



INSIGHT

What's Inside

President's Corner

Inclusive Thinking for INCOSE's Future 3

Special Feature

Introduction to this Special Edition on Model-based Systems Engineering (MBSE) 7

SysML: Lessons from Early Applications and Future Directions 10

Model-based Systems Engineering for Systems of Systems 12

MBSE Methodology Survey 16

Using Model-based Systems Engineering to Supplement the Certification-and-Accreditation Process of the U.S. Department of Defense 19

Executable and Integrative Whole-System Modeling via the Application of OpEMCSS and Holons for Model-based Systems Engineering 21

MBSE in Telescope Modeling 24

Model-based Systems Praxis for Intelligent Enterprises 34

The Challenge of Model-based Systems Engineering for Space Systems, Year 2 36

Integrating System Design with Simulation and Analysis Using SysML 40

A Modeling Approach to Document Production 44

MBSE for European Space-Systems Development 47

SysML is the Point of Departure for MBSE, Not the Destination 54

Fellows' Insight

Key Issues of Systems Engineering 58

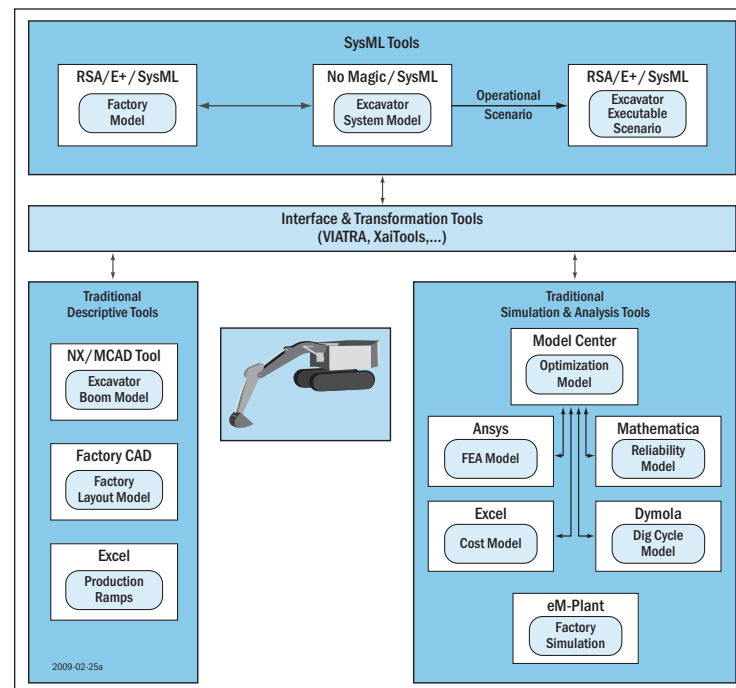
What Are the General Principles Applicable to Systems? 59

Forum

Are You Programmable, Inventive, or Innovative? 65

SPECIAL FEATURE

Model-Based Systems Engineering: The New Paradigm



The Use of Systems Engineering Methods to Explain the Success of an Enterprise 65

Technical Activities

Announcing BKCASE: Body of Knowledge and Curriculum to Advance Systems Engineering 69

New Guidelines for Graduate Software-Engineering Education 70

Notes to the Editor

Book Review: Reviewer's Response to Author's Reply 73

Final Thoughts

From the Chief Editor 76

INCOSE Professional Certification



Now is the time!



Get recognition for your systems engineering experience and knowledge through the INCOSE certification program! Apply today.

Visit www.incose.org and click on the CSEP icon to begin.

INSIGHT

Publication of the International
Council on Systems Engineering

Chief Editor Bob Kenley
insight@incose.org +1 260 460 0054

Assistant Editor Andrew Cashner
andrew.cashner@incose.org

Theme Editor Rob Cloutier
robert.cloutier@incose.org

Advertising Editor Christine Kowalski
advertising@incose.org +1 858 541 1725

Layout and Design Chuck Eng
chuck.eng@comcast.net +1 206 364 8696

Member Services INCOSE Central Office
info@incose.org +1 858 541-1725

On the Web <http://www.incose.org>

Article Submission INSIGHT@incose.org

Publication Schedule. *INSIGHT* is published four times per year. Issue and article/advertisement submission deadlines are as follows: **April 2010** issue – 15 February; **July 2010** issue – 15 May; **October 2010** issue – 08 August; **December 2010** issue – 15 October. For further information on submissions and issue themes, visit the INCOSE Web site as listed above.

Advertising in *INSIGHT*. Please see <http://www.incose.org/Products/Pubs/periodicals/insight.aspx> – or e-mail advertising@incose.org.

Subscriptions to *INSIGHT* are available to INCOSE members as part of their membership. Complimentary copies are available on a limited basis. Back issues are available on the INCOSE Web site. To inquire about membership or to order a copy, contact Member Services.

©2009 Copyright Notice. Unless otherwise noted, the entire contents are copyrighted by INCOSE and may not be reproduced in whole or in part without written permission by INCOSE. Permission is given for use of up to three paragraphs as long as full credit is provided. The opinions expressed in *INSIGHT* are those of the authors and advertisers and do not necessarily reflect the positions of the editorial staff or the International Council on Systems Engineering.

Who are we? INCOSE is a 7000+ member organization of systems engineers and others interested in systems engineering. Its purpose is to foster the definition, understanding, and practice of world class systems engineering in industry, government, and academia. INCOSE is comprised of chapters located in cities worldwide and is sponsored by a corporate advisory board and led by elected officers, directors, and membership board.

2009 INCOSE Board of Directors

President: Pat Hale, M.I.T.
President-Elect: Samantha Brown, BAE Systems
Secretary: Bob Kenley, Kenley Consulting, LLC
Treasurer: Ricardo Valerdi, M.I.T.

Director for Leadership and Organizational Development: Bill Ewald, Macro International

Director for Communications: Cecilia Haskins, Norwegian University of Science and Technology

Director for International Growth: Tat Soon Yeo, Temasek Defence Systems Institute

Director for Commercial Outreach: Henk van der Linden, SRON

Director for Strategy: Ralf Hartmann, EADS Astrium GmbH

Corporate Advisory Board Chair: Art Pyster, Stevens Institute of Technology

Member Board Chair: Jonette Stecklein, NASA

Member Board Co-Chair: Richard Grzybowski, Corning

Technical Director: Regina Griego, Sandia National Laboratory

Managing Executive: Holly Witte, Universal Management Services, LLC

President's Corner

Inclusive Thinking for INCOSE's Future

Pat Hale, patrick.hale@incose.org

Two years ago at the 2008 International Workshop, I started my time as president with a message titled “Doing Serious Work Together While Having Fun!” As I write my last “President’s Corner,” I have been thinking about what makes INCOSE valuable to me and what my hopes are for INCOSE and you, its members, in the future. For the past twenty years, INCOSE has grown steadily in membership, influence, geographic presence, and domain scope. Our profession has similarly increased in importance and impact, until we stand on the threshold of a world so interconnected that only systems thinking and the application of the methods we know as good systems engineering can help us to forge durable solutions to world challenges.

A highlight of 2009 was the *CNN Money* Web site’s selection of the systems engineer as the number-one job in America, and, significantly, CNN asked INCOSE to recommend people who could represent the diverse jobs in today’s systems-engineering marketplace. Many of you probably saw the profile about our own Anne O’Neil of the New York City MTA, describing the challenges and satisfaction of her job as a chief systems engineer. It is a milestone to have CNN come to INCOSE and to highlight a job outside the defense and aerospace domains. Defense is an important application space for systems engineering, but our profession’s influence has spread across many more domains of practice in the past decade, and the best practices we share have been put to use in products



and systems of tremendous variety.

At the 2008 IW, I emphasized that our members are the most important asset we have, and I want to reaffirm that judgment. INCOSE’s greatest strength is the generosity, skill, and dedication of its members. Our products are a direct result of that dedication and professionalism, and even as the challenges and products grow in complexity, our individual members, often backed by our corporate members, are successfully forging ahead in multiple initiatives that will enable continued growth and success in applications of systems engineering. For those who are leaders of this organization, whether at the chapter, national, or international levels, we must always remember that we serve the members and the profession, and allow that sense of service to guide our decisions.

I see a bright future for systems engineering and the potential for an equally bright future for INCOSE. Perhaps the only troubling current that has rippled through the past twelve years as I have served on the board of directors is the tension between national and global concerns, particularly in the United States. INCOSE has a destiny that is global, and will draw on the immense strength and resources of a global existence, providing that we can establish a foundation that creates a level playing field for global collaboration. Yes, INCOSE began in the United States as NCOSE, and many of our senior members grew from those roots, just as the systems-engineering profession, while originally conceived in the telecommunications industry, grew to its greatest strength in defense-and-aerospace-systems development and operations. While we should remember those roots and be proud of them, the time has come to acknowledge that we will achieve our greatest destiny only as a truly global organization, pursuing our vision and mission as equal international collaborators, and working to enable such collaboration on a worldwide scale. Accordingly, at the final Board of Directors meeting I presided over as president, the board unanimously passed a resolution to “to conduct the due diligence necessary to (re)establish a U.S. national organization.” While it may not seem intuitive that a U.S. national organization is a requirement for INCOSE to be truly global, I hope to convince you all that it is, for the following primary reasons:

- INCOSE is currently established as a non-profit corporation

IT'S POSSIBLE TO HAVE A SAFE AND SECURE HOME WITH ST ENGINEERING'S COMPREHENSIVE SECURITY SOLUTIONS AND CAPABILITIES



No challenge too daunting. That's the bold approach that has propelled the ST Engineering Group to where we are today – a dominant player in the world's engineering sectors, with over 100 subsidiaries in 42 cities across the world.

As a Group that represents global defence and engineering business, we spearhead the delivery of multi-dimensional total quality solutions in the Aerospace, Marine, Land Systems and Electronics sectors.

We have realised many dreams and created countless possibilities. So keep dreaming as we continue to make your dreams a reality.

Engineering our future.

 **ST Engineering**


aerospace • electronics • land systems • marine
www.stengg.com

in the state of California, which sometimes leads us to behave in a fashion that belies our international mission and goals, simply to comply with California corporate law. All of the other nations in INCOSE have formed national chapters under independent charters in order to comply with their legal and cultural frameworks and to serve both legitimate national considerations and globally shared concerns.

- Non-North American chapters can and do invest their assets and efforts into products that serve national interests, as well as contributing to international products. It is not surprising that other nations have national priorities in addition to INCOSE's global work; rather, it is surprising that the only nation that does not have a national organization and voice is the United States, INCOSE's birthplace. Because our board is a global mix of representation (absolutely appropriate for the international governing body), we refrain from lobbying U.S. government agencies or establishing positions that are specific to the U.S. While this is a requirement for a global leadership, it deprives U.S. members of a potentially powerful voice in U.S. policies pertaining to systems engineering and associated matters.
- Only by establishing a globally level organizational framework can we enable global collaboration, and realize INCOSE's full future potential, without national tensions at the international governance level.

As part of the motion to perform due diligence for "INCOSE-US," three subcommittees of the board were established to examine different aspects of the plan:

1. Legal, administrative, or bylaw changes to enable a split between U.S. and global INCOSE organizations.
2. Financial models, equitable asset distribution and division of services: What should the global organization provide, and what is better done on a national level?
3. Governance: How should the new global organization be governed to provide fair representation and organizational stability at minimum overhead?

These subcommittees have been working hard to develop an initial report and recommendations for further action, to be delivered at the 2010 International Workshop. I encourage you to make your voice heard during 2010 and to lend support to crafting the best organizational framework to carry your organization forward for the next twenty years. I thank you for the privilege of serving as your president for the past two years. It has been a lot of work, but it has also been, in great measure, a labor of love. 

**INCOSE PAST PRESIDENTS**

Paul Robitaille, 2006/07
 Heinz Stoewer, 2004/05
 John Snoderly, 2002/03
 John Clouet, 2001
 Donna H. Rhodes, 2000
 Ken Ptack, 1999
 William W. Schoening, 1998
 Eric C. Honour, 1997
 V. A. (Ginny) Lentz, 1996
 James Brill, 1995
 George Friedman, 1994
 Brian Mar, 1993
 Jerome Lake, 1992

Promote INCOSE

To obtain materials to promote INCOSE in the workplace and at events such as regional conferences, and symposia, contact the INCOSE Central Office:

info@incose.com

+1 858 541-1725, or access the INCOSE Web site at

www.incose.org.

7670 Opportunity Road
 Suite 220
 San Diego, CA 92111-2222

We supply INCOSE table signs, promotional items, and informational materials.

CORPORATE ADVISORY BOARD — MEMBER COMPANIES

Air Force Center for Systems Engineering

Alliant Techsystems

Analytic Services-Applied Systems Thinking Institute

BAE SYSTEMS

Boeing Commercial Airplane Co.

Boeing Integrated Defense Systems

Boeing Integrated Defense Systems – East

Booz Allen Hamilton Inc.

C.S. Draper Laboratory, Inc.

Carnegie Mellon University Software Engineering Institute

Certification Training International

Defense Acquisition University

EADS Astrium

EADS Military Air Systems

EADS N.V.

Federal Aviation Administration (U.S.)

General Dynamics

Honeywell International

IBM Corporation

ITT

Japan Manned Space Systems Corporation

JAXA (Japan Aerospace Exploration Agency)

Jet Propulsion Laboratory

Johns Hopkins University

L-3 Communications Integrated Systems

Lockheed Martin Corporation

ManTech International Corporation

MAP systeme

Missouri University of Science and Technology

Mitsubishi Electric Corporation

National Aeronautics and Space Administration

National Geospatial – Intelligence Agency

National Reconnaissance Office

National Security Agency

Naval Surface Warfare Center – Dahlgren Division

Northrop Grumman Corporation

Northrop Grumman Information Technology – TASC

Office of the Under Secretary of Defense (AT&L), Systems and Software Engineering

Project Performance International

Raytheon Corporation

Rockwell Collins, Inc.

Rolls Royce

Saab AB

SAIC

Sandia National Laboratories

Serco Defence, Science and Technology

Siemens – UGS PLM Software

SPAWAR Systems Center Charleston

SRA International

Stevens Institute of Technology

Swedish Defence Materiel Administration

Systems Engineering Innovation Centre

Tectura Corporation

Thales

The Aerospace Corporation

The MITRE Corporation

UK MoD Integration Authority

United Technologies Corporation

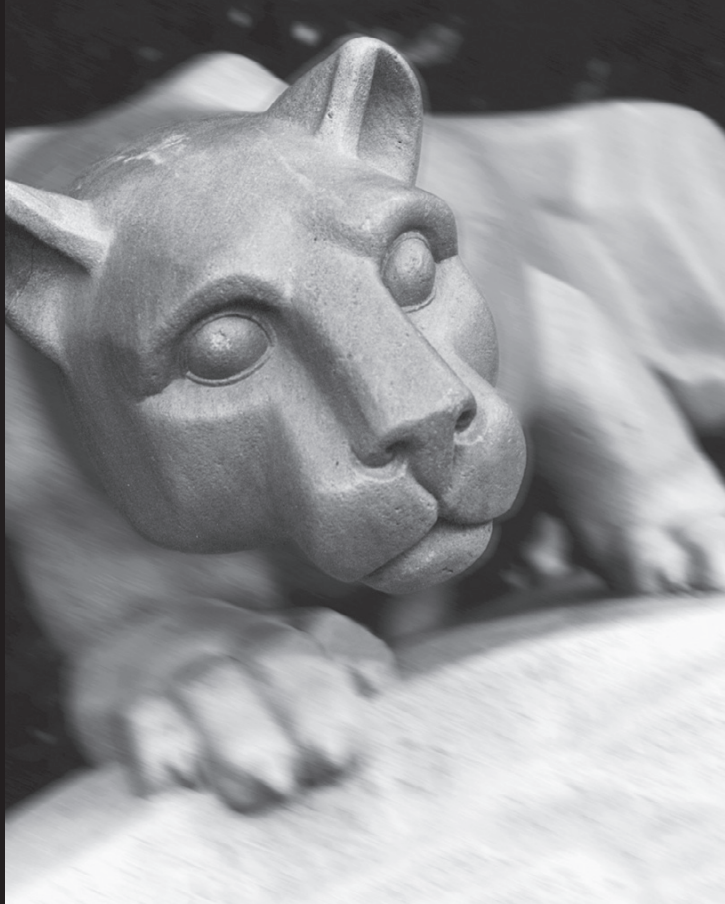
University of Southern California

US Army ARDEC

US Army CERDEC

Vitech Corporation

PENN STATE | ONLINE



Online Master's Degree in Systems Engineering

Advance Your Career

- Gain a quality education in a convenient online format
- Apply your skills to any engineering discipline
- Build a professional network with classmates
- Become a leader in your organization
- Finish in as little as two years



Apply now

www.worldcampus.psu.edu/INCOSE

Penn State is committed to affirmative action, equal opportunity, and the diversity of its workforce. U.Ed.OUT 10-0369/10-WC-148bkh/bjm



Introduction to this Special Edition on Model-based Systems Engineering

Robert Cloutier, robert.cloutier@incose.org

Model-based systems engineering (MBSE) has been with us for many years now. In the fall of 1998, a special issue of *INSIGHT* (vol. 1, no. 3) proclaimed it as “a new paradigm,” and included these articles:

1. “The INCOSE Model Driven System Design Interest Group,” by Howard Lykins and Bob Cohen
2. “Information Models as a Prerequisite to Software Tool Interoperability,” by Byron Purves and Loyd Baker
3. “Aspects of Modeling,” by Ingmar Ogren
4. “The Benefits of Model Based Engineering,” by David W. Oliver
5. “DD21 Smart Product Model,” by Jerry Golub

A little over a decade later, it seems appropriate to take a closer look at what we have learned, and where we may be heading with regard to MBSE. One sign of the rising interest in MBSE is that during this year’s International Symposium in Singapore, there were two MBSE tracks, and participants of the MBSE Initiative also met the day before the conference began. Further evidence can be found in this issue of *INSIGHT*, where there are thirteen articles on model-based systems engineering, ranging from the current and future trends, research in MBSE, and most important, examples and lessons learned — reflecting a consensus that MBSE is “ready for prime time” (under certain conditions).

