

SysML – an Assessment

Erik Herzog, Asmus Pandikow
Syntell AB
P.O. Box 10022, SE 100 55 Stockholm
Sweden
{herzog, pandikow}@syntell.se

Copyright © 2005 by Erik Herzog and Asmus Pandikow. Published and used by INCOSE with permission.

Abstract. SysML is a new standard language for system modeling, built on top of the new version of the popular UML and tailored for the specific needs of systems engineers. In this paper we present a brief overview of the SysML along with observations on the language and some challenges facing the systems engineering and tool vendor community.

The purpose of the paper is not to criticize the outcome of the SysML standard but to evaluate its strengths and weaknesses and identify areas for future improvement.

Introduction

SysML represents a refreshing new modeling language specifically developed for systems modeling. It is a new approach within the Systems Engineering community in the sense that it is jointly developed and submitted for standardization by systems engineering experts as opposed to other methods and tools being developed to proprietary standards. This approach provides the same basis for all tool vendors to implement tools based on the language – a good enabler for healthy development of the meager systems engineering tool market. The question is thus: What are the strengths and weaknesses of this new modeling language?

Initially, it was our impression that SysML was closer to being a finalized standard than it in fact was. Since the original submission of this paper two completely updated SysML drafts (versions 0.85 and 0.9) have been published. In many cases the sources of our original negative observations have been corrected, while we have found more appealing areas of the language. These improvements have made completion of this paper much harder than originally anticipated: All easy picks against the language has been removed. As a consequence, we have abandoned our original intention not to participate in the development of the language and we have made a number of minor comments and contributions to the specification.

The objective with this paper is to analyze the outcome of the project and compare it with other methods and tools being used in the systems engineering community. Any criticism shall not be viewed as an attempt to diminish the accomplishments of the group developing SysML, but as indicators to areas of future work, extensions and improvements. After all, would it not be a shame if the perfect systems development languages were finalized so early in our careers?

It should also be noted that the review presented in this paper has been based on published draft documentation only. The standard will potentially be updated, making any observations made herein invalid.

There could potentially be cases where the authors have misunderstood the intent captured in the specification and we would like to take the opportunity to apologize in advance for any such mistakes. However, misunderstandings of this kind can also be indicators for future improvements and clarification.

SysML can be assessed from multiple viewpoints. In this paper, we consider two technical viewpoints – a data management perspective and a method perspective - as well as a business perspective. Identified factors influencing the adoption of SysML and an assessment of SysML position against these factors are also presented.

The rest of this paper is outlined as follows. In the next section, we outline a product data centric view of an enterprise. The purpose of this section is to outline the environment SysML will co-exist in. This is followed by a sampling of papers reporting on the use of object-oriented technology in general, and UML in particular, for different aspects of systems engineering. The purpose of the review is to establish whether there exists consensus in the systems engineering community for a UML extension for systems engineering. The next sections provide overviews of UML 2.0 and SysML respectively and list our observations on the SysML language and proposals for future extension. We also take a brief look on the prospects of SysML being accepted by tool vendors and end-users based on past OMG records before summarizing our findings and concluding the paper.

Product data in the Enterprise

Product data is typically managed in multiple environments within an enterprise. Four partly overlapping environments can be identified for a generic organization:

- The **PDM** (Product Data Management) system captures information about products, product configurations, product relationships and, in recent developments, configurations of produced product instances. The product data managed within a PDM system is normally data related to production of the product, i.e. drawings, production procedures, etc. Recent additions in the form of the PLCS project have provided support for representing individual information on produced systems over time. Other elements, such as requirement and design documentation, are normally handled at document level, as for example software. Development and standardization of PDM information models is driven by STEP (ISO 10303).
- The **Software Engineering configuration management** system captures information on individual software projects. These systems are usually not designed for managing non-software information. Hence, only a coarse view on the non-software world is catered for.
- The **System and software design environments** capture a design view on a system. A design environment normally supports one or more modeling methods, each with a defined semantics. OMG, the Object Management Group, is the developer of the UML language, and hence, the standardization organization for software design product data. In fact, we view OMG as the current major driver behind design environment development due to the high level of detail in the UML specification. Standard UML development environments include code generation facilities. Integration between UML tools and software configuration management tools is a common feature of most UML tools.
- The **Requirements engineering environment** captures information about the characteristics a system shall possess. For any complex system this means the inclusion of some kind of coarse view on a (hardware or software) product structure within the tool. Typically, individual requirements are captured in text with the environment providing capabilities to cross-link individual requirement as objects. Configuration management functions are normally provided within the environments.

