents the pilot project for MBSE introduction, the Skeldar UAV system. Section 3 presents the palette of improvement initiatives underway at Saab Aerosystems and, further; the rationale for selecting UML/SysML for systems modeling is presented in Section 4. Sections 5 and 6 present the method selected for introducing MBSE and experience gained from the introduction. Section 7 outlines desirable extensions and modifications to SysML to provide better support to systems engineers, and a summary with conclusions is presented in Section 8.

2. UAV DEMONSTRATORS AND PRODUCTS

Saab has limited previous experience of UAV systems and has consequently conducted a series of studies to gain a better understanding of the challenges associated with the development and operation of autonomous air vehicles. The studies conducted include two flying demonstrators prior to the first product—Skeldar. A brief overview of the demonstrators and the Skeldar product is presented below.

2.1. UAV Demonstrators

SHARC (Swedish Highly Advanced Research Configuration) technology demonstrator is used to develop and demonstrate autonomous behavior.

FILUR (Flying Innovative Low-observable Unmanned Research vehicle) is a stealth demonstrator to demonstrate the effect of signature management.

Both SHARC and FILUR were developed using Saab’s experience of model based design gained from fighter development, partly described in Andersson and Sundqvist [2006].

2.2. The Skeldar UAV Product

SKELDAR V150 is Saab’s first UAV product—a short to medium range mobile UAV system that makes it possible to perform take-off and landings without field preparations or extra equipment, and which is suitable for both military and civil applications. Figure 1 presents the aerial vehicle and
ground control station. The modular design allows different configurations such as alternative UAV Control System (UCS) deployment, different ground-segment solutions and to choose among payloads:

**Aerial Vehicle Avionics.** Avionics include redundant computers, Global Positioning System (GPS) receivers, Inertial Measurement Units (IMUs), an air-data system, and a magnetic heading indicator allowing fully autonomous operation.

**UAV Control System (UCS).** The UCS is also a modularized solution, according to NATO standard STANAG 4586 [NATO, 2004], and can be integrated onto different hardware platforms such as trailers or containers. Incorporating air vehicle and sensor operator workstations, the UCS is capable of simultaneously controlling more than one UAV.

**Data Links and Payloads.** Communication between the air vehicle and the UCS is achieved via secure direct links containing sensor and command/control data which are transmitted via separate communication links. The air vehicle is designed to carry a range of payloads such as Electro optical/Infrared (EO/IR), Synthetic Aperture Radar (SAR), and Electronic Warfare (EW) sensors.

### 3. TOWARDS MBSE

A major Saab internal change initiative, 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). With the EMPIRE project, a set of systems engineering techniques/tools were 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. Examples are:

- Graphical modeling and code generation with SystemBuild by National Instruments [2007]
- Software Configuration and Change Management with Dimensions by Serena [2007]
- Requirements Management with DOORS by Telelogic [2007a]
- Software Modeling with UML and Rhapsody by Telelogic [2007b]
- Product Data Management with Teamcenter by Siemens [2008].

The ongoing methods and tools change program at Saab Aerosystems is called MBSE, and it integrates a range of change initiatives. As a means to analyze strengths and weaknesses of different modeling methods and to organize the work in the change program, the modeling tools and related techniques/methods was divided and sorted into modeling domains, as shown in Figure 2. One purpose was to verify that the efforts were broad enough and all domains were covered by appropriate investigations/studies of how to evolve the organization further to achieve efficiency, quality, and attractive engineering environments.

Introduction of UML/SysML discussed in this paper is mainly within the “Usage, Needs & Requirements” and “Architecture & Interfaces” domains, but integration with the other areas is also discussed.

![Figure 2](https://wileyonlineibrary.com)
4. RATIONALE FOR INTRODUCING UML/SysML

The following parameters influenced the decision to introduce UML/SysML in the Skeldar project for support of a model based systems engineering approach. Ideally the modeling method shall:

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

Experience gained from software design and documentation with UML using Telelogic Rhapsody in an earlier avionics program supported the decision. Rhapsody was the tool selected also for the Skeldar project.

The introduction of UML/SysML at Saab and in the Skeldar project was based on analysis of:

- the need for SE support in the product/project (e.g., modularity, variants handling)
- existing best practice and tool providers/vendors relations in previous projects
- the benchmark of existing state-of-the-art SE engineering support tools and methods
- provision of SE support in a longer term, throughout the system lifecycle
- the company strategy for future partners and consultants.