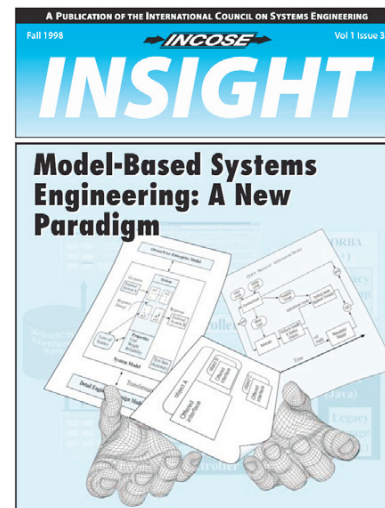
It is clear reading these articles that there is no single view of what constitutes model-based systems engineering. Some may say it is synonymous with using the Systems Modeling Language (SysML). Further, there are the methodology wars — functional decomposition versus object-oriented decomposition. Others will say MBSE consists of behavioral analysis or some other tool of the month. For my part, I have been suggesting to my students that the

specific tool, or language, or approach, is not the important thing; rather, systems engineers should model to understand the problem, and to communicate with others about the problem. If your modeling approach helps you accomplish that, it is a good thing.

There is no common thread that runs through the articles, except that it is important to model. Therefore, you can read the articles in any order you wish. I will briefly introduce the articles here, and you can turn to those you are most interested in first—but I highly recommend that you ultimately read them all. Let me lead off with Sandy Friedenthal and his presentation of lessons learned from early adoption of SysML. He goes on to discuss the future directions for SysML, including the potential for a SysML

certification program. Ron Williamson follows that up with an article on the forthcoming Unified Modeling Language (UML) profile for defense projects, focusing specifically on complex systems and system of systems. This profile is called UPDM.¹ Next, if you are unsure about the alphabet soup of modeling tools available for MBSE, Jeff Estefan introduces us to a systems-engineering modeling tool survey that was conducted by the Jet Propulsion Laboratory, and is available through INCOSE. There exists a U.S. Department of Defense certification and accreditation process, and Curtis Barefield addresses the use of MBSE as a supplement to that process.

Did I mention research? Jose Garcia presents some of his research by looking at the use of holons and operational evaluation modeling for context-sensitive systems for MBSE, and Russell Peak and his team present a fascinating project that is underway with Lockheed Martin and John Deere to integrate system design with simulation and analysis using SysML. It is a



1. The Unified Modeling Language Profile for the United States Department of Defense Architecture Framework and the United Kingdom Ministry of Defense Architecture Framework.

Table 1. MBSE authors

	Author	Affiliation
1	Robert Cloutier	Stevens Institute of Technology
2	Sanford Friedenthal	Lockheed Martin
3	Ron Williamson	Raytheon
4	Jeff Estefan	Jet Propulsion Laboratory
5	Curtis Barefield	Booz Allen Hamilton
6	Jose S. Garcia, Jr	The Boeing Company
7	R. Karban	European Southern Observatory
	R. Hauber	HOOD group
	T. Weilkens	Oose GmbH
8	Jack Ring	Innovation Management
9	C. Delp	NASA/JPL
	L. Cooney	NASA/JPL
	C. Dutenhoffer	NASA/JPL
	R. Gostelow	NASA/JPL
	M. Jackson	NASA/JPL
	M. Wilkerson	NASA/JPL
	T. Kahn	NASA/Ames
	S. Piggott	Canadian Space Agency
10	Russell Peak	Georgia Institute of Technology
	Chris Paredis	Georgia Institute of Technology
	Leon McGinnis	Georgia Institute of Technology
	Sandford Friedenthal	Lockheed Martin
	Roger Burkhart	Deere & Company
11	Steven Jenkins	Jet Propulsion Laboratory
12	Harald Eisenmann	EADS Astrium GmbH
	Hans-Peter de Koning	European Space Agency
13	Anatoly Levenchuk	TechInvestLab.ru

real-life example of how SysML can be used as a central repository for a multitude of data from engineering tools. Jack Ring offers us a “Model-based Systems Praxis for Intelligent Enterprises.”

And now, let’s discuss the practitioners. Robert Karban, Rudolf Hauber, and Tim Weilkens demonstrate for us the real-world application of SysML to the design of the European Southern Observatory telescope. First it was used to reverse engineer the design artifacts, and then it was used to forward engineer part of the European Extremely Large Telescope (E-ELT). Chris Delp and his team

presents another “space-age” article for this issue, “The Challenge of Model-based Systems Engineering for Space Systems, Year 2,” and Harald Eisenmann discusses European space-system development. Next, Steve Jenkins addresses a modeling approach to document production.

Finally, we have an article from our newest INCOSE chapter from Russia. Anatoly Levenchuk reminds us that while SysML is gaining acceptance and momentum, we always need to be looking forward. He reminds us that SysML has its roots in the software community’s Unified Modeling Language (UML). Moreover, there are a number of new technologies coming out of that same software practice toward which systems engineers should not turn a blind eye.

And there you have it: MBSE articles from noted authors and members of INCOSE, researchers, practitioners, and visionaries. I would like to thank each and every one of the authors who contributed to making this issue a large success. It has been a pleasure working with each of you. Your enthusiasm was such that I never had a problem “running down the next submission.”

Table 1 lists the contributors’ names, and organizational affiliations. As you can see, the contributors come from a wide array of organizations in the United States and Europe that are serious about MBSE.

I hope you enjoy reading this special issue on model-based systems engineering. ①

Start off a banner year for the INCOSE Foundation with a gift to the future.



You make the difference – we provide the means through scholarships and grants.

Contact Holly Witte, Foundation Managing Director, for more information.

holly.witte@incose.org

*Have you remembered the Foundation in your will?
Many companies match gifts. Please ask your company to match your gift.
We accept all major credit cards.*

School of Systems and Enterprises

SOFTWARE ENGINEERING

Delivering graduate programs tailored to the real-world education needs
of today's Software and Systems Professionals

Master of Science in Software Engineering

Graduate Certificates in:

- Software Engineering
- Systems-Centric Software Engineering
- Software Engineering in Finance
- Financial Software Engineering

Courses delivered in convenient and flexible formats, on-site at industry and government locations, online via Stevens award-winning WebCampus and on-campus at Stevens in Hoboken, NJ and Washington, DC.

The Master of Science program in Software Engineering (SSW) emphasizes the skills needed to apply software technologies to the realization of software products on time, within budget and with known quality. Our courses teach you the latest and best software engineering skills and theory, to allow you to effectively architect, build, and maintain both large and small scale systems.

The program consists of the following six required core courses, and four faculty advisor directed electives:

- SSW 540 Fundamentals of Software Engineering
- SSW 533 Software Cost Estimation and Metrics
- SSW 564 Software Requirements Analysis and Engineering
- SSW 565 Software Architecture and Component-Based Design
- SSW 567 Software Testing, Quality Assurance and Maintenance
- SSW 689 Software Systems Reliability Theory and Practice

Electives: Students are required to complete four advisor-approved electives, or any one of the Graduate Certificates listed above.

FOR ADDITIONAL INFORMATION CONTACT:

Linda Laird

*Program Director, Software Engineering
School of Systems and Enterprises
Stevens Institute of Technology
Hoboken, NJ and Washington, DC*

Email: Linda.laird@stevens.edu

STEVENS
Institute of Technology



<http://sse.stevens.edu/Software>

SysML: Lessons from Early Applications and Future Directions

Sanford Friedenthal, sanford.friedenthal@incose.org

The Object Management Group's Systems Modeling Language (OMG SysML™) is "a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and parametric equations that can integrate with other engineering analysis models. SysML represents a subset of UML 2¹ with extensions" to satisfy the needs for system modeling.² Further information on SysML can be found at <http://www.omgsysml.org>.

The OMG SysML specification was adopted in 2006 and version 1.0 became an available specification in September 2007. Since then, the SysML specification has continued to evolve, with version 1.1 published by the OMG in November 2008 and version 1.2 submitted to the OMG in September 2009. Several tool vendors have implemented SysML in their tools, several writers have written books and articles, and academic departments have begun to offer courses that include SysML. Early adopters across the industrial world have begun to use SysML in a broad range of aerospace and commercial applications. The INCOSE Model-Based Systems Engineering Initiative has yielded results from applying SysML in some of the MBSE Challenge teams (see "The Challenge of Model-based Systems Engineering for Space Systems, Year 2" in this issue). This experience and infrastructure are providing a foundation for more widespread adoption of the language as part of a model-based systems-engineering approach. This article highlights some of the observations and lessons from early applications of SysML and looks at future directions for the language to further enable the practice of MBSE.

Lessons Learned

The following are some lessons learned from early applications

1. Unified Modeling Language, version 2.0.
2. Object Management Group, "What is OMG SysML?" *OMG Systems Modeling Language: The Official OMG SysML Site*, http://www.omgsysml.org/#What-Is_SysML (accessed 26 Oct. 2009).

of SysML in support of MBSE.

1. MBSE is a cultural change. A model-based approach to systems engineering involves a fundamental shift from traditional documentation-based approaches. In MBSE, the model becomes a primary artifact to represent the system specification and design. The system model is managed and controlled, and some of the documentation becomes a by-product that is populated by the modeling information. This change from a traditional document-centric view of systems engineering can require a different way of thinking about how the systems-engineering effort is planned, executed, and controlled. An organization or project team should not make the transition to MBSE in an ad-hoc manner, but should employ concepts of organizational change in support of continuous improvement. These concepts include clearly identifying the issues to be addressed by MBSE, engaging stakeholders, developing and executing a plan for improvement or transition, and monitoring the results.

2. A well-defined MBSE method is essential. MBSE formalizes the practice of systems engineering through the use of models. As such, MBSE requires a high level of rigor to leverage its benefits. In developing a system model, one can quickly get overwhelmed with the amount of information that is generated about the system. An MBSE method must be clearly defined to support the model development. The method should also provide guidance on how to organize the system model to ensure it can be navigated, managed, and controlled. Several different MBSE methods are summarized in an article in this issue by Jeff Estefan ("Survey of Model-Based-Systems-Engineering Methodologies"), which provides an excellent starting point for identifying candidate MBSE methods.

3. New practitioners need training in the language, methods, and tools of MBSE. MBSE with SysML requires a set of skills that take time to learn. The learning curve can take several months to reach a moderate level of proficiency in the application of SysML in support of MBSE. It should be noted that there are distinct concepts and proficiencies required to learn SysML, the MBSE method, and the modeling tools. As a result, it has been found use-

ful to provide separate the training in the language, method, and tools to develop proficiency in each area. The training should also be adapted to different members of the project team. In particular, a small core modeling team may require more significant MBSE training, while the larger project team may only require sufficient training to understand the modeling artifacts. After the initial training, ongoing mentorship is essential to provide the support needed to help the team climb the learning curve.

4. Pilot projects can be used to validate the MBSE approach. Before deploying MBSE to a significant project, it is recommended that pilot projects be implemented to validate the MBSE approach, and its applicability to the project. The pilot shows how MBSE can most effectively be applied to a targeted set of programs. The pilot can also serve to build the skill base for use on the program, and the pilot results, including the modeling artifacts, can serve as a starting point or template for the targeted programs. A pilot project should be well planned with clear objectives, deliverables, milestones, and sufficient resources to achieve the objectives. In addition, team continuity with effective leadership and stakeholder participation are essential elements for a successful pilot project.

5. Well-defined modeling objectives and scope are critical to MBSE success. An old axiom is that modeling is intended to address specific concerns or answer specific questions. This is particularly critical in applying MBSE. The application of MBSE to a particular project should have a well-defined purpose, objectives, and scope, and the scope should be consistent with the planned resources and schedule. There are many aspects of MBSE that can provide value to a project, such as improvements in the specification quality, integrity of system design, productivity through the evolution of the system design, reduced risk, and other potential benefits. However, different levels of model breadth, depth, and precision are required to support different purposes. Scoping the model to meet its objectives within program constraints is essential to managing stakeholder expectations including those from program management, the customer, and other members of the development team.

Future Directions

SysML is in its early stages of adoption. Some of the areas that are currently being pursued are highlighted below.

1. Language evolution. The language continues to evolve in response to end-user and tool-vendor feedback. The OMG SysML Revision Task Force for SysML

version 1.3 was chartered in September 2009 and is cochaired by Roger Burkhart and Rick Steiner. The scope of revisions through the Revision Task Force is limited by OMG policy. Major revisions are handled through a new request for proposal.

A SysML Request for Information was issued at the June 2009 OMG meeting. The survey is used to elicit feedback on issues and recommendations relative to the use of SysML in support of MBSE. The survey results will be analyzed by Rob Cloutier and made available to the OMG and INCOSE following the period of data collection and analysis, which will conclude early 2010. This feedback will provide a key input to identify future enhancements to incorporate into a SysML roadmap.

2. SysML Certification. A OMG Certified Systems Modeling Professional (OCSMP) certification program has been jointly initiated by the Object Management Group and INCOSE. The certification objectives are to certify systems engineers

and other practitioners on SysML with the purpose of (a) helping systems-engineering professionals to assess and demonstrate their knowledge and skills in SysML and its application to MBSE, (b) helping organizations grow their capability in this critical skill area, and (c) promoting the use of SysML in support

of MBSE. The certification program will consist of four competency levels aimed at model reviewers who need to interpret the diagrams, and model developers who need to create the models. The certification program is expected to be in place in 2011.

3. Integration with Simulation and Analysis. There has been significant effort to establish approaches to integrate the system model in SysML with various simulation and analysis models. Some of this work is addressed later in an article by Russell Peak. This is considered a critical area to more fully leverage an MBSE approach across a diverse set of modeling and simulation domains. One such example is the integration of SysML with Modelica models. Modelica is a sophisticated and standardized simulation modeling language that is maintained by the Open Modelica Association. A working group has been established as part of the OMG's Systems Engineering Domain Special Interest Group to formalize the mapping between SysML and Modelica; it is chaired by Chris Paredis. This effort is expected to result in a customized version of SysML that can be automatically transformed to a Modelica model and executed by a Modelica modeling tool.


4. MBSE Tool Interoperability. The system model must integrate across a range of modeling domains, including hardware, software, analysis, and test models and tools. Model and data interchange standards are essential to achieve the model and tool interoperability. Evolution of model and data interchange standards continues to be a focus of the OMG's standards activities. The OMG Model

.....
*A model-based approach to systems engineering
 involves a fundamental shift from traditional
 documentation-based approaches.*

Interchange Working Group is coordinating vendor efforts to demonstrate and enhance their ability to exchange modeling information via the XML Metadata Interchange (XMI) standard. This working group established a set of test cases to incrementally verify the exchange capability of UML, SysML, UPDM, and other profiles. A second effort, led by David Price and Alice Feeney, is focused on the integration between SysML and ISO Application Protocol 233 (AP233). AP233 is a STEP-based³ data-exchange standard targeted to support the needs of the systems-engineering community by integrating systems-engineering data with other types of engineering analysis that are typically associated with hardware design. A validation tool developed by Peter Denno from the U.S. National Institute of Standards and Technology is being used to support interoperability testing.

5. SysML Integration with UPDM and Other Profiles. SysML is being integrated with other UML profiles. In particular, SysML can be leveraged when applying the UPDM, as described in Ron Williamson's article in this issue. The UPDM can be used for architecting at the system-of-systems level, and then integrated with SysML for systems modeling. Other profiles that are being integrated with SysML include MARTE (Modeling and Analysis of Real-Time Embedded Systems).

Summary

Model-based systems engineering is part of INCOSE's *Systems Engineering Vision 2020*, and will be fundamental to the future practice of systems engineering. SysML was jointly developed by INCOSE and the Object Management Group to provide an enabling capability for MBSE. There has been considerable early application of SysML across a range of industry and application domains, and much has been learned about the challenges of transitioning to a model-based approach. The need for a systematic approach to transition from a document-based approach to a model-based approach is essential, and includes the need for well-defined MBSE methods, training, and piloting of the approach. The MBSE initiative is helping to address some of these challenges by providing a body of knowledge that can be shared across industry and academia. In addition, several initiatives are underway to enhance the SysML language, certify systems modelers, enhance the integration with simulation, improve tool interoperability, and integrate with other domain-specific modeling languages such as UPDM and MARTE. 

3. STEP: Standard for the Exchange of Product Model Data, ISO 10303

Model-based Systems Engineering for Systems of Systems

Ron C. Williamson, ronald.williamson@incose.org

INCOSE's model-based systems-engineering effort is focused on improving model-based methods for systems engineering. One of the MBSE activities, System of Systems, addresses the modeling and systems-engineering capabilities necessary to develop enterprise-wide solutions in a more cost-effective, timely, and high-quality manner than would be possible with traditional systems engineering. The Defense Acquisition University's *Defense Acquisition Guidebook* defines a system of systems as "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities" (2004 version, chapter 4, quoted in ODUSD 2008: v). Developing the vocabulary, methods, and tools in support of enterprise architectures is a critical element of both the MBSE system-of-systems top-down needs-development strategy and the bottom-up strategy, which leverages the best practices and tooling in the industry. The development of model-based standards that clearly define the metamodels associated with enterprise architectures and associated models is fundamental to the success of model-based development.

The partnership and cross membership between INCOSE and the Object Management Group (OMG) has produced a productive synergy of ideas and methods; this collaboration continues with the ongoing efforts to develop a unified profile for military architecture frameworks. An industry standards team, composed of INCOSE and OMG members, has been established to build on previous efforts within the OMG to develop a modeling standard that supports both the U.S. Department of Defense Architecture Framework (DoDAF) and the U.K. Ministry of Defence Architecture Framework (MODAF). The modeling standard is called UPDM, the Unified Profile for DoDAF and MODAF.