We perceive this fragmented situation for data management systems as one of the key obstacles for improving productivity in the systems engineering process. There have been numerous activities targeted at overcoming the restrictions of current systems. A few general trends are listed below.

- The scope of PDM systems are being extended into the system and software engineering domain as illustrated by the work of the SEDRES project (Johnson & Herzog, 2001) and most notably the PLCS standard, which includes basic support for representing requirements and system functionality.
- The scope of UML is extended to provide more detailed support for non-software artifacts. There are also numerous tool environments that provide links to requirements engineering environments.
- A large number of efforts have been undertaken during the last 15 years to bridge the gap between the system and software engineering domains, for example (Alabiso, 1988), (Ward, 1989), (Fichman and Kemerer, 1992), (Pandikow, 2002), which indicates a strong need for integration.
- Requirements engineering environments are extended to provide restricted design support (Dick and Chard, 2004).

One aspect of the SysML assessment is performed in the light of this fragmented situation on the tool market. The question is whether the SysML does decrease the fragmentation described to any substantial extent?

Use of Object-Oriented specification methods for Systems Engineering

Originating from software engineering, the UML is by no means an unknown in the systems engineering community. Numerous reports have been published on the use and adaptation of the UML for systems engineering purposes. In these, two trends can be distinguished: One declaring the abstractions provided by UML as being straightly applicable for systems engineering and another one requiring more or less drastic modifications before making the UML suitable. Among the examples of the former kind are:

- Gonzales (Gonzales, 2001), promoting the use of the unchanged UML for capturing the operational scenarios of a system. The UML notation is proposed for representing system stakeholder views on the system, indicating a position where UML is accepted “as-is” for systems engineering purposes.
- Another similar example is the indirect recommendation of the use of the unmodified UML for system specification by the US department of defense by including UML examples in the DODAF deskbook (DODAF, 2004).

Conversely there are many publications identifying shortcomings and proposing smaller or larger extensions to standard object orientated methods, e.g.,

- Cocks (Cocks, 1999), identifies a number of problems associated with the use of early object modeling techniques for systems modeling, i.e.,
 - Maintaining perspective, early object-oriented design methods lack support for

representation of composite structures.

- Premature architectural decisions, no separation between functional and physical architecture representations. This also makes it harder to perform system trade studies.
- Cluttered diagrams, object-oriented methods support for generalization/specialization constructs are perceived to be difficult to apply to systems engineering problems. The challenge to reuse system components is different for software and systems engineering.
- The OOSEM method (Lykins et al., 2000), the approach of the OOSEM is to extend UML to include functionality to represent data-flow diagrams etc. In effect, OOSEM is a realization of a traditional systems engineering method using UML notation.
- Axelsson's proposal for modeling real-time control systems (Axelsson, 2001), this extension of UML introduces support for continuous variables, equations, clocks, system modes and representation of composite structures.
- Dori's object-process methodology (Dori, 2002) representing a drastic change of standard object-oriented and UML-based concepts.

The sample of publications presented above is an indication that there indeed exists a need for improving support for systems modeling for complex systems – at least when early object oriented modeling methods are applied for systems modeling.