Due to the product strategy, a traditional product lifecycle model is enhanced with a system life cycle model, (see Fig. 3), including the phases of a whole product family seen from a system/software development point of view. An example of a need from the life-cycle step “New variants” and forward is the tool-chain support for variants handling, including onboard configuration of the system software according to the type of equipment installed.

5. APPLYING UML/SysML TO THE SKELDAR PROJECT

The Skeldar project is medium-sized, involving around 100 people. The number of project participants trained in UML and the Rhapsody tool was approximately 50. This section describes, in varying depth, the different activities and artifacts that were covered during the development of the Skeldar UAV system.

5.1. Planning

A concept study was performed prior to introducing UML/SysML, the tool chain, and training activities. From a product point of view, the modeling technique, information model, tool chain, and process adoption is part of the product/system according to ANSI/EIA-632 [AISI/EIA, 1999]. This implies that the Systems Engineering Management Plan and the Software Development Plan are important plans and documents as regards integration of the chosen methodology. The concept study covered a number of aspects to be described and prepared before implementation in the development project. Major issues were:

- Methodology and adaptation to current processes
- Training and mentor strategy
- Configuration Management
- Integration with other tools
- Requirement traceability and documentation strategy.

The reminder of this paper details a few of these aspects.

5.2. Training and Mentorship

Training was considered to be important since most team members were unfamiliar with UML/SysML or even with object-oriented methods. The training was divided into three categories:

- Modeling language basics (UML2)
- Introduction of modeling tool and project specific UML/SysML profile usage
- Just-in-Time (JiT) training.

In the ideal case, all project members should be introduced to UML/SysML, the tool and the modeling principles, before project start or before entering the project (see Fig. 4). In reality, the project had already started before the planning of MBSE training took place. People also enter a project at various points in time and have to wait for a scheduled introduction course. This waiting time before training may partly be compensated by mentor support. Mentor support has the benefit of focusing on the problem area, which is efficient, but there is seldom time to provide the “big picture” of the

<table>
<thead>
<tr>
<th>Phase</th>
<th>Core development</th>
<th>New variants</th>
<th>Enhancements</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main work</td>
<td>Definition, specification, design, implementation and initial production</td>
<td>Variants specification, design and implementation. Production adoption</td>
<td>Rework of system, integration of new functions/features. Obsolescence management</td>
<td>Maintenance and support of system. Corrections. User feedback handling</td>
</tr>
<tr>
<td>Main objective</td>
<td>First product release</td>
<td>Defined product family</td>
<td>Keep product competitive</td>
<td>Keep users satisfied</td>
</tr>
</tbody>
</table>
modeling approach, which may result in limited understanding.

The JiT training was a necessary complement to basic training. Whenever the team entered a new modeling phase, the setup of a seminar provided the basic concepts and tool features for that phase. In combination with mentor support, this gave both the understanding and skill to perform the modeling tasks.

5.3. Used Profile of UML/SysML

In order to have a unified way of describing the product, a restricted set of modeling elements and diagrams was selected (see Fig. 5). Note that only standard UML2 diagrams were selected due to the relative immaturity of SysML tools at the start of the project. Still the diagram selection corresponds closely to a subset of SysML in the sense that:

- Class diagrams were used instead of SysML Block Definition Diagrams to capture the system physical building blocks.
- Deployment diagrams were used instead of Internal Block Diagrams (IBD) to capture system physical interaction and allocation. The UML Composite Structure Diagram is considered a closer analog to the SysML IBD, but deployment diagrams can be used as a restricted form of IBDs.

Of the remaining four UML diagrams used—Activity, Sequence, Use Case and State Machine diagrams—there is a direct counterpart within SysML. By restricting the set of modeling elements, a reduced number of modeling alternatives and a more homogeneous model were achieved. This also had the benefit of a shorter learning period and increased modeling efficiency. In addition to the selection of diagrams, selections of modeling elements for the different modeling phases were indicated in a project methodology document. For example, in sequence diagrams these modeling elements were allowed in the analysis phase (not the same elements as in the design phase):

- Actors
- Objects (System as a Black Box)
- Messages
- Message to Self.

This kind of information was expressed for all allowed diagrams in the different phases. The allowed set of diagrams was also classified as Basic or Optional and documented in the methodology document. An overview of used diagrams versus development phase is given in Figure 6.