UPDM defines an industry standard representation for enterprise architectures that are compliant with DoDAF 1.5 or MODAF 1.2. UPDM leverages the existing SysML standard for requirements, parametrics, allocation, and other critical features that enable UPDM to integrate with SysML models for system-level specification, design, and analysis. UPDM is an Object Management Group (OMG) initiative. UPDM is expected to lead to significant improvements in the consistency, quality, and tool interoperability of enterprise architectures that comply with these frameworks. In addition, it is expected to be fully compatible with both UML models for specification and design models for Level 0 compliance and SysML specification and design models for Level 1 compliance. Developing a specification that fully supports DoDAF/MODAF is essential for organizations developing NEC

(network-enabled capability) systems. The UPDM Team will also make use of NATO's recently adopted architectural framework standard, NAF version 3. NAF is based on MODAF 1.1 but has been extended to support service-oriented architecture (SOA). SOA views have since been included in MODAF 1.2.

Although the UPDM team group is independent of the OMG, it submitted a new specification to the OMG using the OMG fast-track Request for Comments adoption process. The final UPDM specification was anticipated to become an available specification by September 2009. UPDM will then be updated through the OMG technology adoption process to address the requirements of DoDAF 2.0. The team is also considering the Security and Information Protection views of the Canadian DNDAF.

The UPDM team has already defined working groups to focus on specific aspects of the ongoing specification updates and plans to set up a forum to enable interested parties to keep up to date with the progress on the specification. The membership of the UPDM team comprises development tool vendors and defense-industry contractors along with representatives of the key government agency stakeholders—the U.S. Department of Defense and the U.K. Ministry of Defence.

In addition to the UPDM effort, the INCOSE MBSE SoS activity intends to leverage and influence the ongoing standards efforts to develop model-based approaches to systems engineering at the level of the enterprise or the system of systems. Based on industry feedback, the SoS activity will focus on key perspectives relevant to the MBSE SoS activity including (but not limited to)

- Executable models,
- Business structure and behavioral models,
- Service-oriented models,
- Security models, and
- Information models.

Case Study: Gap Analysis using UPDM

To further illustrate the current mainstream approach to modeling systems of systems, we will summarize a case study of a subset of the key perspectives listed above. First, let's review a brief summary of the key concepts included in UPDM as a system-of-systems modeling framework.

- Viewpoints and views
 - ◊ Critical organizing constructs to enable the effective management of potentially complex models
 - ◊ Extensible to accommodate new viewpoints and views
- Strategic capability, operational/business, services, systems, standards viewpoints

- ◊ Predefined set of viewpoints and views to accommodate DoDAF 1.5 and MODAF 1.2 specifications “out of the box”
- ◊ User model visualization integrated via standard UML, SysML, and SoaML visual modeling standards
- Context, interfaces, constraints, and parametrics
 - ◊ Supports core systems-engineering principles directly applicable to system-of-systems-level analysis and engineering
 - Context via block or composite structure model elements and diagrams
 - Interfaces using standard and flow ports on structural elements with strongly typed interface specifications
 - Constraints applicable to structural, linkage, dynamics, and parametric model elements
 - Parametrics capturing the boundary conditions and key performance parameters allocated to structural and behavioral models elements
 - Technology standards
 - ◊ Includes current and emerging standards that may be allocated to any model element
 - ◊ Leverages allocation mechanisms built into SysML

Next, the key set of requirements that drove the UPDM standards specification were captured as a domain metamodel structure (see metamodel structure in figure 1 below) in a set of model element packages. This metamodel defined the key terms, relationships,

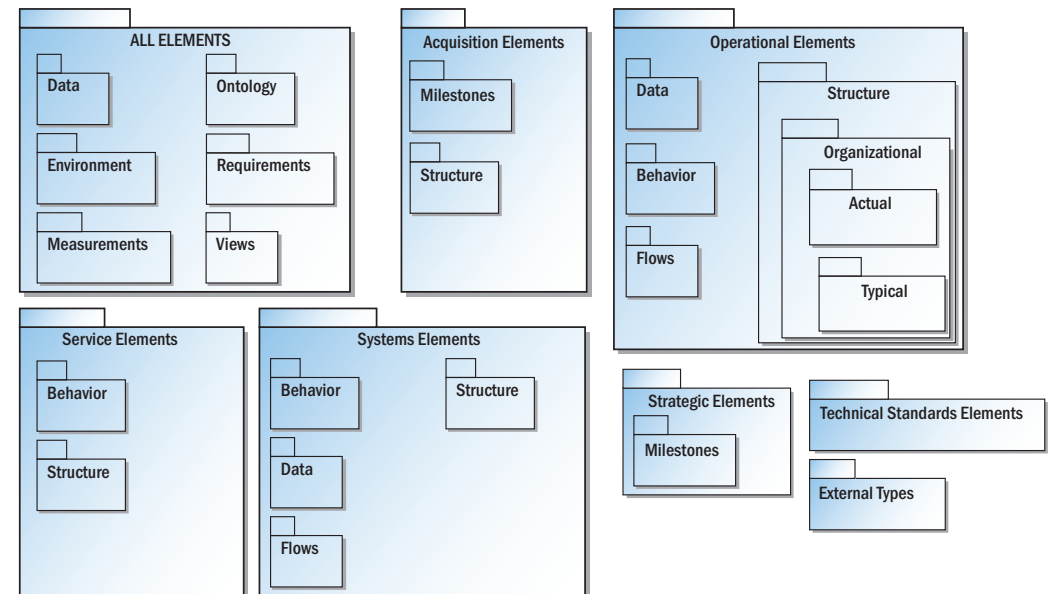


Figure 1. UPDM system-of-systems metamodel structure

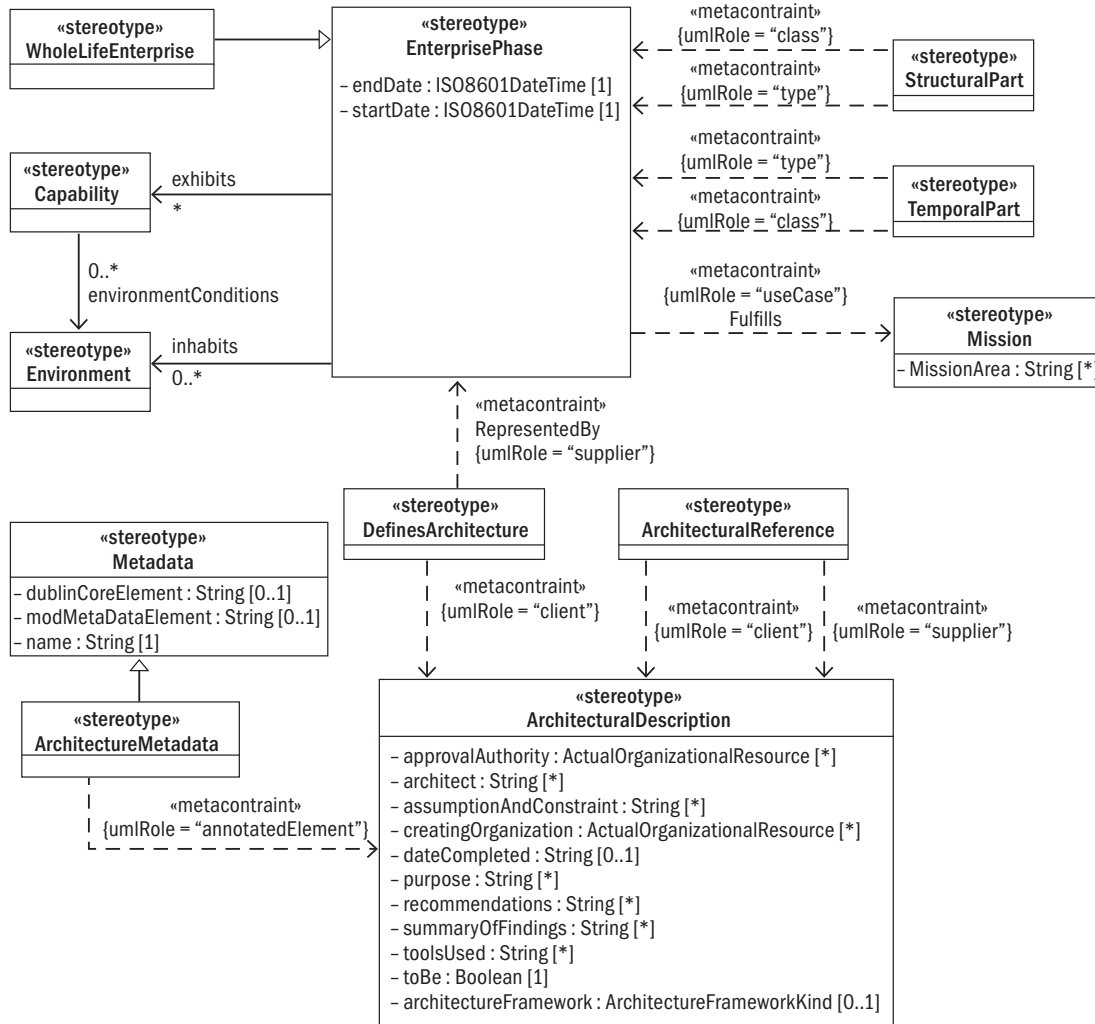


Figure 2. UPDM profile example

and constraints that defined the underlying semantics of the architecture frameworks (i.e., DoDAF 1.5 and MODAF 1.2). These requirements were then translated into a profile specification (see profile example in figure 2).

Note in figure 1 that the emphasis in each package (where applicable) on key modeling constructs such as behavior, structure, data, flows, milestones, requirements, views, environment, measurements and ontology. Extensibility is accommodated through the use of external type definitions.

As depicted in figure 2, the structural and temporal aspects of an enterprise are critical to

modeling the evolution of system of systems over time within the context of a well-defined environment and a well-defined mission (either business or military oriented). The logical concept of a capability is fundamental to enterprise-level analysis, keeping the level of discourse away from any physical realization of the solution space.

Modeling Tools Support

During the several years of evolution of the UPDM standards effort within the Object Management Group process, all major tool vendors evaluated and/or participated in the effort. All key stakeholders participated in the process in addition to the tool vendors and included representatives major aerospace companies (as end users of the tool and MBSE for SoS subject-matter experts), the United States Department of Defense, British Ministry of Defence, the Canadian DND, and NATO customer representatives. Key vendors have committed to implementing the UPDM standard in their tools and include the following:

- Artisan Software Tools, UPDM profile
- EmbeddedPlus, UPDM plug-in
- IBM, Rational Rhapsody, UPDM profile
- NoMagic, MagicDraw, UPDM profile
- Sparx Systems, enterprise architect UPDM profile

References

- National Defense Industrial Association. 2009. DoDAF 2.0 meta model (DM2) walk-through. Slides from presentation given at U.S. Department of Defense Enterprise Architecture Conference (Washington, DC). <http://www.ndia.org/DoDEntArchitecture/Documents/DoDAF%20Metamodel%20Walkthru%202009-06-01r2.pdf>.
- Object Management Group. 2009. *Unified Profile for the Department of Defense Architecture Framework (DoDAF) and the Ministry of Defence Architecture Framework (MODAF): FTF beta 2*. Available at <http://www.omg.org/cgi-bin/doc?dtc/09-05-08>.
- ODUSD (Office of the [U.S.] Deputy Under Secretary of Defense for Acquisition and Technology, Systems and Software Engineering). 2008. *Systems engineering guide for systems of systems*. Version 1.0. Washington, DC. Available at <http://www.acq.osd.mil/sse/docs/SE-Guide-for-SoS.pdf>.
- U.K. Ministry of Defense. 2008. *The MOD architecture framework version 1.2*. <http://www.modaf.org.uk>.
- U.S. Department of Defense. 2007. *DoD architecture framework version 1.5*. 2 vols. Washington, DC. Available at http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_I.pdf and http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_II.pdf.



Collaborative Systems Engineering Solution:

FOSTERING INNOVATION AND MASTERING FUTURE SUSTAINABLE PRODUCT DEVELOPMENT

At a time when:

- More than 40% projects fail due to lack of requirements and traceability capability.
- More than 50% of projects fail, due to poor systems architecture validation.
- 80% of costs are committed in the first 20% of product lifecycle.

How can we manage Product & Process development complexity and comply to ever growing stringent industry regulations ?

The “Collaborative Systems Engineering” Solution is the answer:

- Ensuring right-to-market delivery through complete traceability from customer needs to final product validation.
- Mastering a collaborative model based systems definition unifying Requirement, Functional, Logical & Physical views.
- Assuring the right engineering decisions are made by simulating product behavior & environment, powered by a multi-engineering modeling language.

Visit www.3ds.com/hightech for more information



MBSE Methodology Survey

Jeff Estefan, jeff.estefan@incose.org

One of the activities of INCOSE's MBSE Initiative deals with Processes, Practices, and Methods. This activity is comprised of two components. The first of these is the subject of this article: under my leadership, this component aims to provide the international systems-engineering community with a survey, to be updated annually or on an as-needed basis, of some of the leading MBSE methodologies used in industry.¹ The other component, under the leadership of Ray Jorgensen and Joe Bedocs, deals with configuration-management practices with the objective of identifying and codifying approaches to manage and control the model baseline and address integrating models with multiple users. In the future, it is hoped that the scope of the overall activity be expanded to publish industrial best practices and MBSE principles.

Survey Background and Status

The MBSE methodology survey was originally initiated as an internal study at NASA's Jet Propulsion Laboratory (JPL) in Pasadena, California, as part of an initiative to advance the state of the practice in model-based engineering. JPL then identified the work product as a roadmap for the 2007 fiscal year, along with industry benchmarking in MBSE. The objective of the study was to go beyond a simple survey. The engineers at JPL felt that additional supporting material was important in helping to ground the concepts of MBSE. This included differentiating the meaning of processes, methods, and tools, characterizing the role of lifecycle models—specifically, those for project, acquisition, and systems engineering—and identifying and characterizing the role of models in support of MBSE processes. In addition, they felt that there needed to be some discussion of the role of the Unified Modeling Language (UML) and SysML, which are visual modeling-language

1. An MBSE methodology can be characterized as the collection of related processes, methods, and tools used to support the discipline of systems engineering in a model-based or model-driven context. We adopt the definition of INCOSE Fellow James N. Martin that differentiates process from method and tool (Martin 1997). Martin defines *process* as a logical sequence of tasks performed to achieve a particular objective. Processes essentially define what needs to be done without specifying the *how*. A *method* consists of techniques for performing a task, in other words, it defines the *how* of each task described by the process. A *tool* is an instrument that, when applied to a particular method, can enhance the efficiency of the task—provided it is applied properly and by somebody with adequate skills and training in the tool.

standards from the Object Management Group.

While the MBSE methodology survey report was published initially as an internal JPL engineering memorandum, it was scrubbed, cleared for external release, and published on 25 May 2007 as Revision A (Estefan 2007). I then contributed this document to the INCOSE MBSE Initiative as part of its Processes, Practices, and Methods activity. It was also made available for public download on the OMG SysML official Web site, available at <http://www.omgsysml.org>.

The publication was intended to be a “living” document, to be updated annually or on an as needed basis. This first study surveyed five methodologies, each of which will be briefly summarized below. An important caveat is that this was a survey report only. It was not intended to be a formal assessment of these candidate MBSE methodologies, because such an assessment is tightly associated with specific project and organizational needs, and thus, any assessment would be highly subjective. It is up to the community of users to assess which candidate MBSE methodology best suits their needs.

Analysis performed for the 2007 survey and reported during the MBSE Workshop held in conjunction with the 2008 INCOSE International Workshop in Albuquerque suggested that there was a robust set of MBSE methodologies ready for adoption by organizations and practitioners at the time of publication. Feedback from the INCOSE community suggested that a gap existed for considering additional methodologies, some of which were highly targeted at software or software-intensive systems. Given the large scope of model-based software-engineering methodologies, I determined that such a survey would be too broad and that the software community already had a robust set of candidate model-based methodologies to choose from. Consequently, I decided to focus exclusively on MBSE methodologies for the INCOSE MBSE Initiative. In addition, INCOSE Fellow Jack Ring suggested that consideration should be given to incorporating the pioneering MBSE work of INCOSE Fellow Wayne Wymore on the “Wymorian” notation and associated methods. Further analysis was needed to determine if additional candidate MBSE methodologies had been overlooked.

As a result of the gap analysis performed following publication of Revision A of the MBSE methodology survey report, an updated version was published on May 23, 2008 as Revision B (Estefan 2008a). As with the Revision A report, this publication was made

publicly available on the OMG SysML Web site. It was also submitted to INCOSE publication staff for consideration as an official INCOSE technical publication. This updated revision incorporated the suggestion from the 2007–2008 gap analysis to include an overview of Prof. Wymore’s pioneering MBSE work. The statement of scope was also updated to reflect the focus on systems-engineering processes rather than software-engineering processes. An additional methodology was added to result in a total of six methodologies surveyed. This new methodology documented in Revision B was Prof. Dov Dori’s object process methodology (OPM). Finally, a new section was added to describe the role of OMG model-driven architecture (MDA) and executable UML foundation. Of course, Revision B also included minor editorial updates from Revision A.

Results from this report were presented during the MBSE track at the 2008 INCOSE International Symposium in Utrecht, the Netherlands, in June 2008 (see Griego 2008). During the course of the remainder of the 2008 year through early 2009, work was focused on formatting the Revision B survey as an official INCOSE Technical Data publication (INCOSE-TD-2007-003-01) (Estefan 2008b), which is now available for public download on the INCOSE Technical Resource Center Web site at <http://www.incose.org/ProductsPubs/techresourcecenter.aspx> (under the “Modeling and Tools Technical Committee” heading).

During the 2009 INCOSE International Workshop in San Francisco in January 2009, a gap analysis determined that an additional MBSE methodology should be reviewed and incorporated into another revision of the survey. The addition added Weilkiens’ SYSMOD methodology, which is described in his textbook *Systems Engineering with SysML/UML* (Weilkiens 2007: 271–284) and available at <http://sysmod.system-modeling.com/>. Throughout this effort, managed under the auspices of the MBSE Initiative, I have worked to support the infusion of some of the methodologies studied in the survey into various efforts of MBSE challenge teams, particularly the space-systems challenge team led by Chris Delp (Delp et al. 2008, 2009).