UML 2 at a glance

UML 2 is an extensive language offering support for multiple views on a system and for modeling a wide variety of systems. There are 13 standard views, or diagrams, each with its own dedicated perspective, strengths and weaknesses. A system can be described using several views; however, a single view does only describe a particular part / perspective of the system. A brief overview of each view is presented below. A complete overview is available in standard UML reference books, e.g. (Pender, 2003) or (Eriksson et al., 2004):

- Class diagram, the Class is the core concepts of object-oriented languages. From a computer science point of view a class is an abstract data type coupled with operations for manipulating the data held by the type. The class diagram is the mechanism for displaying classes and relationships among classes, thus providing a view on the static elements of a software system.
- Package diagram, in UML the Package concept provides the mechanism for logically partitioning a large model of a system into smaller manageable parts, e.g., subsystems. The Package diagram primarily supports model management, it allows for capturing relationships between packages, e.g. for stating that elements of a package are used within another package.
- Object diagram, supports the creation of examples (object instances) based on a class diagram. The diagram type is primarily valuable for identification of object relationships and for the definition of test cases.

- Composite structure diagram, the purpose of this diagram is to visualize the elements of a class, including interaction points (ports).
- Component diagram, a representation of software fragments in the implementation environment with focus on interwiring the components. In doing so they may be used to capture the configuration of a system.
- Deployment diagram, provides a view on the hardware (non-software elements) of a system as well as how the software artifacts are to be deployed on the hardware.
- Use case diagram, provides the means for identifying the actors interacting with a system and their system uses as well as interdependencies among the system uses. Use case diagrams are also good for identifying system states and modes, including stakeholders in respective state or mode.
- Activity diagram, provides the means for representing the workflow of a system much like the EFFBD method used extensively in Systems Engineering (Long, 1996).
- Sequence diagram, allows for the representation of messages exchanged between objects over time. Typically, Sequence diagrams are scoped to capture interaction in small scenarios, i.e. one scenario per use case.
- Communication diagram, is similar in scope to the sequence diagram (or vice versa) except the time dimension is not captured in favor of capturing modeled links between objects. Still, the focus is on capturing interaction between objects.
- Interaction overview diagram, provides the mechanism for linking multiple behavior diagrams so that their context is captured. For example, *Behavior A* precedes *Behavior B*. In this respect, the Interaction overview diagram is similar in functionality to the Package diagram.
- Timing diagram, provides the mechanism for capturing system state transitions over time. It may be used for capturing the desired system behavior over time.
- State machine (statechart) diagram, provides support for representing the conditions for how external stimuli can change the state of an object over its lifetime.

UML 2 represents a major extension over the previous UML 1.x versions. The most important new features in the view of Kobryn (Kobryn, 2004) are presented below.

Support for component-based development via composite structures. Three new concepts – part, port and connector – are included and the result is similar to the system physical architecture model documented by Herzog (Herzog, 2004) and by Axelsson (Axelsson 2001). These elements allow the construction of component hierarchies of arbitrary depth.

- Hierarchical decomposition of behavior. There is support for representing structural behavior just as for components. This allows for readable diagrams containing complex behavior descriptions.
- Cross-integration between behavior and structure. In UML 2, it is possible to allow the same element to occur in multiple views providing a better overall understanding over the complete model of a system.

- Integration between action semantics and behavioral constructs. There is now a formally defined language for guard and action statements. This simplifies code generation from a model, although this may be of limited interest for a systems engineer. Still, a formal language also allows for effective simulation of a model.

It must be noted that many of the extensions listed in the previous section against the acceptance of UML in Systems Engineering community has been taken up in UML 2. It is therefore relevant to ask whether there is additional value in a dedicated Systems Engineering language. This question is further addressed in the evaluation section of this paper.

SysML – an Overview

This section presents an overview of the SysML language. The overview is necessarily brief, more in-depth information is available in (Friedenthal and Kobryn, 2004) and the SysML specification (SysML, 2005). SysML can perhaps best be characterized as an extended subset of UML 2. Many of the diagram types supported by UML 2 have been dropped, others have been modified substantially and a small number of SysML-specific ones have been added. An overview of the diagrams in SysML is presented in Figure 1 below.