As work went on, there was occasionally the need from developers to add an element type to a diagram type. This was discussed among the team and decided by the modeling mentor.

Figure 4. Concept for training and mentor support during the project. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]

Figure 5. Selected profile from the UML2 available diagrams. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]
5.4 Model Structure

Based on the project needs (teamwork, modularity, configuration control, and parallelism) a model package structure was specified, which basically are four views of the system and a generic set of packages for each system component. The model structure (different views) should not be confused with system architecture (different parts); the systems architecture is reflected in one view of the model. The purpose is to guide the team to identify, design, and describe all important aspects of the product. The four views are:

1. **The System Analysis View** presents the required services and behavior of the system considered as a black box. No internal design is described, nor are nonfunctional requirements. Three different modeling techniques are used: Use Cases, State Machine Diagrams, and Functional Chains. The Functional Chain is a system safety concept for describing systemwide functionality (i.e., information flow through the system, from sensors to actuators) and is described by means of text and activity diagrams (see Fig. 7).

2. **The Logical View** presents the technical realization of the system requirements that are expressed in the System Analysis view and other documents. This is done by addressing system components that can realize system services pointed out in the Use Cases or Functional Chains. The diagrams in the Logical View represent different aspects of the system such as:
   - Structure; system components, and their relationships
   - Interaction; behavior by interaction between components (see Fig. 8 as an example)
   - Behavior; dynamics of a single system component.

3. **The Physical View** aims at describing the hardware architecture: for example, general Central Processing Units (CPUs), dedicated electronics equipment, sensors, actuators, and links between them.

4. **The Deployment View** displays how software components are deployed on available hardware.

The Logical view is by far the largest and is further structured in the following categories represented in dedicated packages:

- Internal interaction to realize system functionality, typically sequence diagrams
- Design patterns for commonly used solutions, e.g., communication, data registration, etc.
- Packages for the subsystems to be further designed

![Figure 6. Usage of diagram types in respective development phase.](Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.)

![Figure 7. Functional chain in an activity diagram.](Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.)
• Data types of the engineering kind, interesting from a subsystem interaction point of view
• System verification, methods, test cases, etc., complete or with reference to other documents.

Packages for the subsystem design are further structured in order to prepare for implementation and test, especially for those developed directly in Rhapsody. Each subsystem includes a generic set of packages:

• Design, internal structure, interaction, and behavior
• Interface, provided services (operations), including arguments
• Verification, test drivers, etc.
• Simulator, a simplified and tested version of the subsystem under development
• Requirements, for formal specification of the subsystem.

For parts not further designed in Rhapsody, only the interface, requirements, and test packages are of interest.

5.5. Implementation and Code Generation Strategy

Code generation with Rhapsody is a direct continuation of the detailed design of system components. The software is, however, developed in a separate model for each system component, to allow for clear configuration control between systems and software models and to enable concurrent work in parallel teams. It also allows for different software development methods/tools for different components, e.g., UML/Rhapsody-based, Simulink-based, or traditionally document-driven/hand-coded development. A great deal of the design can be reused by the usage of reference to selected parts of the System Model as depicted in Figure 9.

5.6. Methodology, Rules, and Guidelines

The development methodology and guideline documents prepared in the early phase of the project conforms to traditional object-oriented analysis and design, but is adapted to the system model described above. An overview of the modeling concept is presented in Figure 10. Use Cases, which loosely translate to states and modes found in traditional specifications, are used for requirements analysis. Requirements were documented using text statements, but also in model elements. The modeling methodology is further described in Johansson and Herbertsson [2006].
5.7. Integration with Other Design/Documentation Tools

In a large organization it is of great importance that new methods can be integrated with existing structures. At Saab, DOORS is important for requirements and document management. A characteristic of MBSE is that reports are generated from the models, forming a basis for parts of the documentation. In the Skeldar project, documentation was generated in accordance with standard templates based on MIL-STD-498 [DoD, 1994]. An evaluation of how to integrate/synchronize requirement elements in Rhapsody models and requirements in DOORS with the recommended gateway tool (from the vendor) was performed. The result showed that the tool integration was immature, dependency of three tools available concurrently reduces the overall availability, and more strict rules on baselines and change management are needed. The decision was to handle textual requirements exclusively in DOORS and manage traceability “manually.”