Methodologies in Brief

The latest revision of the MBSE methodology survey at the time of submittal deadline for this *INSIGHT* article was Revision B, which surveyed six methodologies. It is possible that Revision C or some variation such as a Wiki-based update to the survey might be available by the time you are reading this article (see the section below on “Future Work”). Nevertheless, a very brief summary of the six methodologies published in the Revision B report is provided here. Space limitations prevent a more detailed description. The interested reader should refer to the INCOSE Technical Data publication INCOSE-TD-2007-003-01 for more details as well as references for additional information (Estefan 2008b). A link to this publication is provided in the reference list.

IBM Telelogic Harmony-SE. This methodology is a service-request-driven approach, described by SysML structure diagrams and state/mode changes (activities), which are described as operational contracts. This approach somewhat mirrors the “vee” model. Task flow and work products include top-level process elements of requirements analysis, system functional analysis, and architectural design. Detailed task flows and work products are provided for each process element and modeled as SysML activity diagrams.

INCOSE Object-Oriented Systems Engineering Methodology (OOSEM). This methodology integrates a top-down (functional decomposition) approach with a model-based approach. It leverages object-oriented concepts and uses SysML (formerly UML) to support the specification, analysis, design, and verification of systems. It is intended to ease integration with object-oriented software development, hardware development, and testing. OOSEM includes the following activities: analyze stakeholder needs, define system requirements, define logical architecture, synthesize candidate allocated architectures, optimize and evaluate alternatives, and validate and verify system.

IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDS). This methodology extends the RUP style of concurrent design and iterative development to support new roles, artifacts, and disciplines for systems engineering. This approach includes an emphasis on business modeling, business actors, and flow of events as well as systems-engineering model levels and model viewpoints. This approach introduces the concept of “locality,” meaning a member of a system partition representing a generalized or abstract view of physical resources, which are linked by connections. This methodology also provides a schema for allocated vs. derived requirements by tying use-case flowdown and flow of events in a “white-box” view of a system to locality/subsystem (allocated requirements) and collaboration (derived requirements) and subsystem-level flowdown activity. There is also support for designing additional components beyond RUP software focus (e.g., hardware systems)—this is known as “RUP+.”

Vitech MBSE Methodology. This methodology initially was created by INCOSE Fellow Jim Long and offered as a Vitech MBSE training series. Jim Long has also offered a streamlined version of the training at past INCOSE International Workshops and Symposia as well. The training module contains four primary concurrent systems-engineering activities, which are linked and maintained through a common system-design repository. Each activity is linked within the context of associated “domains,” namely, process (systems-engineering activities), source requirements, behavior, verification and validation, and architecture.

The methodology recommends a strong adherence to an agreed-upon “system-definition language”— that is, a systems-engineering schema or ontology to manage the syntax and semantics of model artifacts. This approach uses the incremental process known as the “onion model,” which allows complete interim solutions at increasing levels of detail during the system-specification process; this produces a lower-risk design approach by checking for completeness and discovering constraints early in the design process. Detailed testing methods are described in support of system verification-and-validation activity.

JPL State Analysis (SA). Developed at NASA’s Jet Propulsion Laboratory, this methodology leverages model- and state-based control architecture. This approach defines *state* as a representation of the momentary condition of an evolving system; it defines *models* as describing how the state evolves, and *state variables* as abstractions representing “knowledge” of the state (in other words, the known state of the system is a value of its state variables at the time of interest). Together, state and models supply what is needed to operate the system, predict its future state, control it toward its desired state, and assess its performance. This approach defines an iterative process for state discovery and modeling, and allows models to evolve as appropriate across the project’s lifecycle. The state-analysis requirements process helps bridge the gap between the requirements on software specified by systems engineers. SA information is compiled in a Structured Query Language (SQL)-compliant database referred to as the “state database.”

Dori Object Process Methodology (OPM). This methodology is a formal paradigm for systems development, lifecycle support, and evolution. It combines simple object process diagrams with object process language (constrained natural language) and the basic building blocks of an object (something that exists or has potential of existence physically or mentally), a process (pattern of transformation that object undergoes), and the state (situation object can be in). It is a reflective methodology that refers to system lifecycle as system evolution. In OPM, the “system developing” (SD1) process contains three main stages: (1) requirement specifying, (2) analyzing and developing, and (3) implementing. SD1 also includes a “using and maintaining” state. Each process element can be “zoomed” multiple times. Visual models (object process diagrams) and associated object process languages are represented and captured in the OPCAT tool (available at <http://www.opcat.com>).

SYSMOD. As stated earlier, a gap analysis resulting from the MBSE Workshop at IW09 indicated that the Weilkiens SYSMOD methodology should also be evaluated as a possible candidate MBSE methodology to be included in a future survey report.


Future Work

Initially, the MBSE methodology survey was intended to be a living document. However, it is difficult to maintain such a deliverable over the course of several years. Consequently, an alternative approach to disseminating this type of information to the community of systems-engineering practitioners is being investigated at the time of this article’s submission. One potential solution might be a Wiki-based system that would be made publicly available via an INCOSE Web resource. This would allow authors or key focal individuals of specific MBSE methodologies, as well as emerging MBSE methodologies, to post content to the Wiki reflecting a synopsis of their particular methodology as well as links to additional resources describing the methodology. Until such an alternative becomes a reality, the participants of the INCOSE MBSE team will continue to publish annual updates to the MBSE methodology survey.

Acknowledgement and Disclaimer

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

References

- Crisp, H. E., R. Wray, J. Carl, and S. Brown. 2007. *Systems engineering vision 2020*. INCOSE-TP-2004-004-02. Version 2.03.
- Delp, C., C-Y. Lee, O. de Weck, C. Bishop, E. Analzone, R. Gostelow, and C. Duttonhoffer. 2008. The challenge of model-based systems engineering for space systems. *INSIGHT* 11 (5): 14–18.
- Estefan, J. 2007. Survey of model-based systems engineering (MBSE) methodologies. Revision A. White paper, INCOSE MBSE Focus Group. Available at http://www.omgsysml.org/MBSE_Methodology_Survey_RevA.pdf.
- . 2008a. Survey of model-based systems engineering (MBSE) methodologies. Revision B. White paper, INCOSE MBSE Initiative. Available at http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf.
- . 2008b. Survey of model-based systems engineering (MBSE) methodologies. Revision B. INCOSE-TD-2007-003-01. Prepared by the Model-Based Systems Engineering Initiative of INCOSE. Seattle, WA. Available at http://www.incose.org/ProductsPubs/pdf/techdata/MTTC/MBSE_Methodology_Survey_2008-0610_RevB-JAE2.pdf.
- Friedenthal, S., R. Griego, and M. Sampson. 2008. Workshop in Albuquerque promotes model-based systems engineering. *INSIGHT* 11 (2): 45–47.
- Griego, R. 2008. Model-based systems engineering activities. *INSIGHT* 11 (4): 45–46.
- Martin, J. N. 1997. *Systems engineering guidebook: A process for developing systems and products*. Boca Raton, FL: CRC Press.
- Weilkiens, T. 2007. *Systems engineering with SysML/UML*. Burlington, MA: Morgan Kaufman. 

Using Model-based Systems Engineering to Supplement the Certification-and-Accreditation Process of the U.S. Department of Defense

Curtis Barefield, curtis.barefield@incose.org

MBSE is a methodology that can be used to supplement several ongoing processes and procedures in the federal and civil acquisition systems.

Model-based systems engineering can be used to improve or complement risk analysis and management in the certification-and-accreditation (C&A) process of the U.S. Department of Defense by using modeling to provide a clear picture of residual risk associated with a system or enterprise. MBSE is also useful to create a true living document to enforce configuration management and track design changes through the lifecycle of the system or enterprise. Both ideas are reasonable and achievable, and could serve as a method to incorporate MBSE more fully into the lifecycles of federal and private-sector systems and enterprises. The resulting models could logically be tied to the architecture views created using Department of Defense Architecture Framework (DoDAF) and Ministry of Defense Architecture Framework (MODAF), which provide the exact type of overview alluded to in the briefing slides for MBSE on the INCOSE Web site (<https://connect.incose.org/tb/MnT/mbseworkshop/SAWG%20Shared%20Documents/MBSE%20Overview.ppt>).

Of course, MBSE could be used to create models supporting any architecture framework (thereby providing a deeper insight to the system or enterprise), but federal law has mandated risk management and the identification of risk for federal systems. Anything that will improve risk assessments and lifecycle management could be introduced using existing legacy or new program of record systems in the C&A pipeline. Up-front costs would increase, but the savings could be realized during post-deployment operations.

Governance issues related to Federal Acquisition Regulation (FAR), Defense Federal Acquisition Regulation (DFAR), DoD 5000 (acquisition) and 8500 (IA) series that are not captured or addressed in the architecture framework views would be addressed in the MBSE modeling as constraints; as parametric, activity, or state diagrams; or as use cases with traceability directly to the associated mission assurance category (MAC) and Confidentiality IA controls. Test procedures and results would be visually traceable to models at the level of the system or component.

Due to enterprise and system visibility from the architecture to the component level, the improved risk-analysis process would


reduce post-deployment costs because potential problems would be identified earlier in the lifecycle. Some additional considerations include the following:

1. In the Department of Defense, the Defense Information Assurance Certification and Accreditation Process (DIACAP) has eliminated most of the original System Security Authorization Agreement (SSAA) sections and attachments, effectively eliminating the living document that was designed to manage and access systems throughout their lifecycle. DIACAP has effectively replaced the SSAA with a database record and a few exhibits.
2. The National Information Assurance Certification and Accreditation Process (NIACAP) SSAA, although designed to be a living document, normally becomes “shelfware” until it is time for the annual (periodic) review and re-accreditation at the three-year point, as mandated by the Federal Information Security Management Act (FISMA).
3. Neither DIACAP artifacts nor NIACAP documents provide a means that supports easy risk analysis by C&A reviewers supporting the approval authority.
4. In a number of cases, the reviewers, certification authorities, and approval authorities are not able to glean a detailed understanding of the systems or enterprises due to the limited time allowed for risk analysis.
5. Without a detailed understanding of the reviewed system, a realistic assessment of residual risk is not likely. A system- or component-level model with traceability to the system (or enterprise) architecture would be extremely helpful in visualizing potential risks associated with the system or enterprise.
6. Test cases and results can be captured within the model, which then could be stored electronically with the DIACAP scorecard and other electronic artifacts in the applicable database.
7. Using modeling allows for increased opportunities to apply “what-if” scenarios by using XMI to import the baseline into

SysML applications with a low cost per seat at any level within the command structure.

8. Existing investments in products like DOORS are protected since a number of the low-cost SysML applications can be configured to accept direct input from these existing applications.
9. Modeling effectively improves the ability of an integrated product team to collaborate on the overall effect of changes within their area of expertise on the whole program by providing a single corporate model of the entire system or enterprise.
10. Documentation can be generated from the models using SysML applications so that the paper artifacts can be incorporated into acquisition- or program-related documentation.
11. To produce a living lifecycle model, MBSE models need to be used to trace specific components, processes, test cases with verifications, and use cases to the original requirements visualized in the architectural views. Changes in the system requirements, governance, or component will be clearly visible at whatever level you view the enterprise or system, and the ability to assess changes to costs, schedule, and risks become easier to ascertain.
12. The successful use of MBSE for an actual DoD project in the DIACAP pipeline will help make the case for an increased use of systems-engineering models in support of certification and accreditation and other procedures associated with the federal acquisition process.

This is the true worth of model-based systems engineering when coupled with an applicable architectural framework. Cost associated with implementing MBSE would primarily be applied in the beginning of the enterprise or system lifecycle. Use of modeling in requirements analysis and assessment of alternatives would increase the accuracy of cost estimates by ensuring that all systems and components selected for the program baseline are actually valid. Models at the system and component levels that are traceable to architecture will improve the accuracy in the decision process all the way through the enterprise's lifecycle. Increased accuracy would result in significant cost savings during the enterprise or system's operational life.

MBSE is a methodology that can be used to supplement several ongoing processes and procedures in the federal and civil acquisition systems. The use of the MBSE in the certification and accreditation process offers a means to implement modeling with minimal impact on operational costs and the potential for a reasonable return on investment, based on increased throughput with the improved risk analysis process. For a more detailed discussion of the MBSE process associated with certification and accreditation, please contact me at the e-mail address above. 

ASSISTANT PROFESSOR, SYSTEMS ENGINEERING

Penn State Great Valley School of Graduate Professional Studies is seeking qualified applicants for a faculty appointment in the area of systems engineering. In addition to teaching and research, Penn State Great Valley faculty members perform program, campus, and university service assignments. Required qualifications include a Ph.D. in systems engineering or a closely related field, and a demonstrated record of research accomplishments, preferably in systems design, software systems, or system integration. Prior teaching experience (preferably at the graduate level) is desirable. Experience in the systems engineering field in government or industry is highly valued. The position will be available starting Fall semester 2010 and may be either tenure track or multi-year fixed term, depending on the qualifications, interests, and goals of the candidate.

Located 20 miles northwest of Philadelphia, the Penn State Great Valley School of Graduate Professional Studies (www.sgps.psu.edu) is a special mission campus of The Pennsylvania State University. The campus enjoys a central location in a corporate park among world-class corporate neighbors in a rapidly growing technological corridor. Our graduate students are working professionals.

Penn State offers a competitive salary commensurate with experience, in addition to a comprehensive benefit package. For confidential consideration, submit letter of application, current curriculum vitae, research/publication history, and the names and contact information of three references to: Dr. Phillip A. Laplante, CSDP, PE,

Systems Engineering Search Committee Chair, Penn State Great Valley School of Graduate Professional Studies, 30 E. Swedesford Road, Malvern, PA 19355-1443 by January 31, 2010. Applications can also be sent electronically to jos12@psu.edu. Penn State is committed to affirmative action, equal opportunity and the diversity of its workforce.

ASSISTANT PROFESSOR, ENGINEERING MANAGEMENT

Penn State Great Valley School of Graduate Professional Studies is seeking qualified applicants for a faculty appointment in the area of engineering management. In addition to teaching and research, Penn State Great Valley faculty members perform program, campus, and university service assignments. Required qualifications include a Ph.D. in engineering management or a closely related field, and a demonstrated record of research accomplishments. Prior teaching experience (preferably at the graduate level) is desirable. Experience in the systems engineering field in government or industry is highly valued. The position will be available starting Fall semester 2010 and may be either tenure-track or multi-year fixed term, depending on the qualifications, interests, and goals of the candidate.

Located 20 miles northwest of Philadelphia, the Penn State Great Valley School of Graduate Professional Studies (www.sgps.psu.edu) is a special mission campus of The Pennsylvania State University. The campus enjoys a central location in a corporate park among world-class corporate neighbors in a rapidly growing technological corridor. Our graduate students are working professionals. Penn State offers a competitive salary commensurate with experience, in addition to a comprehensive benefit package. For confidential consideration, submit letter of application, current curriculum vitae, research/publication history, and the names and contact information of three references to:

Dr. Phillip A. Laplante, CSDP, PE, Engineering Management Search Committee Chair, Penn State Great Valley School of Graduate Professional Studies, 30 E. Swedesford Road, Malvern, PA 19355-1443 by January 31, 2010. Applications can also be sent electronically to jos12@psu.edu. Penn State is committed to affirmative action, equal opportunity and the diversity of its workforce.

Executable and Integrative Whole-System Modeling via the Application of OpEMCSS and Holons for Model-based Systems Engineering

Jose S. Garcia, Jr., jose.garcia@incose.org

The executable and integrative approach to model-based systems engineering ... allows both an expansionist and reductionist approach.

The advancements of science and technology in our society have provided us with two realities: (1) a utopian potential of improving our lives and economies, and (2) a growing complexity in integrating technology into our lives, often at a high cost. The field of engineering pervades our lives in almost every aspect of our society and economy. We are continuously dazzled by today's "futuristic" weapon systems, biomedical wonders, and intelligent, almost humanlike, information systems. However, from an engineering standpoint, designing and integrating such systems poses a growing challenge to make them work "as advertised." As evidenced on an almost daily basis in the news, large-scale and complex defense and space projects suffer from a continuous rash of cost overruns, schedule delays, and technological bottlenecks. This is because the problems engineers face in designing and integrating modern technical wonders are growing in complexity. Recognized as early as the 1940s by Bell Labs (and during the 1950s and 60s by the U.S. Department of Defense and NASA, respectively), technological developers have turned to the interdisciplinary field of systems engineering to manage and integrate complex engineering projects.

Systems engineering is an interdisciplinary, process-oriented approach to solving a wide variety of complex technical and societal problems. It is a "lever and fulcrum" that enables highly-complex system problems and projects to be easily and efficiently architected, broken-down, designed, developed, diagrammed, calculated, cataloged, evaluated, integrated, optimized, measured, and managed. Systems engineering as a formal field and methodology developed out of the disciplines of system science and operations research during the twentieth century. This type of "systems thinking" arose during the beginning of the century, initially among biologists, and was subsequently adopted by scientists and engineers to describe the interconnected nature of systems. As the pace of science and technology continues to march forward by leaps and bounds, the field of systems engineering continues to encounter an ever more complex and varied set of challenging,

yet ambiguous, set of systems problems. The problems and projects encountered by systems engineers are more interdisciplinary in nature than formerly, and they behave as complex adaptive systems. A complex adaptive system consists of two or more agents that adjust their behavior to achieve an overall system goal. Agents can be as simple as a desktop computer or Web site, or as complex as a human operator, a factory robot, space satellite, astronaut, war fighter, software application, or an autonomous deep-space probe. In addition, systems are not merely composed of physical entities, technologies, and software, but are comprised of people, processes, and enterprises that are highly networked, intelligent, and interconnected. In a complex system problem, these entities, technologies, people, processes, and enterprises can be conceptualized as interacting agents.

The challenge therefore, is to make these complex systems work and operate effectively and safely. Moreover, as the level of complexity rises, what used to be "systems" problems are now becoming "system of systems" problems. Tackling these more complex problems requires a cost-effective, thorough, and disciplined approach to systems engineering. However, as complexity increases, this "engineering of a system" needs to make use of intelligent and intuitive model-based systems-engineering techniques. Scientists and engineers face an ever-greater array of complex requirements. Complexity must be managed. A paradigm shift is occurring within systems engineering, where the level of complexity of system-of-systems projects can only accurately and precisely be modeled by a modeling construct that is executable (such as a software tool or algorithm). An engineering approach to a "whole-system" understanding of a design or system needs to be taken into consideration from the onset. For example, it is not merely enough to design and deploy a satellite. Everything from its conception, launch, communication, safety, and disposal must be considered. Thus there is an identifiable need for an executable and integrative approach to systems engineering.