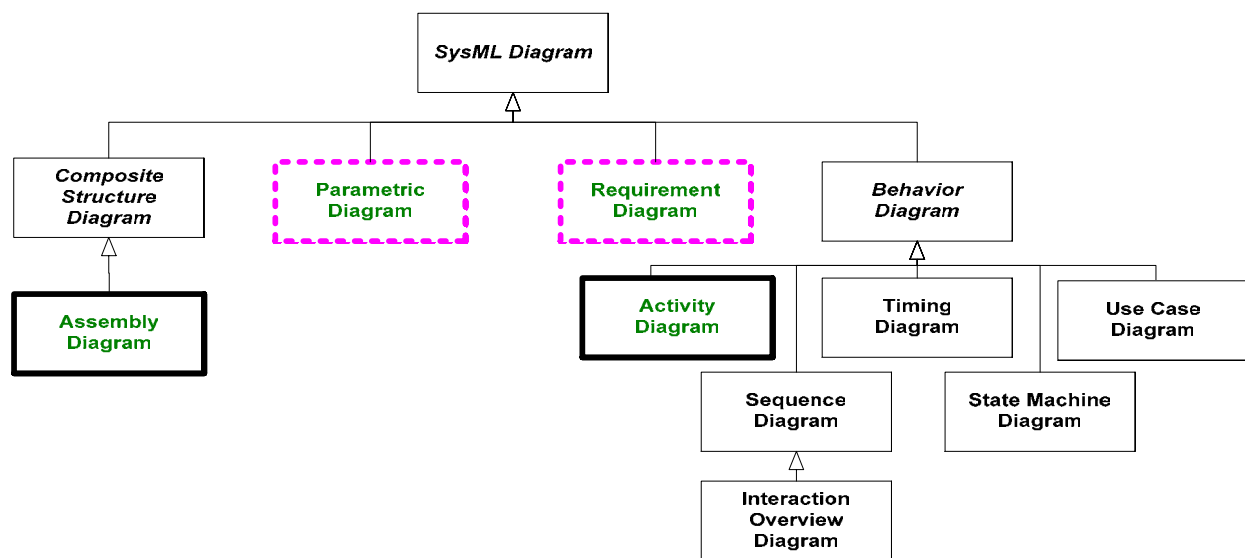


Figure 1, SysML diagram overview

The major extensions made in SysML are listed below:

- Assembly diagram, the assembly diagram is an adaptation of the UML 2 Composite structure diagram. The purpose of the diagram is to capture system components, interfaces and interactions. Interaction is based on items (data, material, energy) being exchanged between components. In this respect it is similar to a traditional physical block diagram. An important restriction compared to the Composite structure diagram is that there is no support for the offered/required interface notation.
- Activity diagram, the extensions made to standard UML activity diagrams include:

- The identification of activity diagram representations similar to those used in the EFFBD notation (Long, 1995). Consequently, it is possible in SysML to model systems behavior in a notation that is compatible to EFFBD. This will facilitate and improve interaction between SysML and traditional systems engineering tools and facilitate the migration to SysML.
- Support for representation of time-discrete and continuous systems.
- Control of the size and behavior of buffers on incoming data flows.
- Parametric diagram, the parametric diagram allows for capturing of engineering analysis (e.g. performance and reliability) models with SysML. These parametric constraints allow for the representation of relationships between properties of different model elements. Standard math symbols are used to capture relationships; however, no formal language is defined for this.
- Requirements diagram, the requirement diagram supports the representation of textual requirement statements, the capture of requirement properties, as well as requirement composition. Requirements may be displayed in a traditional tabular view or in a graphical notation. There are relationships for:
 - Tracing requirements to other model elements for capturing derived requirements and association of requirements to a particular system scenario/state/mode (relationship “trace”).
 - Capturing the fact that a model element satisfies a particular requirement (relationship: “satisfy”).
 - Identifying model elements defining a test (verification) case for determining whether the requirement was fulfilled or not (relationship: “verify”).
- Allocation, contains constructs for capturing that a model element is associated to another model element for guidance for future development activities. For example, may an Activity element be allocated to an element in an Assembly, which normally would imply that the Activity will be performed by the assembly element; however, no such assumptions are implied in the semantics of the relationship.