As this evaluation was made early in the project and both UML/SysML and the evaluated tools have further matured since then, an evaluation today may give different results.

6. EXPERIENCE GAINED

Introducing MBSE breaks with existing systems engineering/development practices within the organization. From one perspective, this is positive as previous accepted practices can be challenged; on the other hand, the new process may alienate engineers, leading to a temporary decrease in productivity and quality. Experiences at Saab are mostly positive, especially in the improved communication between systems and software engineers. The following specific experiences gained from the Skeldar project are noted.

6.1. Planning, Training, and Mentor Support

No matter how much UML/SysML and tool training is performed, there is a critical period in any project where engineers are getting frustrated in their attempts to apply the MBSE techniques. This is partly due to the gap in complexity between the comparably simple examples used in training and the size and complexity of the product under development. In the Skeldar project, experienced mentors were engaged to assist developers to overcome such frustration and to ensure that modeling guidelines were applied by the individual developers. The mentors also had the responsibility to verify and approve implementation of tool/method “add-ons,” such as report generator implementation, document templates, and modeling guidelines. By experience, at least two members in the project team need to have experience in development with the use of object-oriented methods, the modeling tool, and large-scale models. Having two experienced persons reasoning about how to plan the modeling approach and how to partition the model, brings stability and safety to the proposed and adapted method. Initially, in a project, it seems desirable to have one experienced mentor for every 5–7 developers (see Fig. 11). Naturally, mentor support can be decreased as developers gain proficiency. With less mentor support, the risk increases for a diverging model and/or insufficient stringency or quality in the model/system.

Skill in modeling comes mainly from practical work and the learning curve to become highly productive seems to be 3–6 months, depending on modeling focus, engineering background, and ambition. A wide range of learning curves was observed; one engineer with a good background in object-oriented methods became highly productive in 1 month. Hence, pilot projects for MBSE introduction should be allowed to have longer calendar time allotted, until the first increment of system analysis and design has been completed. For following increments the calendar time can be “paid back” through
increased efficiency gained by the MBSE approach. With this initial investment in the “core development” phase, as defined in Figure 3, we believe that there is a great potential for cost and time savings in the following phases.

6.2. Project-Specific Guidelines

UML and SysML are extensible languages; hence it is tempting to extend the language in order to accommodate information that would be better suited for management using traditional specification methods. Within the Skeldar project there was a period when engineers tried to feed all project information into the model without considering how that information should be used or extracted/printed from the tool environment. As a result, there was a time period when there was a lot of redundant information, and inevitably there were consistency problems. To avoid such suboptimizations, it is important to establish clear rules defining a scope of information to be captured in the model prior to project start.

The UML/SysML language allows the same specification or intended design to be modeled in different ways; thus time must be invested in evaluating alternative ways of modeling the system/software assets of interest. Resources are needed both for developing the guidelines and teaching the selected methods on what diagram types and model elements to use such that consistency can be ensured in the long term.

6.3. Interfaces to Further Development and Engineering Specialties

Handover from the systems engineering team to further development is always a crucial task, with high demands on interfacing and promoting common understanding without misinterpretations. The interface between systems and software engineering is usually in focus as it is regarded as especially error-prone. Experiences from the Skeldar project were that no traditional handover from systems to software engineering was needed since:

a. Both disciplines worked together to set up software-relevant interfaces in the system model.
b. The notation and tool used in systems engineering is oriented towards software engineering.
c. The system model specifications were directly used (referencedIMPORTED) in the SW-models.

However, problems arose to a higher extent between systems engineering and other specialty engineering disciplines. In spite of special training provided for, e.g., safety engineers, the feedback was that the models were very difficult to interpret. A conclusion is that model based projects should be prepared to provide “traditional style” information in the form of documents in order to, at some level, serve specific engineering disciplines when requested.

6.4. Integration into Configuration Management

Configuration management is, of course, an issue in large scale projects. Current UML/SysML 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. Current UML/SysML tools (and the UML and SysML standards) lack standard product data management capabilities that are needed for product configuration management and evolving and maintaining a set of realized products and systems. With its roots in software engineering, it is natural that system specification in UML/SysML facilitates the interface to tools for software development (as this is typically performed in the same tool and stored with the same format as system design). Interfaces to tools used in other engineering specialities are, however, weak.