A system of systems is a system abstracted and constructed of

an aggregate of individual systems, which interoperate and interface with each other, and are characterized by emergent properties (see figure 1). The need to engineer systems of systems has spawned the field of system-of-systems engineering, which can be defined as the engineering of large-scale systems.

What these statements say is that the aforementioned paradigm shift to the system-of-systems understanding of system problems, and their engineering via system-of-systems engineering, is an evolutionary value-adding process that presents a variety of challenges that include technology, human dynamics, and sustainability. In addition, a whole-system, or holistic, approach to engineering will facilitate complexity management. Engineering projects grow ever more complex with the increase of unique and challenging requirements. Using the construct of *holons* in the systems engineering process enables this holistic approach.

Holons are system agents that are autonomous and cooperate, and lend themselves well to understanding the interdependent and emergent properties of agent-based systems and their unique requirements. In this article I will argue that an executable and integrative approach to whole-system modeling can be accomplished by applying a methodology known as OpEMCSS (operational evaluation modeling for context-sensitive systems) to all aspects of model-driven design. These aspects include both model-based systems engineering and simulation-based systems engineering. Dr. John R. Clymer, my thesis advisor and professor of electrical engineering and systems engineering at Cal State Fullerton, developed OpEMCSS. Dr. Clymer is an INCOSE Fellow, and has been a member of INCOSE since its inception. Dr. Clymer developed the OpEM graphical language based on parallel processing language concepts and mathematical linguistics.

Operational evaluation modeling for context-sensitive systems is a modeling-and-simulation tool that allows an explicit understanding of complex system interactions and complex-adaptive-system behavior among system components and subsystems. The systems-engineering professional community, as represented

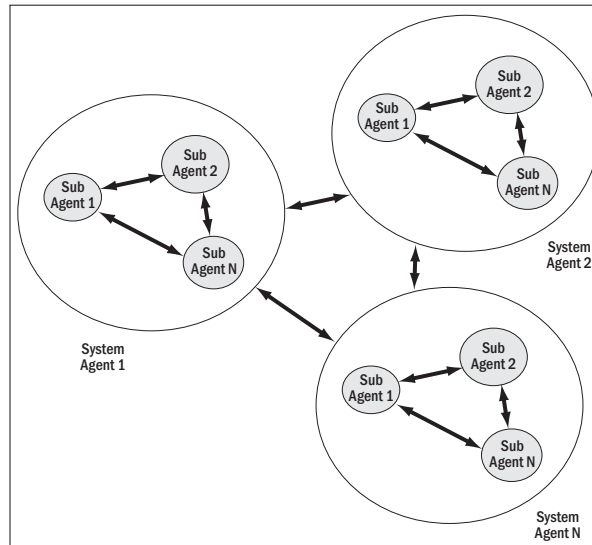
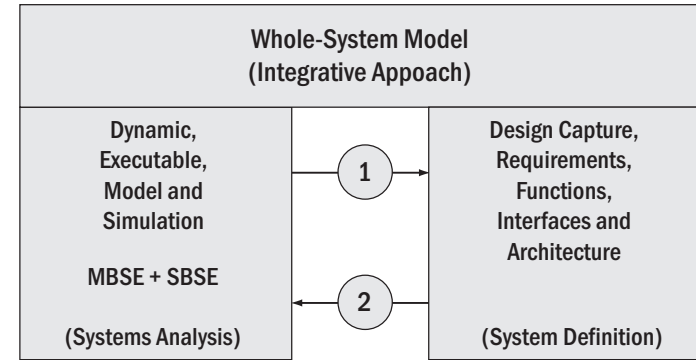


Figure 1. System-of-systems interactions



- 1 This interface provides MOE* and MOP* statistics to System Definition (results).
- 2 This interface provides values, margin, variables, KPP* to dynamic model.

* : MOE – Measure of Effectiveness/MOP – Measure of Performance/KPP – Key Performance Parameters

Figure 2. Integrative modeling paradigm

by INCOSE, has embraced model-based systems engineering. MBSE techniques have been identified as a discipline within the field of systems engineering. There are active working groups and conferences where the state of the art of complex modeling is advanced and where an identifiable need for an executable and integrative approach to model-based systems engineering for whole-system modeling has emerged.

Description of Research

This research was intended to identify and provide an integrative and executable approach to whole-system modeling via the application of OpEMCSS to both simulation-based and model-based systems engineering. The intent of this research was to provide a whole-system understanding of the systems-engineering process. This study, although focused on systems engineering, encompasses multidisciplinary applications. The research and conclusions provided herein will help set the foundation for an integrative modeling paradigm (see figure 2).

OpEMCSS has an extensive library of blocks that make possible the modeling of context-sensitive systems that exhibit emergent behavior. OpEMCSS thus allows the rapid modeling and analysis of complex processes. Emergent behavior arises in multiagent systems. One modeling block used in OpEMCSS is the “Wait until Event” block. This block models resource contention (see figures 3

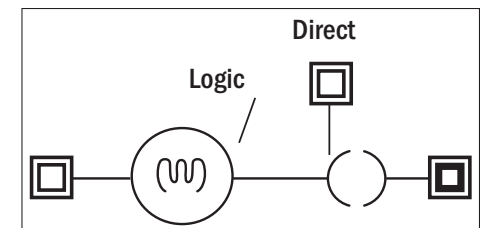


Figure 3. “Wait until Event” block in OpEMCSS library

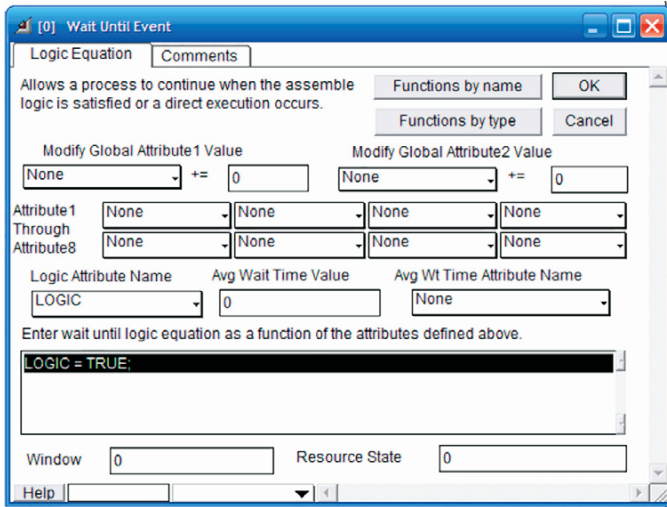


Figure 4. "Wait until Event" block dialog box

into a process and a body of knowledge for executing complex projects involving technology and the physical sciences. At the advent of the twenty-first century and at the apex of the information age, systems engineering has grown into a highly sophisticated, network-centric, complex engineering discipline. This requires a whole-system approach to systems thinking. Systems thinking requires taking an expansionist approach to system design and analysis, in addition to reductionism. The executable and integrative approach to model-based systems engineering for whole-system modeling via operational evaluation modeling for context-sensitive systems allows both an expansionist and reductionist approach.

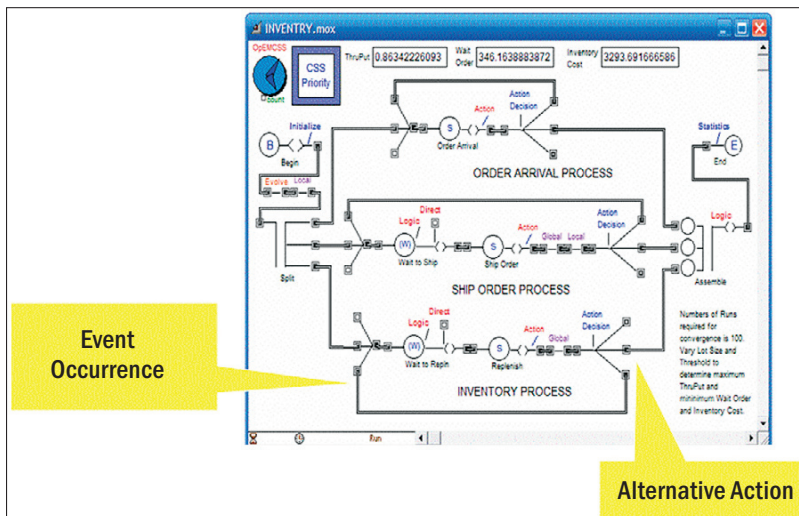


Figure 5. OpEMCSS example model

and 4) as exhibited in systems with multiple agents. Agents can be human operators, satellites, computers, or something else.


Significance of This Research

As the military-industrial complex evolved during the Cold War, the field of systems engineering organized itself

The significance of this research is the wide array of application areas for such whole-system modeling techniques. OpEMCSS provides the ability to model just about any type of system. For example, the following model in OpEMCSS (figure 5) is that of a supply chain.

Modeling the supply chain and simulating the interactions between the various agents allows complexity to be comprehended, managed, and optimized. A supply-chain and value-chain management process is a multiagent orchestration of manufacturers, suppliers, transportation companies, and communication networks, all working in concert to maintain a flow of products. Figure 6 shows a mathematical optimization fitness surface. This data set, as produced by the MBSE techniques modeled in OpEMCSS, allows a system, such as a supply chain or manufacturing operation, to be optimized.

A model is an abstraction or representation of reality. Model-based systems engineering is the practice and discipline within the field of systems engineering that models system interactions and interoperability in order to better engineer or develop an intended system design. Simulation is a computer-based or mathematical-based analysis of a complex system, which measures a system concept. Simulation-based systems engineering is the process and discipline of using simulation to evaluate candidate system designs within an operational scenario.

The ultimate goal of both model-based and simulation-based systems engineering is emulation. In terms of systems engineering, emulation is the real-time operation of a system's behavior (a simulation of the actual system functions and interfaces). Emulation is the ultimate objective of simulation, since simulation is an approximation, and emulation is an actual implementation. Emulation allows you to observe, measure, and anticipate system behavior. For this reason, OpEMCSS should be adopted into the systems-engineering process as the whole-system modeling construct that provides executable and integrative modeling for a holonic systems-engineering process. 

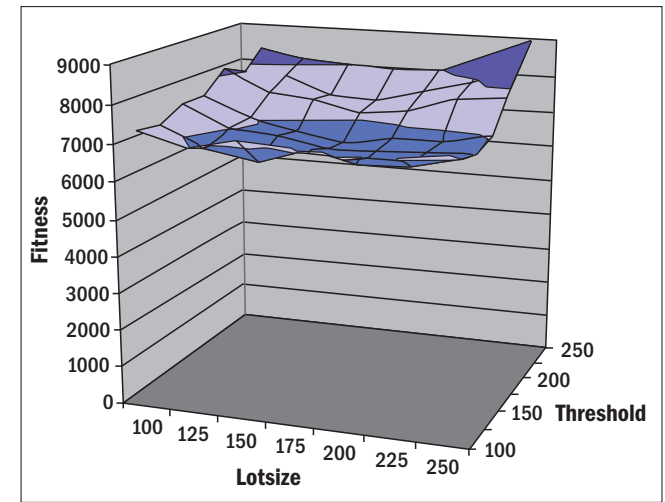


Figure 6. Optimization of complexity

MBSE in Telescope Modeling

Robert Karban, rkarban@incose.org; Rudolf Hauber, rudolf.hauber@hood-group.com; and Tim Weilkiens, tim.weilkiens@incose.org

In the framework of INCOSE's strategic initiative, the *Systems Engineering Vision 2020*, one of the main areas of focus is model-based systems engineering. In keeping with this emphasis, the European Southern Observatory (ESO; <http://www.eso.org/>) is collaborating with the German Chapter of INCOSE (<http://www.gfse.de/>) in the form of an "MBSE Challenge" team. The team's task is to demonstrate solutions to challenging problems using MBSE. The Active Phasing Experiment (APE; see Gonte et al. 2004), a European Union Framework Program 6 project, was chosen as the subject of the SE² Challenge Team (<http://mbse.gfse.de/>). Many technical products in the telescope domain show an increasing integration of mechanics with electronics, information processing, and also optics, and can therefore be rightly considered as optomechatronic systems.

This article presents the results of model-based systems engineering using the Systems Modeling Language (SysML; see Ogren 2000), drawing on experiences within the MBSE Challenge project and also the European Extremely Large Telescope (E-ELT) project. For the former project, SysML models were created by

reverse engineering from existing documentation and from interviews with systems engineers, whereas for the latter project, the practices were applied to a new system. We will make use of Ingmar Ogren's concept of a common project model (Ogren 2000) to establish a common understanding of the system.

Project Description

Our system case study is the Active Phasing Experiment technology demonstrator for the future

European Extremely Large Telescope, which is a high-tech, interdisciplinary optomechatronic system in operation at the Paranal

observatory (see ESO 2009). Telescopes of the next generation need to collect significantly more light than older models, therefore requiring bigger reflecting surfaces that consist of many individual mirror segments. Due to different disturbances (such as vibrations, wind, and gravity), the segments must be actively controlled to get a continuous mirror surface with a phasing error of only a few nanometers over the main mirror's diameter of 42 m. The main challenge is to correctly detect the positioning errors of the segments via specific phasing sensors in order to create a continuous mirror surface.

APE was developed to evaluate those sensors, and was installed on one of the 8 m telescopes that constitutes part of the Very Large Telescope in Chile (VLT) for sky tests. APE can be seen as the black box in figure 1. For the installation it had to comply with various mechanical, electrical, optical, and software interfaces. APE consists of about two hundred sensors and actuators such as wheels, translation stages, lenses, detectors, mirrors, light sources, an interferometer, and twelve computing nodes for control. Since APE had to be deployed in the test lab and in an already existing telescope, for each context it was necessary to model variants of function, interfaces, and structure. All of these characteristics made APE well suited to evaluate the potential of SysML in tackling similar issues.

MBSE Challenge Goals

SysML is only a graphical language and defines a set of diagrammatics, modeling elements, a formal syntax, and semantics. Like any language (formal or informal), it can be used in many different ways, including many wrong ways. Most notably, it is possible by misusing the language to create incorrect models. The main goals of the SE² MBSE Challenge Team are to

- create modeling guidelines and conventions for all system aspects, hierarchy levels, and views;
- provide examples in SysML, solving common modeling problems;
- build a comprehensive model, which serves as the basis for providing different views to different engineering aspects and subsequent activities; and to
- demonstrate that SysML is an effective means to support systems engineering.

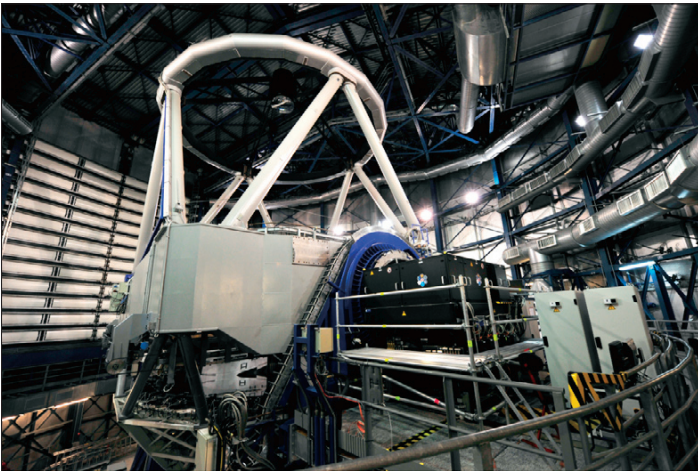


Figure 1. Active Phasing Experiment at the Very Large Telescope

The SE² team has provided their guidelines for modeling on the “frequently asked questions” page of their Web site (<http://mbse.gfse.de/documents/faq.html>). A SysML model, as described in the next section, illustrates the results of their comprehensive modeling. The SysML model is not merely a mental abstraction, but a collection of complex data structures that can be edited, augmented, queried, and reported on by means of a suitable tool, which is an indispensable pillar for MBSE.

Model Structure and Overview

Modeling is all about abstraction (reducing the complexity of a system), improving communication and understanding of the system, and providing reusable system elements. However, capturing a lot of different aspects like requirements, structure, interfaces, behavior, and verification in a model of a complex system like APE leads to a large model. Therefore, the first challenge is to find a clear, intuitive structure for the model, since a well-structured model is crucial for controlling complexity.

In the APE model we applied two techniques to establish a good and easily understandable structure: recursive structure patterns and views. In SysML, packages are the structuring mechanism to group model elements, which were used by the authors for both techniques. Packages are a mechanism to group model elements in higher-level units, similar to the way folders organize files in a computer’s file system.

APE Model Structure Pattern

The APE model uses a recursive structure pattern to model different system aspects on every level of decomposition of the APE. On each level we have packages for these aspects:

- Objectives and requirements
- Context
- System structure
- Behavior
- Data
- Performance
- Verification
- Auxiliary packages: comments, problems, issues

Overview Diagrams

To improve the understandability of the APE model, we provide content diagrams, which describe the system by showing all the different aspects captured by the model. The top-level overview diagram is a “project content” diagram (figure 2) and serves as an entry point and an anchor for navigation through the complex model.

The System Structure View (“APE_Structure” in figure 2) is used to decompose the system and provide the recursive modeling pattern within the subsystem package; for example, the “APE_Structure” sub-package “InternalMetrology” contains the same view packages, the “InternalMetrology” sub-package “PhaseModulator” (figure 3) contains again the same view packages, and so on. For every system decomposition element (like the nested structure view for the “InternalMetrology” in figure 3), a package exists together with an overview diagram that shows the aspect packages of the respective element. The arrows between the packages show their dependencies. This recursive model structure provides an intuitive look-and-feel navigation capability within the APE model.