Overall discussion

In this section a number of high level observations on SysML are presented. The observations are not presented in any particular order. Although SysML is embedded in a framework for object-oriented specification, it is in fact a traditional systems engineering specification language. There is direct support for the four FRAT views identified by Mar (Mar, 1996) as essential for systems modeling, i.e.:

- Functions: Behaviour diagram views.
- Requirements: Requirement diagram view.
- Answer: The Assembly diagram view.
- Test: The identification of test cases.

In this respect, the language is complete, or in other words, it meets the minimal requirements for a systems modeling language. In the different views defined by Mar, and supported by SysML, it is the support for representing system behavior that is of SysML strongest points, whereas the support by the Activity diagram support is most developed. It offers far more powerful modeling primitives than for example EFFBD. The same can be said about the support for representing Statecharts. In addition, the specifications of semantic primitives included in the UML meta-model also suggest that SysML-based modeling tools will inherit the extensive simulation support available in standard UML tools.

However, the modeling power offered by the Activity and Statechart diagrams comes at the expense of a high level of detail and hence, experience and knowledge along with a steep learning curve (but less steep than the one for UML, though) will be required to exploit SysML to its limits.

In the systems engineering community, the explicit support for requirements has been considered to be one of the major shortcomings of the UML. The requirements diagram of SysML is a key addition compared to the UML. It allows a SysML environment to explicitly manage individual requirements as objects as opposed to textual elements. This is a substantial improvement over the current software specification languages including UML, especially the considering the rich set of relationships for capturing allocation and fulfillment of requirements, including the outcome of verification activities.

From a language perspective it is interesting to note that SysML has discarded the class diagram view and does not support non-stereotyped class objects. This view is the one which is closest to source code and of low priority from a systems perspective. Still, it is a sign of quality that it can be removed while maintaining language integrity. In SysML, this concept has been merely replaced with the Assembly concept, which can turn out to improve the acceptance of SysML in the systems engineering community, where the class concept is often associated with pure software systems.

Overall, SysML appears to be complete and provides powerful modeling constructs. Still, we have managed to find a number of detailed issues as outlined in the next section.

Specific SysML observations

This section presents a number of observations, positive and negative, identified from the SysML specification (SysML, 2005). It is interesting to note that the set of observations presented in this section is completely different compared to the set presented in the paper draft. The severity of the observations has also decreased. This is an indication that SysML has matured significantly over the last six months.

Representing the system under verification: A software world legacy is present in SysML in that there is no support for representing produced products/systems. Consequently, there is no built-in mechanism for capturing the configuration of a system being verified. We know that this issue has been brought up recently within the SysML partners (Spiby, 2005) and expect a solution being proposed. Still, it is at present an important view missing in SysML.

Requirement relationships: SysML defines a single binary relationship – “Derive” – that operates exclusively on requirements. The semantics of the relationship is that a new (client) requirement is derived from an existing (supplier) requirement. Additional information may be captured on the relationship documenting the rationale for the relationship. This construct is

present in most publications that touch upon requirements management and systems engineering, e.g. Kossiakoff and Sweet (Kossiakoff and Sweet, 2003), so it may not appear controversial. However, it must be questioned whether the derive relationship is the only type of inter-requirement relationship in use. For example, one could imagine the use a dedicated requirement relationship for identifying alternate (conflicting or semantically identical) requirement statements. For instance, requirement *A* and *B* may conflict with requirement *C*. This kind of relationship does not share the semantics of the Derive relationship and there is no immediate outcome or new requirement created based on the relationship. Moreover, more than two requirement statements are involved in the relationship making it difficult to capture information about the nature of the conflict. Based on this example, we have suggested to the SysML partners that SysML would be more general if:

- The Derive relationship would be modified such that the relationship type could be indicated by a textual attribute;
- The model would be extended to support n-ary (relationships involving more than two requirements) relationships, as this is the case with associations in the UML. It is true that n-ary relationships are usually not recommended as they seldom are supported in implementation environments (Pender, 2003). However, this is not the case for requirements relationships in SysML as they will never be implemented in source code.

Too much freedom for the modeler: There is a fundamental difference between UML and SysML in the sense that UML models for software systems are intended to employ the same concepts during the complete development phases, reflecting the final software. A model captured in SysML, on the other hand, is just an abstraction of the final system. The fidelity of the abstraction in a SysML model will vary depending of the position in the systems engineering process, the type of system being modeled and foremost the ambition of the modelers. In this sense, SysML is foremost a tool for communication intent. The richness of UML that is carried over to SysML through the common ancestry is a potential problem: There are simply so many alternate solutions for partitioning a design using UML. It is easy for a reader of a specification to overlook a critical element of a specification hidden under a rarely used abstraction assuming that those parts of the model actually read and understood is the complete reflection of an author's intent. For instance, it is possible to capture behavior definitions within an assembly object and within a port object defining the interface to the assembly. The specification fragment residing in the port may influence the behavior of the system fundamentally, but may easily be overlooked by a human reader of a model.

Inflexible management of multiple views on single element: In many situations exists a need to capture information about a system element at multiple levels of abstractions. For example, the same version of an element is captured for multiple purposes, e.g. for design and performance analysis. SysML does not provide any construct for collecting these views under a single label. Instead, multiple binary relationships must be used to indicate that the related elements do actually refer to the same element, but at different levels of abstraction.

Extensive support for capturing behavior: One of the most appealing elements of SysML is its extensive support for capturing the behavior of a system. UML's support for finite-state machines and interaction diagrams are adopted without modification. The UML activity diagram is extended substantially to better cope with requirements for representing non-software systems. The modifications are most welcome as they offer far better precisions for capturing the user's

intent with a model.

The only negative observation made, and this is purely academic, are the multiple references to the EFFBD notation available in the ViTech CORE tool. Even though the EFFBD is a fine example of traditional modeling techniques, it is by no means the only notation in use in the systems engineering community. For the sake of objectivity it would be beneficial if either all references to the EFFBD are removed from the standard, or a comprehensive set of modeling methods were included into the documentation.

Prospects for extending SysML

SysML is a major achievement, but by no means the answer to all stakeholder needs in the systems engineering process. There are many potential areas for extensions. Friedenthal and Kobryn (Friedenthal and Kobryn, 2004) suggest extensions to support for evaluation of multiple alternate designs and the introduction of a new logic diagram.

Enhancing explicit support for configuration and version management is another potential extension. UML and SysML appear to be constructed for specifying a single or a small number of systems at a time. Functionality for configuration and version management is not integrated in the language (because it is in many cases added on tool level). This may have been less of a problem with UML 1.x models where the object model was essentially flat, and standard software configuration management tools could be applied. In SysML and UML 2, however, there is explicit support for representing composition structures and hence, also an implicit requirement for maintaining control over which versions of a particular component *X* may be interconnected with a particular version of component *Y*. These configuration control questions are key system issues, where the current UML provides limited guidance. In fact, UML needs to build in such functionality on top of the existing constructs. This is expected to be a major undertaking requiring substantial modification to the UML meta-models (but, incidentally, not on end-user functionality).

In contrast, this is exactly the area where the STEP framework (ISO 10303) is excelling. It appears sensible and desirable to merge the UML meta-model and the STEP framework such that true through-life representation of a product (system) is enabled within a single standardization framework. Technically seen this would not presents any problems as presented by Pandikow (Pandikow, 2001), but it may not be a viable undertaking from a political perspective.