6.5. Model Management

Tool support for verification and management of artifact relationships does not scale up. Engineers are forced to spend a lot of time for managing the layout of diagrams such that artifact relationships are captured properly on screen. Better user interfaces and more flexible ways of viewing and interacting with large models are needed. For instance, it should be sufficient to let users specify allocation relationship among artifacts, delegating the creation of the actual layout diagram capturing the relationship to the modeling tool.

6.6. Meta-Model and Information Structures

It is not possible to make all stakeholders (of the model content) proficient in interpreting and navigating the model. Consequently, a way of extracting information from the model to traditional documents must be established so that stakeholders/colleagues can view and review model content as effective as with a traditional document approach, without extensive mentor support. One reason for the need of mentor support is the relatively large and complex base (meta-model) of UML and SysML [OMG, 2008b, 2006, 2008a]. The SysML language has its roots in UML and is defined by a subset of the UML meta-model called UML4SysML as a core. This core is further extended in the SysML definition. Figure 12 provides the measurement of the number of defined atomic elements of UML, SysML, and UML4SysML. The total size of SysML (UML4SysML + SysML 1.1) is smaller than the complete UML language, in relative terms 76% counted by stereotypes/classes and 69% counted by diagram types. The measurements in Figure 12 are taken from the XMI-format distributions of UML and SysML [OMG, 2008d, 2007, 2008e].

To master a modeling tool for UML and/or SysML, training and practice must provide some understanding of the definition of these model elements, how they relate, how to apply them efficiently in modeling tasks, but also what not to use of all available features. Even if only a subset of UML/SysML is used in the project, the users—and in particular the mentors—need to have understanding of the underlying meta-model and its implementation in the tool. The fact that SysML is based on UML, and that UML is in turn based on a “merge” of several other modeling notations/standards, makes the meta-model cumbersome and nontransparent.
6.7. Summary of Experience

It is well known that the coordinated introduction of new methodology and tools require endurance and a bolstered support organization. In the Skeldar project this was addressed through methodology definition, extra training, and mentor support, as outlined earlier. Still there was a substantial lag until the development team could be considered effective in the new development environment. Also, as training was primarily focused on the prime users there was a setback when design was presented to engineering specialist reviewers who had little previous exposure to UML/SysML. Clearly, methodology training must be offered to all involved stakeholders in the development work—not just the primary users of the methodology. In this respect the use of UML/SysML was a complicating factor in the interface to systems engineering, just as the systems to software engineering interface was simplified. It is also worth underlining the importance of adequate modeling guidelines clearly describing what information should be captured in UML/SysML and what should be captured using traditional methods. In the absence of such guidelines, users have a tendency to add information to the model just because the possibility exists, leading to information inconsistency and redundancy. This said, the introduction of UML/SysML into the Skeldar project must be considered a success and a model for future use of UML/SysML in SAAB Aerosystems development projects.

7. IDENTIFIED NEEDS FOR SysML IMPROVEMENTS

Current SysML implementations are moving towards the full capabilities defined in the SysML specification. However, further extensions are desirable in order to improve the value of SysML tools.

7.1. Support for Large-Scale Requirements Analysis and Management

The capability to model requirements objects in SysML is useful, but the format for requirements modeling does not scale up, as reported in Herzog and Pandikow [2005]. For full acceptance, SysML tools must provide flexible navigation, editing, searching, and reporting. For efficient visualization and editing of relationships from requirements to design artifacts, user interfaces and documentation formats like the configurable hierarchical design matrices described in Johansson and Krus [2006] are on the wish list.

7.2. Verification Support

SysML inherits the basic UML structures for capturing verification procedures. Since there have been multiple initiatives aimed at extending UML verification support, this may appear sufficient. There is, however, a crucial difference between software verification and system verification. Unlike software, manufactured physical products are not identical. There will be small variations; moreover, wear and tear will change the characteristics of the product. For this reason it is essential that SysML is extended to support representation (and the actual configuration) of a realized system along with the capability to associate verification results.