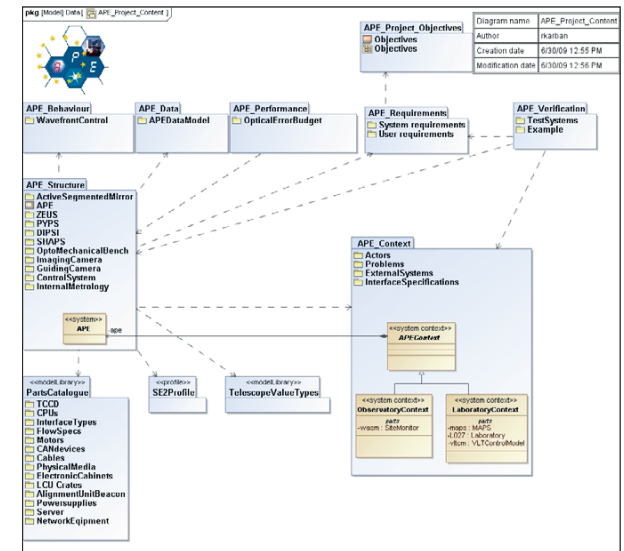


Figure 2. APE project-content diagram

Aspects

Packages are used to separate specific aspects of the system at each decomposition level. Each package provides one or more aspects on APE for a specific perspective, such as context, structure, data, and so on. Each of these aspects should be documented so the model makes clear which attributes or characteristics are observed within the aspect.

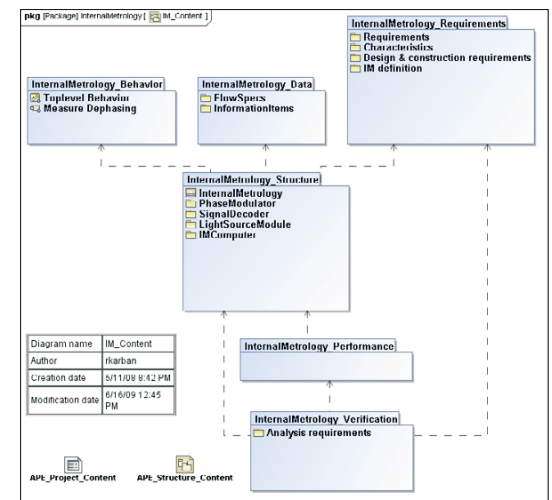


Figure 3. Subsystem content diagram

Objectives and Requirements

The next sections describe the APE modeling approach for each of the view-points mentioned above. APE, like any complex system, has a large number of functional, performance, physical, and interface requirements that have to be

satisfied. This implies the need for formal requirements management during the project. APE has about 50 high-level system requirements. The control system has also about 50

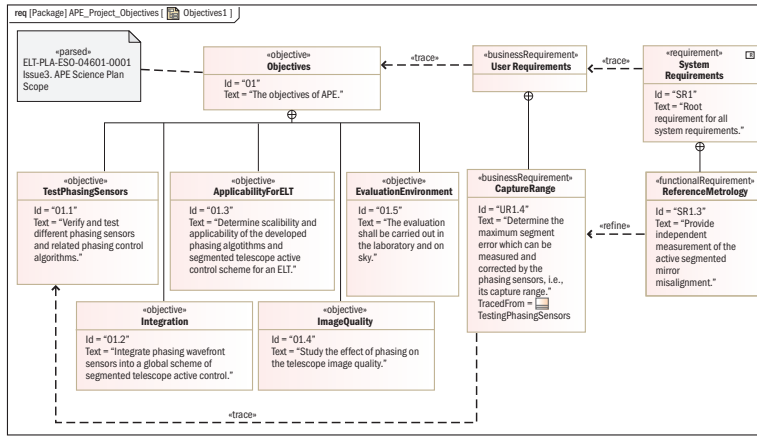


Figure 4. APE objectives diagram

requirements, refined by 150 use cases. We used the SysML requirements diagrams to show the main objectives of APE (figure 4). The limitations of standard text-based requirements-management tools were overcome by visualizing key requirements and their impact on system design and verification in an intuitive way.

The following user defined types extend the SysML requirements modeling features to organize and trace objectives, business and system requirements of APE. Figure 5 shows the dependencies of the requirements and automatically created traceability matrices.

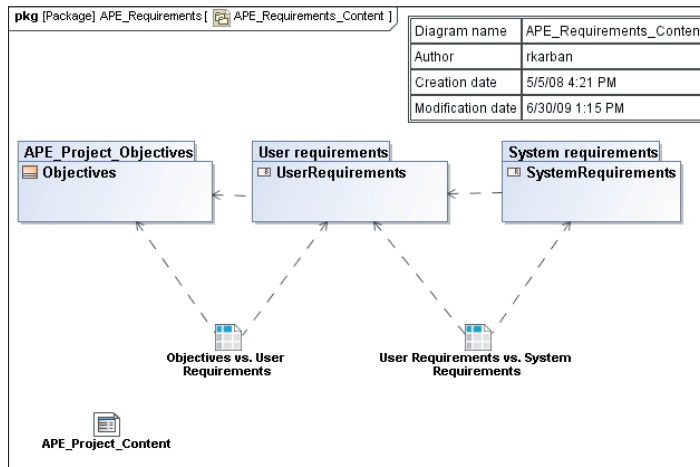


Figure 5. Traceability among different types of requirements

In the figure, *objectives* are project-specific stereotypes of a class to capture the objectives for the projects (see figure 4); *business requirements* are project-specific stereotypes of a SysML requirement to capture the high-level requirements for the projects; and for the *system requirements*, SysML requirements are used to capture the detailed requirements for the projects.

Context

The system’s context defines the system’s boundaries and is modeled using SysML internal block diagrams. A SysML internal block diagram (IBD) shows a

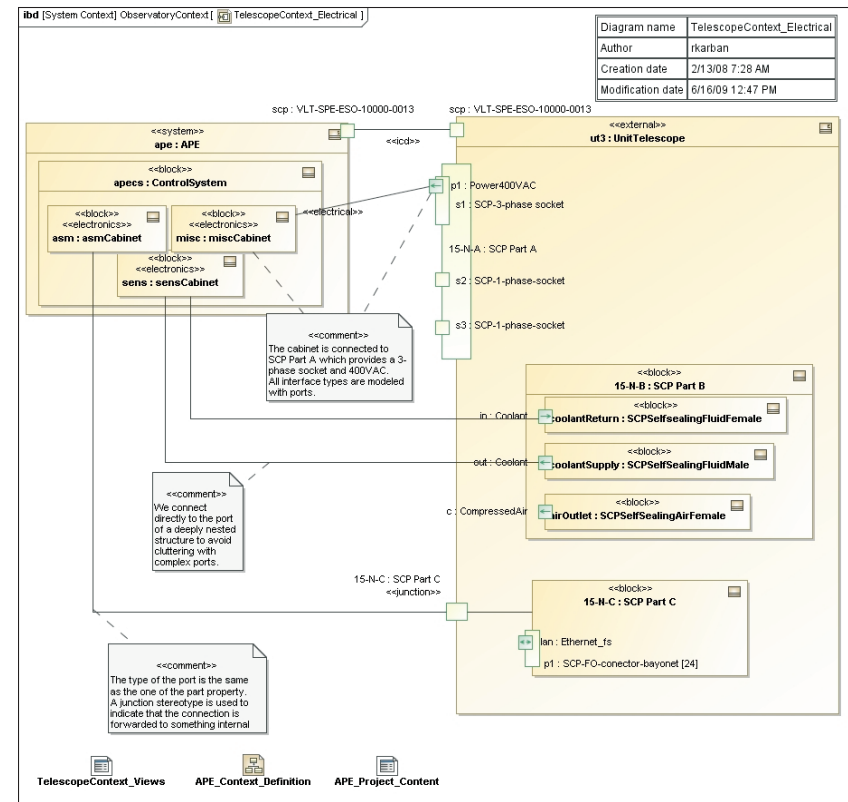


Figure 6. Internal block diagram of electrical context

block, its parts, and its interfaces. For the system context our main focus is on system interfaces.

SysML uses ports to model the interfaces of a block. SysML provides a standard port for service-like interfaces and flow ports for physical interfaces. However, there are different possibilities to use these ports for capturing system interfaces (see figure 6):

- Standard ports to model abstract interfaces as representation of an interface control document (ICD) as shown for port “scp”
- Flow ports for a combination of mechanical and flow interface at block level (model physical and logical properties at the border of a block, hiding its internals), as shown for port “15-N-A”
- Model mechanical and flow interface directly at the specific part level, as shown for “coolantReturn” and “coolantSupply,” crossing the border of the outer part
- Model mechanical and flow interface at block and part level, as shown for port “15-N-C”

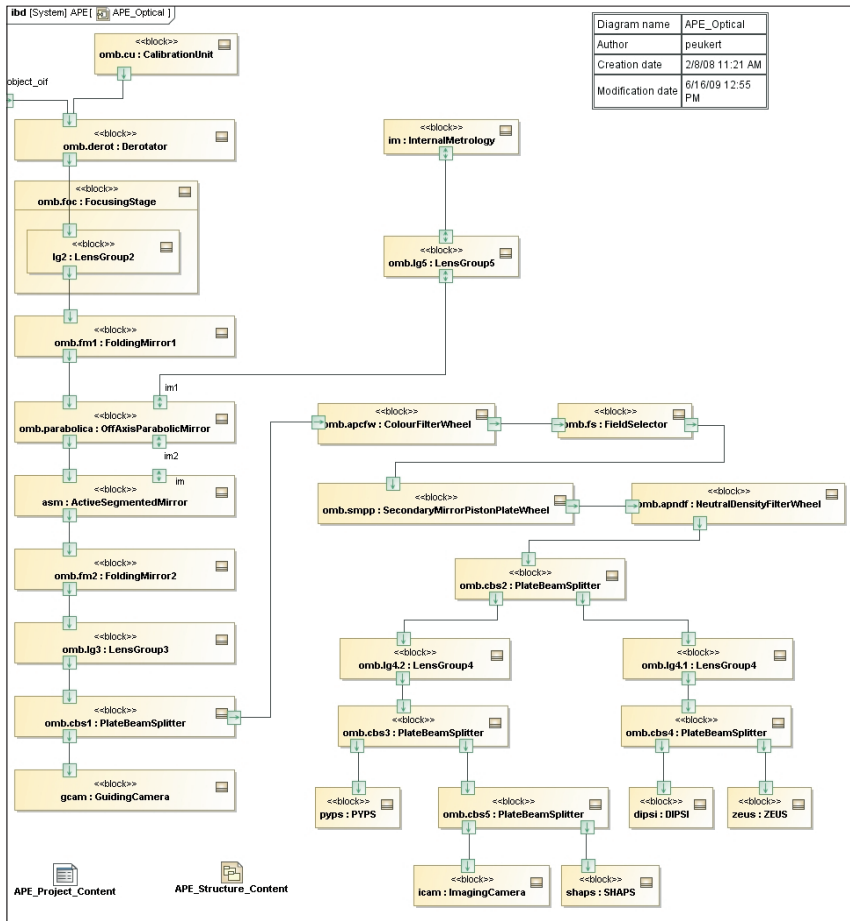


Figure 7. IBD of the APE optical layout

To ensure consistency between the interface control document and the model, the latter serves as the basis for the former. Figure 6 shows the context of the telescope from an electrical viewpoint (ports on control system side are not modeled).

System Structure

The system structure may be the most self-evident aspect to model. We have used SysML block definition diagrams (BDDs) to model the product tree and internal block diagrams to model the structure and interfaces of APE and its subsystem. The APE hierarchical breakdown is based on the product tree. It has several levels, going from the highest level into more and more details, using decomposition of its elements.

On the other hand, a complex system has much more than just one internal structure. There are multiple views showing electrical, optical, and mechanical elements that are interconnected, and therefore multiple structures exist. The same

components can be connected in different views in different ways. We have used IBDs to show the electrical, optical, and mechanical layout of APE and its components at different levels. Figure 7 shows the optical layout in an abstract manner. The connectors are stereotyped as *optical*, but they are elided for readability.

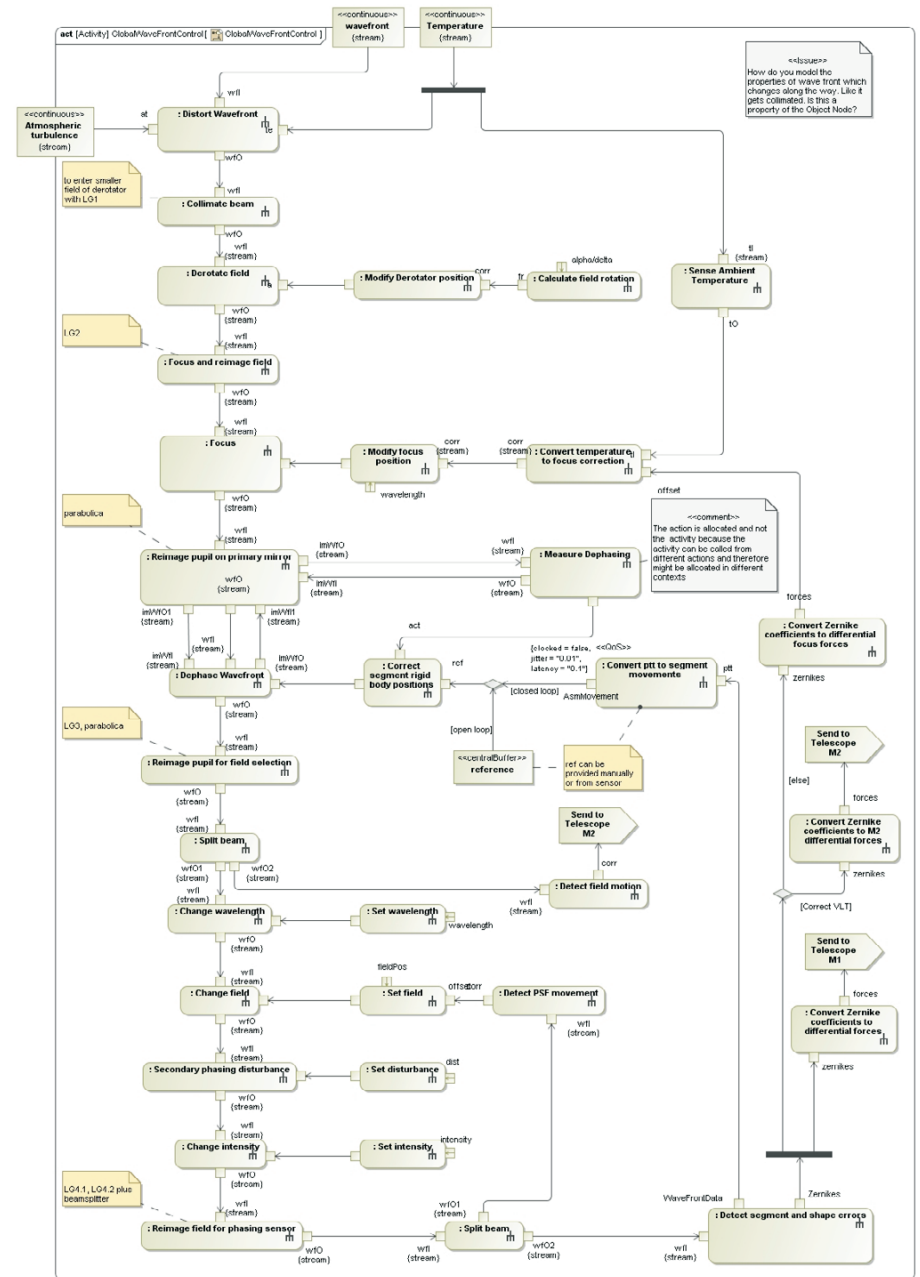


Figure 8. Activity diagram for APE wave-front control

Behavior

A complex system is much more than just the collection of its elements and their structural architecture, because its behavior derives from the collaboration of its parts. Therefore it is essential to capture the behavior of the system to be able to understand it. We have used activities to model the behavior of APE and its subsystems. SysML activity diagrams can be used to show the actions taken by the system and its data and control flow. Figure 8 shows the wave-front control of APE. It shows at the same time the physical effects of the system (like distortion of the wave front), as well as sensing, actuating actions, and control flows.

Data

Another aspect is the information or data handled by the system. SysML provides block definition diagrams for the definition of data, and it provides internal block diagrams, activities, and sequence diagrams for data usage and flow. APE uses SysML “dataTypes” to define the data of APE and its subsystems. Figure 9 shows the composition structure of a measurement and its relation to movements of the segmented mirror (“AsmMovement”). Those data types are used to define ports in IBDs and objects in activity diagrams.

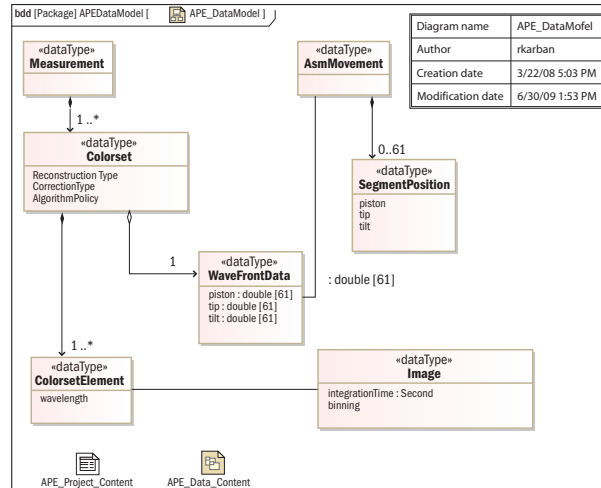


Figure 9. Block definition diagram of APE wave-front data

Verification

For every large system, verification is an essential part of the system acceptance in order to prove that the system meets its requirements. SysML supports the modeling of test cases with a specific model element, “testCase.” The “testCase” element can be used at different levels for component integration, software integration, and system integration. Furthermore, APE has two different test contexts: verification in the lab and in the sky.

Model Library and Systems-Engineering Profile

Besides the modeling of the different aspects, the APE modeling project provides a model library and an SE² profile. Data types and model elements that are frequently used are modeled in a model library to increase reuse. Abstract types are

used as place holder for specific building blocks. They are classified in different catalogue packages (figure 10).