Standardization assessment

In this section, different aspects of the standardization efforts are presented and the SysML performance is analysed according to these aspects. The frame of reference is defined by similar projects, such as AP-233 up to its current status.

Momentum maintenance. Standardization efforts are often initiated by smaller groups of stakeholders with a high level of motivation. Later in the process, this momentum usually decreases, due to the gradual decrease of engagement of the involved participants. Hence, a prerequisite to success is to actively maintain the momentum and the spirit of the involved parties.

SysML: We are impressed with the short cycle-time from the UML for system engineering RFP to the resulting SysML specification. SysML is slated for being available on the market long before public interest has faded even with it being 6-9 months late in the standardization process as compared to the schedule presented by Friedenthal and Kobryn at the 2004 INCOSE symposium (Friedenthal and Kobryn, 2004). SysML is by no means perfect, and there is plenty of room for improvements. However, this is irrelevant as long as the standard is good enough for being actually employed and used.

Tool vendor involvement and support. Early involvement of tool vendors is crucial. Besides experience and know-how, tool vendors can provide the standardization efforts with early respective implementations in tools, enabling verification, test, and marketing of the standard, gaining a competitive edge compared to contenders.

SysML: The number of tool vendors involved in drafting the SysML specification shows that there exists not only a pull from the market, but also a push from the vendor community. This is highly promising for the acceptance and support of SysML. It is also worth noting that SysML will provide vendors with an opportunity for a new product, making it much more vendor friendly compared with infrastructure standards such as AP-233.

Life-cycle concept and continuous improvement. All efforts and their results need to follow a life-cycle concept in order to provide for adjustments and adaptations during the development of the standard. This prevents the efforts from being (partly) outdated during the course of the project.

SysML: OMG has an excellent track record with UML and its stepwise improvement, which most likely will also apply to the development of SysML.

Conclusions

There must be no doubt that we are very impressed by the achievements of the SysML group. The group has in a very short time extended the UML for use by systems engineers, and done so in a way that consensus has been maintained within the group. This is a major accomplishment in an international standardization project.

Technically seen, the scope of SysML is sufficiently wide to capture those four views available in most system specification environments. A number of issues against the language have been captured, but none of them can be categorized as being major.

SysML will allow systems engineers to capture system design in a traditional manner, but yet similar to the notations and abstractions used by software engineers. In this sense, it is expected that SysML will improve interaction between software and systems engineers. The other success factor is that SysML will open the dynamic UML tool market for systems engineers. New tool vendors will present tools directly targeted to systems engineers and tool license costs are expected to decrease substantially.

Having said this, there are still substantial challenges for the systems engineering organizations and tool vendors ahead. SysML may provide a better interface to software engineering. This is an important milestone in the steady but slow integration process among methods used in different engineering disciplines. Nonetheless, there is still a long way to go until a there is integrated tool and infrastructure support available for all engineering disciplines within organizations developing complex technical systems.