7.3. Semantic-Presentation Separation

In the early days of object modeling there were a large number of diverging graphical syntaxes, for example, OMT [Rumbaugh et al., 1991] and Objectory [Jacobson et al., 1992; Booch, 2006; Shlaer and Mellor, 1988]. Given the aim of UML—to create a unified language—it is understandable that a tight coupling between UML meta-model elements and their visual appearance on diagrams was selected. This tight coupling has, however, developed into a constraint now that UML is extending into new domains. Consider, for instance, the supported mechanisms for allocation in SysML:

- A stereotyped dependency can be used for capturing allocation of an artifact onto another.
- A dedicated compartment can be introduced to capture the fact that an artifact is allocated to another and conversely that an artifact has other components allocated to itself.
- Swim lanes in activity diagrams can be introduced to capture activity allocation to another artifact.

These three alternative representations can be used to express the same semantic intent, i.e., allocation. However, the semantics is defined for each individual representation instead of having a single semantic representation with three syntactical dialects. This is a consequence of the tight coupling between semantics and syntax in the UML framework. Having local semantics defined for each syntactical element makes equivalence checking of model and model transformations cumbersome. As the UML family continues to extend in scope, the number of similar cases are expected to grow, and, as a consequence, it would be appropriate to separate the definition of language semantics from the definition of syntactical elements used to express the semantics.

7.4. Data Management and Exchange

Current UML/SysML implementations typically provide interfaces to standard software configuration management tools and thus provide version management of model artifacts. There is, however, limited or no support for traditional prod-
uct configuration management, such as variants handling. Design patterns for this have evolved, and standard implementations exist in STEP (ISO 10303) standards, for example, ISO-10303-214 [ISO, 2003]. Further research is required to find a way of leveraging SysML model development with configuration management down to subsystem and object level.

7.5. Tool Functionality Adapted to Systems Engineering Usage

Finally, it is noted that the current generation of SysML tools are implemented directly upon standard UML environments. As a consequence, there are large numbers of elements in the interface that are related exclusively to software engineering as compared to the relatively few elements of interest to systems engineers. Tool usability for systems engineers would increase if software-specific elements could be hidden.

8. CONCLUSIONS

In this paper, the approach taken and experiences made when introducing UML/SysML as a systems modeling language at Saab Aerosystems are presented. For coming complex heterogeneous products, such as UAV systems, major parts of the development effort will be made in systems and software. Systems Engineering has become increasingly complex, with requirements on supporting diverse applications for different markets and engineering disciplines. For the Skeldar project, the introduction of UML/SysML has largely been positive. The main contributing fact for the success is undoubtedly the large training program instigated for the project team. This said there are several areas where method and tool support must be improved, e.g., language and tool complexity as well as configuration management support before MBSE will be the natural method to apply in all projects developing complex systems.

REFERENCES


G. Booch, Object-oriented analysis and design with applications, Addison-Wesley, Reading, MA, 2006.


Henric Andersson is a Systems Engineer with control engineering background at Saab Aerosystems AB. Former work includes simulation-based design of PFBC power plants and development of flight control systems. Henric works currently in the Systems Engineering group responsible for recommending methods and tools for use on Saab aerospace programs, with focus on methods for large scale development. Henric’s key interest and focus is modeling and simulation techniques and tool integration. He is an industry Ph.D. student at Linköping University.

Erik Herzog is a Systems Engineering specialist at Saab Aerosystems AB. He received his Ph.D. at the Department of Computer and Information Sciences at Linköping University, Sweden. His professional interests include development and introduction of systems engineering processes, specification methods, information modeling, and tool integration techniques. Dr. Herzog is active in many INCOSE working groups and has implemented and is managing the INCOSE i-pub system.

Gert Johansson received his Ph.D. in Mechanical Engineering and Computer Science at Linköping University in 1996 on the subject of sensor feedback control of industrial robots. He worked as a professor at Linköping University until 1998, lecturing and conducting research in the field of automation. In 1998 he formed Conformics AB, where he worked with integration of systems engineering tools and information standardization (ISO, STEP) in, for example, the SEDRES EU-project. In 2003 he joined Combitech AB, where he works as a consultant manager and senior systems engineering expert.

Olof Johansson received his Ph.D. in Engineering Information Systems at Linköping University in 1996 and an MSc in Computer Science and Engineering in 1988. He is currently a postdoctoral fellow at the Division of Machine Design, Department of Management and Engineering, Linköping University. Since 2002 he has worked as an industrial consultant in engineering information systems. Since the mid 1980’s he works part time as consultant at ROJTEC, a company specializing in model-driven software development, giving courses, implementing database driven information systems, and serving as mentor in engineering software development.