Catalogues can be easily extended by using inheritance. Furthermore, the preliminary design of a system can initially work with an abstract type (when the detailed requirements are yet unknown) and decide later which specific type to use for the implementation. A generic connector gets a different context-specific pin assignment by inheritance. For each specific assignment a separate specialization is needed. The SE² profile provides the project-specific extensions to SysML: stereotypes like “objective,” constraints in element usage, enumerations, and so on.

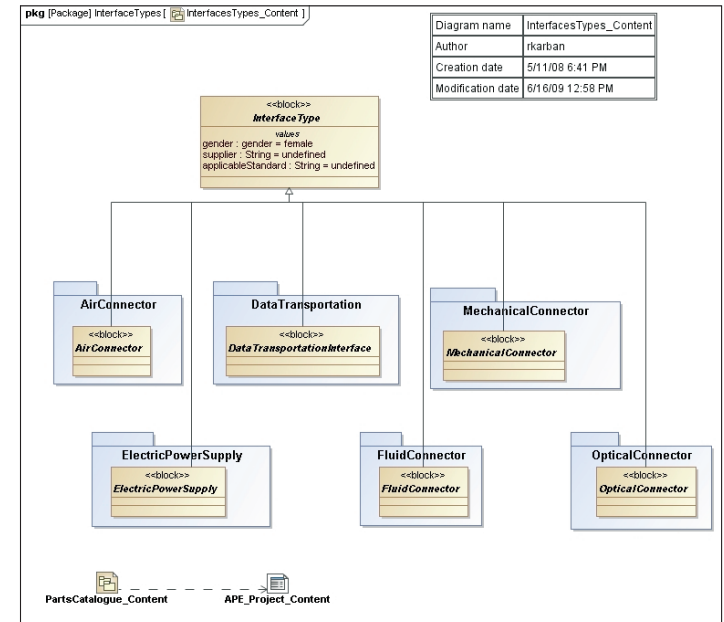


Figure 10. APE abstract types

Modeling Challenges

SysML is a new language. This creates two inherent challenges: Is SysML sufficiently mature for real projects, and is it accepted by a wide range of systems engineers? Especially the fact that SysML is based on UML sheds a special light on these challenges. Could a modeling language that was initially defined for software development be used to model systems, and will systems engineers accept a language with origins in the software discipline? An overall result of our project is that this question can be answered yes.

The APE project is a very good challenge for SysML. It is complex and interdisciplinary without a special focus on software; it is a real system and not the simplified coffee machine so often used as demonstration project. Although we found that SysML is practicable to model complex systems, we have made a list of the language’s shortcomings. The most significant ones are these:

- Variant modeling
- Connection of nested blocks
- Grouping of interfaces with nested ports

- Logical vs. physical decomposition
- Functional multilayer abstraction
- Reuse of blocks, allocation, and instances
- Structural multilayer allocation
- Defining quality of service
- Transition to UML for software
- Configuration and quality control
- Navigability

There are four aspects related to these:

Notation: It is a real challenge for a modeling language to provide an interdisciplinary notation for complex systems. It must be easy to understand and be capable of modeling details unambiguously.

Model: Behind the notation is the real model, i.e., the data structure and semantics of the information.

Tool: The implementation of the SysML specification is a challenge for tool vendors.

Methodology: SysML is a language without any methodology. You need a methodology or at least some best practices for good modeling.

In the following sections, we discuss one issue related to each of these aspects.

Notation: Connection of nested blocks

Figure 11 shows the connection of control-system elements with telescope elements: for example, the sensor cabinet (“sensCabinet”) is connected with the “coolantReturn” and “coolantSupply.” The solid line is a nested connector. It crosses the boundaries of the encapsulated system blocks.

If the modeler would like to hide the internal structure of the telescope in figure 11, the nested connector would also be hidden. Typically we still want to see that there is some kind of relationship between the telescope and APE. SysML doesn’t provide a presentation option to show the link in this case.

As a workaround we propose a standard port with stereotype *junction* that divides

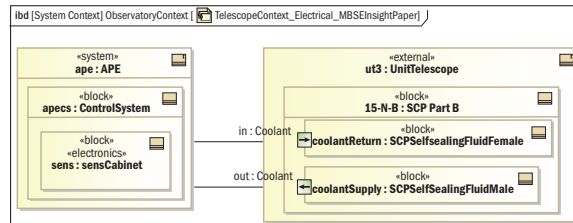


Figure 11. Nested connectors

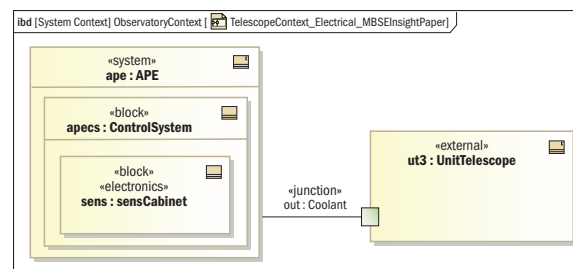


Figure 12. Junction ports for nested connector

the nested connector at each relevant boundary crossing in separate connectors. Figure 12 shows that the junction port resolves the issue (see also figure 6). We propose that SysML support the concept of junction ports, which would enable tools to offer convenience functions for nested connector modeling (we have issued this proposal to the SysML Revision Task Force).

Model

SysML supports two kinds of ports, that is, interaction points between a system block and its environment. The standard port provides or requests services, whereas the flow port specifies the flow of items inside or outside the system block. SysML has no explicit support of more complex ports that combine single, reusable ports. We propose nested ports for SysML. They allow a decomposing structure, the potential for reusability and individual delegation of item flows that go through this port (see figure 6). Nested ports conform to the abstract syntax of SysML. Therefore some tool vendors already provide this important modeling feature. SysML doesn’t say anything about the semantics of nested ports. Because we issued this proposal to the SysML Revision Task Force, a forthcoming SysML version will include a special kind of this concept called nested flow ports.

Tool

SysML has some modeling areas where creating and editing model information in a diagram is cumbersome. For example for requirements, which are often entered in bulk, it is easier to use a table than a diagram to enter or modify the data. The same is true for relationship modeling like the allocation of behavioral elements to structural system elements. It is easier to assign the relationships in a matrix than in a diagram with lots of arrows. SysML already defines table and matrix formats. Most tools provide them as limited read-only views on the model. What we need are table and matrix views with the ability to create and modify model elements.

Methodology

SysML is a pure language and doesn’t prescribe any methodology. For example, SysML allows one to model the “allocate” relationship between nearly any model elements. But where is this feature useful in a specific project? How can the relationship be determined from the model? What are the consequences of having an “allocate” relationship between two elements? You don’t find answers to these questions in the SysML specification. While there are some books that describe methodologies for modeling systems with SysML (Friedenthal 2008; Weilkiens 2008), our MBSE Challenge Team has found some best practices and modeling guidelines to complement them. Last but not least, each project needs its own specific set of methods.

Configuration and Quality Control

As soon as a common project model is created and more than one person uses it, configuration control becomes a fundamental requirement. In particular, consistent linking among model elements must be ensured. Individual changes must be traceable as well as creating visual differences to follow in detail what has changed where. Due to the extensive linking, side effects (introduced by changes) can go unnoticed and corrupt the model. This can only be mitigated by establishing rigorous configuration-management practices and using tools that allow rollbacks.

Experiences from a New Project — E-ELT Telescope Control System

The European Extremely Large Telescope is a telescope with a primary mirror of 42 m in diameter, composed of 984 hexagonal segments, and four other mirrors with diameters ranging from 2.5 m to 6 m. Figure 13 shows the E-ELT in comparison with the Very Large Telescope and the Roman Coliseum, which might be an indicator for its complexity.



Figure 13. Size comparison of the European Extremely Large Telescope (left), the Very Large Telescope (center) and the Roman Coliseum (right). (Image created by ESO; reprinted by permission.)

The telescope consists roughly of 10,000 tons of steel and glass in a structure the size of a big football stadium; it needs 20,000 actuators, some of which have to be controlled to location accuracies within a nanometer and to angular accuracies within 0.02 degrees. It requires high-performance computation up to 700 Gflop/s, and data transfers rates of up to 17 Gbyte/s. The control system has to deal with about 60,000 I/O points, 15 subsystems (one particular subsystem requires the coordination of 15,000 actuators alone), and interacting, distributed control loops with sampling rates ranging from 0.01 Hz to several kHz.

The telescope control system (TCS) includes all the hardware, software, and communication infrastructure required to control the telescope and the dome. Many subsystems will be contracted and have to be properly integrated. Therefore, the TCS defines for the subsystems the interfaces and requirements as well as

standards to be used for the field electronics, software, and hardware.

The common project model, where all TCS-relevant components of the observatory are logically represented, enables the development team to

- share a common, consistent view of the system, like the context in figure 14;
- structure requirements, in combination with a requirements management tool;
- describe the complex behavior of the system;
- define the architecture and detailed design;
- standardize meta-architecture across the various subsystems;
- allocate function to structure and allow full traceability; and
- analyze different design variants.

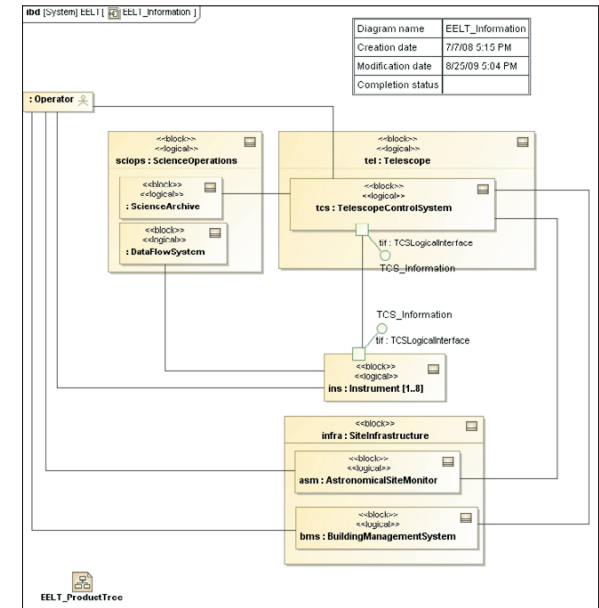


Figure 14. TCS context diagram

TCS architecture is highly data-oriented, making it well suited for representation with SysML constructs like activities and pins, blocks and ports, and state machines. The design of the TCS covers different views as control, electrical, and even mechanical because components have to be mechanically mounted.

Diagrams are extracted from the model to create paper-based documentation, as required by the project. The reporting and plug-in facilities of the modeling tool allow the engineer to automatically create the recursive structure as defined by the guidelines, and to make cost estimates using a predefined parts catalogue and estimates of the required communication infrastructure to accommodate the necessary throughput.

The successful application of MBSE to a project of this scale was only made possible by the existence of the guidelines produced by the SE² Challenge Team. We consider those guidelines, together with their accompanying examples, a precondition to allow one to focus on the content, rather than on hands-on technicalities.

Conclusions

The SE² Challenge team used an existing, complex system to create a

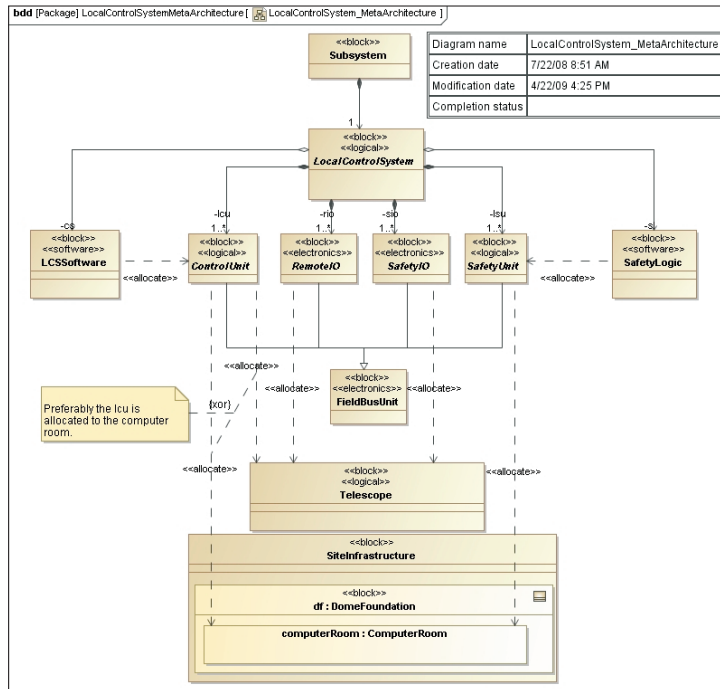


Figure 15. Meta-architecture of the local control system

comprehensive SysML model and solve common daily modeling problems. The engineering team of the E-ELT TCS is able to successfully apply the established guidelines, model structures, and modeling procedures to a new large-scale system in the optical telescope domain. The results demonstrate that SysML is an effective tool to document the complexity of requirements, interfaces, behavior, and structure, and is instrumental to enhance the traceability between requirements, design, and verification and validation.

A formal language and an adequately strict tool enforce structured thinking and a detailed description of the problem at hand. This increases the consistency and reveals undefined or unclear parts of the problem. Some limits of SysML were reached, because it does not offer out-of-the-box concepts for optical or electrical engineering. However, this can be overcome by extending the language using domain-specific profiles.

The most important value in a systems-engineering project is to have a common understanding among all stakeholders; therefore, fancy SysML constructs should be avoided when starting up. Even if SysML is a very good tool for communication and improving understanding, not all aspects of systems engineering can be fully covered by modeling with SysML, and systems engineers will still need the expressiveness and level of detail of traditional engineering activities like numerical

analysis, simulation and prototyping to handle non-functional aspects like safety, security, reliability, usability, and logistics.

References

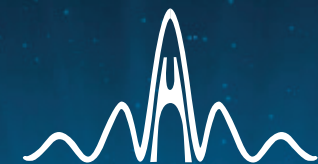
- ESO (European Southern Observatory). 2009. Very large telescope information. ESO (Web site). <http://www.eso.org/public/astronomy/teles-instr/paranal.html> (accessed 3 Nov. 2009).
- Friedenthal, S., A. Moore, and R. Steiner. 2008. *A practical guide to SysML: The systems modeling language*. Burlington, MA: Elsevier/Morgan Kaufmann.
- Gonte, F. Y. J., N. Yaitskova, P. Dierickx, R. Karban, A. Courteville, A. Schumacher, N. Devaney, et al. 2004. APE: A breadboard to evaluate new phasing technologies for a future European Giant Optical Telescope. In *Ground-based Telescopes*, ed. Jacobus M. Oschmann, Jr. (vol. 5489 of *Proceedings of SPIE* [The International Society for Optical Engineering]), 1184–1191. Bellingham, WA: SPIE.
- Ogren, I. 2000. On principles for model-based systems engineering. *Systems Engineering* 3 (1): 38–49.
- OMG (Object Management Group). 2008. *OMG Systems Modeling Language (OMG SysML™): version 1.1*. OMG document no. formal/08-11-02. <http://www.omg.org/spec/SysML/1.1/> (accessed 3 Nov. 2009).
- Weilkiens, T. 2008. *Systems engineering with SysML/UML: Modeling, analysis, design*. Amsterdam: Morgan Kaufmann OMG Press/Elsevier.

Subscribe to PPI's FREE Systems Engineering Newsletter

PPI's Newsletter (SyEN) contains scores of news and information on developments in the field as well as a technical article for the technical project professional, on a monthly basis.

Subscribe at:

<http://www.ppi-int.com/newsletter/systems-engineering-newsletter.php>



PROJECT PERFORMANCE
INTERNATIONAL

www.ppi-int.com

Leading-Edge Training - 5-Day Courses and Workshops

Systems Engineering

for Technology-Based Projects and Product Developments

www.ppi-int.com/training/systems-engineering-course.php

Presented by Mr. Robert Halligan

Provides valuable principles, and effective methods with which to implement those principles, taking a single system in workshop format from requirements through to design optimization.

A world renowned short course in Systems Engineering, that has been presented to over 4,500 professionals on six continents. Relevant to beginners and seasoned veterans alike, offering a structured approach from cradle to grave. This course provides valuable information on guidelines, standards, templates, matched with real-life examples and practical workshops, allowing theory to be put into practice.

Who Should Attend?

- Project Directors
- Project Advisers
- Project Managers
- Engineering Managers
- Systems Engineers
- Software Engineers
- Design Engineers and all other Engineers
- Consultants

Dates	Location	Code	Course Fee*
7 - 11 December, 2009	Las Vegas, USA	P006-383	USD 2,420
4 - 8 January, 2010	Singapore	P006-377	SGD 3,595
11 - 15 January, 2010	Amsterdam, The Netherlands	P006-389	EUR 2,075
1 - 5 February, 2010	Melbourne, Australia	P006-391	AUD 2,960
15 - 19 February, 2010	La Spezia, Italy	P006-393	EUR 2,075
12 - 16 April, 2010	Las Vegas, USA	P006-397	USD 2,890
26 - 30 April, 2010	London, United Kingdom	P006-398	GBP 1,595
10 - 14 May, 2010	Pretoria, South Africa	P006-399	AUD 2,700
31 May - 4 June, 2010	Helsinki, Finland	P006-400	EUR 2,075

Cognitive Systems Engineering

www.ppi-int.com/training/cognitive-systems-engineering.php

Presented by Dr. Gavan Lintern

This world-leading course teaches methods of cognitive analysis and cognitive design, and illustrates how they can be applied to enhance human systems effectiveness and safety within system development and acquisition. The course covers two complementary frameworks for the analysis and design of cognitive work. The basic tools of each are described and then demonstrated to illustrate how they can be applied to enhance human systems integration within systems development and acquisition.

Experiential design exercises give delegates practical experience with these techniques. The course, while standing alone, complements PPI's 5-day systems engineering course.

Who Should Attend?