References

- Alabiso, B., "Transformation of data flow analysis models to object-oriented design". In *OOPSLA 1988 Conference Proceedings, Special Issue of SIGPLAN Notices*, volume 23 (11), pages 335-353, November 1988
- Axelsson, J., "Unified modeling of real-time control systems and their physical environments using UML", In *Proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 2001
- Cocks, D., "The Suitability of Using Objects for Modeling at the Systems Level". In *Proceedings of the 9th Annual International Symposium of the International Council on Systems Engineering International Council on Systems Engineering*. 1999.
- Dick J. and Chard J., "Combining Requirements, Models and Design", In *Proceedings of the 14th Annual International Symposium of the International Council on Systems Engineering*. 2004.
- DODAF, *DoD Architecture Framework Deskbook*, Version 1.0, DOD, 2004
- Dori, D., *Object-Process Methodology – A Holistic Systems Paradigm*, Springer, 2002
- Eriksson Hans-Erik, Penker Magnus, Lyons Brian, Fado David, *UML 2 Toolkit*, Wiley, OMG Press, 2004.
- Fichman, R.G. and Kemerer, C.F., "Object-oriented and conventional analysis and design methodologies: Comparison and critique", *IEEE Computer*, 25 (10), pages 22-39, October 1992
- Friedenthal S. and Kobryn C., "Extending UML to Support a Systems Modelling Language", In *Proceedings of the 14th International Symposium of the International Council on Systems Engineering*, 2004.
- Gonzales, R. "Completeness of Conceptual Models Developed Using the Integrated System Conceptual Model Development Process". In *Proceedings of the 10th International Symposium of the International Council on Systems Engineering*, 2000.
- Herzog, E., *An approach to Systems Engineering Data Representation and Exchange*. Linköping Studies in Science and Technology, Dissertation 867, 2004.
- Johnson, J. and Herzog, E.. "The Data Standard AP-233: An Invigorator for Global Systems Engineering". In *Proceedings of the 11th Annual International Symposium of the International Council on Systems Engineering*, 2001.
- Kobryn C., and Björkander M., "Architecting Systems with UML 2.0", *IEEE Software* July/August 2003.
- Kobryn C. "UML 3 and the future of modeling", *Journal of Software and System modeling*, Vol 3, number 1 pp 4-8, 2004.
- Kossiakoff, A. and Sweet, W., *Systems Engineering – principles and practice*, Wiley, 2003.
- Long, J., "Relationships Between Common Graphical Representations used in Systems Engineering". In *Proceedings of the 5th Annual International Symposium of the International Council on Systems Engineering*. 1995.
- Lykins, H., Friedenthal, S., and Meilich, A., "Adapting UML for an Object Oriented Systems Engineering Method", In *Proceedings of the 10th Annual International Symposium of the International Council on Systems Engineering*, 2000
- Mar, B. "Back to Basics Again - A Scientific Definition of Systems Engineering". In *Proceedings of the 7th Annual International Symposium of the International Council on Systems Engineering*. 1997.
- Pandikow, A., "Support for Object-Orientation in AP-233". *Proceedings of the 11th Annual*

- International Symposium of the International Council on Systems Engineering*, 20001.
- Pandikow,, A., *A generic principle for enabling interoperability of structured and object-oriented analysis and design tools*, dissertation no. 793, Linköpings Studies in Science and Technology, Linköpings universitet, Sweden, 2002
- Pender, T., *UML Bible*, Wiley, 2003.
- Spiby, P., “Missing support for realized systems”, verbal comment during INCOSE SysML Review, 2005-01-30.
- SysML, *Systems modeling Language: SysML version 0.9*, 2005-01-10.
- Ward, P.T., “How to integrate object orientation with structured analysis and design”, *IEEE Software*, pages 74-82, March 1989

Biography

Dr. Erik Herzog is working as a senior systems engineering consultant at Syntell AB, Sweden. Between 1996 and 2001 he was the principal modeller of the ISO 10303-233 information model as a part of his Ph. D. studies at the Department of Computer and Information Sciences at Linköping University. Erik completed and defended his Ph.D. after a long last in 2004. His interests include Systems engineering, specification methods, process definition and introduction, information modelling and tool integration techniques. Erik is a founding member of the Swedish INCOSE chapter.

Dr. Asmus Pandikow. Dr. Pandikow works as senior consultant at Syntell AB in Sweden. He supports clients in business development, systems engineering, quality and process management with strong focus on tool integration in these areas. Dr. Pandikow has a background of 15 years of software and systems engineering in industry and academia. Before joining Syntell, he worked at the university of Linköping, Sweden, on the proposals for the ISO 10303-233 (AP-233) within the European Union sponsored SEDRES projects. Dr. Pandikow is a founding member of the Swedish INCOSE chapter.