- All designers of systems which include people
- Human system integrators
- Specifiers of user interfaces
- Designers of user interfaces
- Designers responsible for usability
- Systems engineers
- Software engineers who implement user requirements
- System safety engineers
- Engineering managers and team leaders

Dates	Location	Code	Course Fee*
19 - 23 April, 2010	Melbourne, Australia	P1082-4	AUD 3,845
24 - 28 May, 2010	London, United Kingdom	P1082-5	GBP 2,120
7 - 11 June, 2010	Las Vegas, USA	P1082-6	USA 3,990
8 - 12 November, 2010	Adelaide, Australia	P1082-7	AUD 3,845

These courses are available publicly and on-site. Visit www.ppi-int.com to register your interest in PPI's leading-edge training.



Project Performance International
PO Box 2385, Ringwood North
Victoria, 3134, Australia

Tel: +1 888 772 5174
Fax: +1 888 772 5191
contact@ppi-int.com

Leading-Edge Training - 5-Day Courses and Workshops

Requirements Analysis & Specification Writing

www.ppi-int.com/training/requirements-analysis-specification-writing-course.php

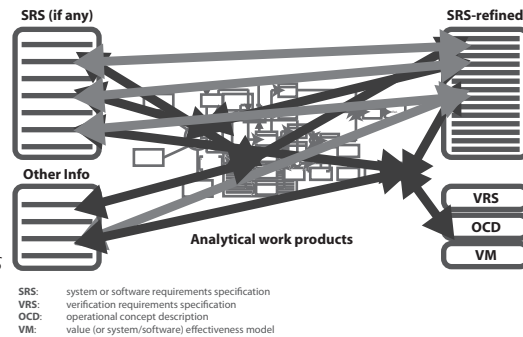
Presented by Mr. Robert Halligan

Requirements problems are at the top of the list of why projects go wrong. The 3-day Requirements Analysis module takes participants step-by-step in workshop format through a practical requirements analysis, to achieve an objectively adequate standard of requirements. In Specification Writing (2-days), you will learn how to structure a requirements specification, and how to best express requirements and other text in English. Real-world examples generate group discussion to assist in understanding.

Who Should Attend?

- Acquirer Personnel
- Supplier Personnel
- Developer Personnel

and anyone else who, in any capacity, deals with requirements



Dates	Location	Code	Course Fee*
18 - 22 January, 2010	Amsterdam, The Netherlands	P007-259	EUR 2,075
22 - 26 February, 2010	Las Vegas, USA	P007-260	USD 2,890
19 - 23 April, 2010	Melbourne, Australia	P007-261	AUD 2,960
30 Aug - 3 Sep, 2010	Las Vegas, USA	P007-262	AUD 2,960
6 - 10 September, 2010	Amsterdam, The Netherlands	P007-263	EUR 2,075
27 Sep - 1 Oct, 2010	Cape Town, South Africa	P007-264	AUD 2,700
11 - 15 October, 2010	Adelaide, Australia	P007-265	AUD 2,960
22 - 26 November, 2010	Melbourne, Australia	P007-266	AUD 2,960

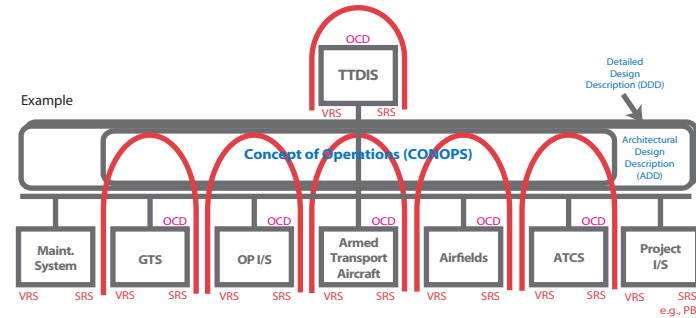
OCD & CONOPS

in Capability Development

www.ppi-int.com/training/ocd-conops-course.php

Presented by Mr. Robert Halligan

This course is a 5-day immersion in the development of military capability, with a focus on problem definition, Operational Concept Description (OCD - how the capability, and each element of its solution, will be used), and the concept of operations (CONOPS - how the military outcome is to be achieved).



The course is consistent with a systems approach to problem solving, as advocated by defense administrations worldwide. Systems engineering is an interdisciplinary, collaborative approach to the engineering of system solutions (of any type).

Dates	Location	Code	Course Fee*
22 - 26 March, 2010	Las Vegas, USA	P958-06	USD 3,990
17 - 21 May, 2010	Pretoria, South Africa	P958-07	AUD 3,500
24 - 28 May, 2010	Adelaide, Australia	P958-08	AUD 3,845
9 - 13 August, 2010	Adelaide, Australia	P958-09	AUD 3,845

These courses are available publicly and on-site. Visit www.ppi-int.com to register your interest in PPI's leading-edge training.



Project Performance International
 PO Box 2385, Ringwood North
 Victoria, 3134, Australia

Tel: +1 888 772 5174
 Fax: +1 888 772 5191
contact@ppi-int.com

Model-based Systems Praxis for Intelligent Enterprises

Jack Ring, jack.ring@incose.org

Motivations

At the 2007 International Workshop Sandy Friedenthal asked the Fellows Committee to participate in the Model-based Systems Engineering Grand Challenge. I volunteered to demonstrate MBSE of intelligent enterprises. Fellow John Clymer, professor of engineering at California State University, Fullerton, immediately joined the project, as did one of his grad students, Jose Garcia, Jr., a Boeing employee. Frank Lillehagen of Commitment AS, Norway, agreed to share ideas and findings from his work with clients on the knowledge-management aspect of an intelligent enterprise.

Expeditions

Our work is informed by the research and operating experiences of several participants who joined this project along the way, and by notables such as INCOSE Fellow A. Wayne Wymore (*Model-based Systems Engineering*, Boca Raton, FL: CRC Press, 1993), INCOSE Pioneer, John N. Warfield (*Understanding Complexity: Thought and Behavior*, AJAR, 2002), Thomas Gilbert's advice (*Human Competence: Engineering Worthy Performance*, Amherst, MA: HRD Press, 1996), and the 94 contributors to the INCOSE Intelligent Enterprises Working Group report, "About Intelligent Enterprises: A Collection of Knowledge Claims" (March 2007, INCOSE-TD-2007-001-01).¹

We decided to work from the specific to the general, starting with actual cases and evolving to general principles. Although SysML and AP233 were prominent in the MBSE initiative, we preferred to explore and understand the challenges inherent in modeling intelligent enterprises before bothering with the ways and means of doing so. We established three objectives: (1) to clarify the factors to be addressed in producing descriptive and prescriptive models of an intelligent enterprise, (2) to experiment with methods and tools for producing such models, and (3) to determine the effectiveness of candidate methods and tools by observing how well they have served those who engage in the real-world staffing and operation of intelligent enterprises.

We have explored intelligent enterprises in various business, education, and military domains. Our more intriguing focus is on the kind of intelligent enterprise that accomplishes systems engineering, probably the most complicated and ambiguous kind of enterprise.

1. Available at <http://www.incose.org/practice/techactivities/wg/intelent/docs/IEWGKnowledgeClaimsCollectionReport2007-0315.pdf>.

Results

Several results have been produced to date:

- Recommended edits to the INCOSE SE Vision 2020 report.
- Presented a paper at the Conference on Systems Engineering Research 2007.
- At IS07, which was themed "Intelligent Enterprises," we presented tenets and concepts regarding both the essence of intelligent enterprises as systems and of the model-based praxis of systems, notably, identifying, designing/architecting, engineering/constructing, adopting/adapting/assaying, and learning.
- Also at IS07, participated in a plenary panel that addressed the question, "Are New Systems-Engineering Principles Required to Treat Enterprises as Systems?"
- Also at IS07, we conducted a panel discussion in quest of a strategy for evolving a whole-systems-modeling capability. This capability was envisioned as 100,000 practitioners by 2015 performing at 10 times better productivity and innovation than the 2007 norm.
- Findings from this expedition were presented at the IEEE Systems Conference, January 2009, with a recommendation that the expedition be repeated according to the Interactive Management method with professional facilitation involving ten to fifteen exemplars of the praxis of systems.
- Conducted a tutorial in the model-based systems engineering of intelligent enterprises at the International Conference on (Inter-) Enterprise Systems Theory and Theory in Action, 2007.
- Presented a progress briefing at IW08.
- Presented eight papers at the Conference on Systems Engineering Research 2008 (six by John Clymer's graduate students).
- Worked with Ralph Hodgson to introduce development of formal ontologies into MBSE deliberations. Started SysMO Forum at sysmoforum@googlegroups.com.
- Presented a progress report at IS08 regarding both the capabilities modeling approach (Clymer) and the knowledge-management approach (Lillehagen).
- Presented a progress briefing at IW09 (thanks to Jose Garcia, Jr.).
- Produced three straightforward test cases that are representative of management choices in an intelligent enterprise. These can help determine whether any proposed modeling language or tool is adequate.
- Produced a companion paper, "Executable and Integrative Whole-System Modeling via the Application of OpEMCSS and Holons for Model-based Systems Engineering," by Jose Garcia, Jr., in this issue of *INSIGHT*.

All of these results reveal that there is much more to be done because the current standards, exhibits, guides, and handbooks for systems engineering do not explicitly provide for the model-based systems engineering of intelligent enterprises, particularly those enterprises that are intended to accomplish systems engineering.

Intelligent System/Enterprise Tenets

The following are a number of key current tenets regarding intelligent enterprises as systems:

- An enterprise facilitates commerce between a marketplace and supplierplace.
- An enterprise consists of two or more people taking action with limited resources toward mutual purpose.
- Intelligent means that the purpose is mutual not only to the people taking action but also to the stakeholders and to the principles of systems and society, all while thriving throughout unpredictable change.
- An enterprise must add sufficient value to marketplace–supplierplace commerce to sustain market standing, productivity, innovation, and liquidity.
- An enterprise is not just an information-technology system or an enterprise-architecture framework. An enterprise breathes, perspires, and celebrates. Action consists of orchestrating “frontal lobes” and suppressing errors.
- An enterprise is bounded only by the set of requests to which it responds, voluntarily or not.
- An enterprise is intelligent when it can determine the gap between its situation and its goal and formulate action that reduces the gap. “Intelligent” is a mode of behavior, not a measure of performance.
- An enterprise can be thought of as a system having context (stakeholders and requests), content (operators and operands), structure (relationships among the operators and operands and among the relationships), and behavior (the stimulus-response characteristic exhibited by the system).
- Enterprises differ in key dimensions of extent, variety, and ambiguity.
- Intelligent enterprises are implicit systems in which, in the limit, every action changes: (a) the gradient of a relationship (adjusts), (b) the pattern of relationships among existing content (adapts), and (c) even the content of the system (co-aligns).


MBSE Praxis

The praxis of intelligent enterprises should be informed by these key current tenets:

- MBSE contributes to a project in three ways: (1) establishing a common language throughout the project, e.g., producing an ontology; (2) modeling the intended system, and (3) converging creativity to closure.

- Modeling an intelligent enterprise or system entails building a descriptive model of the problematic situation (such as Boardman’s Systemigrams or systems dynamics modeling) and a prescriptive model of the intervention system (such as SysML, OpEMCSS, or WattSystems).
- Both models must represent: (a) the truth, the whole truth, and nothing but the truth; and (b) the minimal implicate order (necessary and sufficient information for those who will develop and test the system). Minimal implicate order typically includes explication of input/output, performance, cost, technology, test, and tradeoff. Technology considerations include thermodynamics, informatics, biomatics, teleonomics, human social dynamics, economics, and ecologies.
- The order of design decision and priorities for tradeoff are (a) design for enthusiasm, (b) design for learning, (c) design for culture, as in axiology, including a quality ethic, integrity, and an articulation of a standard of care regarding all stakeholders.
- Because any model of an intended system is a theory, the model must (a) reflect uncertainties (no deterministic models), (b) provide an estimate of uncertainty in its output, and (c) be accompanied by the modeler’s recommendations on ways of determining the limits of applicability of the model.
- MBSE of an intelligent enterprise must be accomplished at the rate of change of the problematic situation, the resource limitations, and the knowledge growth or refinement achieved. An enterprise is in continual flux. No activity, process, or task is ever repeated exactly. An enterprise model that represents less than the requisite implicate order or is not maintained at the requisite pace will be inadequate, or worse, misleading. The enterprise that learns fastest wins, but only if invention fosters innovation.
- A system that is capable of adjusting, adapting and co-aligning must contain a model of itself such that the model can be used to examine alternative change scenarios before commitment. This indicates a highly cellular system with a robust means of orchestrating cell participation based on the request, response, and reward scenarios to be honored.
- The appropriate language and modeling tool must accommodate implicit (context-sensitive), and continually learning (knowledge-adapted) constructs that accurately reflect the characteristics and properties of the underlying technologies.

Invitation

Anyone interested in participating in current plans or in conducting a diverse expedition is encouraged to contact the author. We are especially interested in anyone willing and able to tackle the three test cases using SysML. 

The Challenge of Model-based Systems Engineering for Space Systems, Year 2

C. Delp, chris.delp@incose.org; L. Cooney, lauren.cooney@jpl.nasa.gov; C. Dutenhoffer, chelsea.dutenhoffer@incose.org; R. Gostelow, roli.gostelow@incose.org; M. Jackson maddalena.jackson@incose.org; M. Wilkerson, marcus.wilkerson@incose.org; T. Kahn, theodore.kahn@incose.org; and S. Piggott, stephen.piggott@incose.org

The INCOSE Space Systems Working Group (SSWG) Challenge Team was formed to address the challenge of applying model-based systems engineering to a real-world space system. In our first year of work the team explored the use of MBSE for space systems (Delp et al. 2008) using the FireSat example from *Space Mission Analysis and Design*, edited by Wiley J. Larson and James R. Wertz (1999). In the second year we are building on the existing FireSat model to explore how reusable and descriptive model libraries can describe and manage information for trade studies as well as expanding the examples of modeling space systems.

Challenge Team Objective and Approach

An objective of the INCOSE MBSE Challenge is to demonstrate the applicability of MBSE to a realistic, sharable problem space. The SSWG has chosen a fictitious space mission. Over the last two years, the team has developed a model of FireSat using the Systems Modeling Language by treating Larson and Wertz's *Space Mission Analysis and Design* (SMAD) as a traditional document-based design, and mapping textual artifacts to visual model-based constructs. Development of the SysML model follows INCOSE's object-oriented systems-engineering method (OOSEM), outlined by Friedenthal, Moore, and Steiner (2008), and the JPL-developed State Analysis methodology (see <http://mds.jpl.nasa.gov>). The team has also developed a library of reusable space-system model constructs within the FireSat model itself. The members of the SSWG challenge team represent NASA's Jet Propulsion Laboratory, Goddard Spaceflight Center and Glenn Research Center, the Japanese Aerospace Exploration Agency, the European Space Agency, the Canadian Space Agency, the Boeing Company, the Lockheed Martin Corporation, and student teams at the Massachusetts Institute of Technology and the Georgia Institute of Technology.

FireSat is a conceptual mission used by Larson and Wertz to provide a unified context for the practical examples in their textbook. The FireSat mission calls for a space-based system to detect, analyze, and monitor forest fires, with data ultimately provided

to the U.S. Forest Service. Larson and Wertz then use the FireSat context to document examples of mission analysis and design, from mission characterization and requirements definition to specific space and ground subsystems. The advantage of using the FireSat example for the Space System Working Group's MBSE Challenge is that it is a non-proprietary example, and it offers complex and challenging problems. Though SMAD does not contain a complete documented system design, the text provides sufficient descriptions and artifacts to create model-based analogs corresponding to artifacts widespread in the space and aerospace industry.

The goal of this effort is not to demonstrate another way to simulate or numerically analyze a complex problem, but rather how a descriptive model can serve to unite design and analysis artifacts in a manner that is not possible with modern document-based office-automation software. This integrated model overcomes issues with scalability, consistency, and completeness; it also removes the mundane bureaucratic chore of churning out document updates, leaving the systems engineer with more time to engineer the system.

Design Representation: Integrated Models vs. Documents

Space-industry practitioners consider SMAD a foundation for standard practices. It contains the essence of how systems engineering is applied to space systems in current practice. Most of the FireSat examples are analytical design trades using models of the mission, system, and subsystems for the FireSat spacecraft. Although the scale and detail of information represented by the FireSat example are much smaller, FireSat is appropriate to represent the systems-engineering documents and artifacts used in flight-project architectures and designs for the purposes of this exercise.

The examples the SSWG team used from SMAD reflect information found in the corresponding documents typically used in industry to describe space systems. Although the examples do not represent complete documents, the information they contain is a reasonable representation. The SSWG team described the FireSat mission using SysML and stored it in a database. This procedure

.....
The benefit of formal modeling is that we can finally stop being ambiguous and say exactly what we mean.

— Robert Rasmussen, PhD,
 Chief Engineer, Systems
 and Software Division, Jet
 Propulsion Laboratory

Table 1. Systems engineering documents and Firesat examples

System Documents	FireSAT Example
Concept of operations	Mission objective, system requirements, orbit selection
Requirements (SRDs, FRDs etc.)	System requirements
Interfaces (ICDs, etc.)	Data-flow diagram, system requirements
Functional designs	Power functional decomposition
Architecture description	Mission objective
Analysis-specific engineering reports (trades, reliability, etc.)	Solar array selection, orbit selection
End-to-end information systems specification	Data-flow diagram

Note: Common systems-engineering documents are matched with parts of the Firesat example to establish the comparisons that will be made using models. Abbreviations: SRD, System Requirements Document; FRD, Functional Requirements Document; ICD, Interface Control Document

results in graphical representations replacing traditional documents. These representations still contain the same textual narrative from the document. Capturing the description in this way conveys the information that would have been referenced or repeated in the document through a single reusable location. Additionally, the description is now formal enough that it can be integrated with computer simulations, computer-aided design, and other quantitative design tools. The model captures design and trade exercises, which are automatically propagated to the systems specification, including requirements and other elements.

Models treat system and contextual elements and components as first-class citizens of the design. Each component of the system, the system itself, and each system that the system interacts with is explicitly defined in the model and contains all the information relevant to its description. As such the components can easily be counted and analyzed for complexity.

Although many documents are produced from templates, modeling facilitates a broader architecture for reusability. The Space Systems Library is a collection of descriptive model assemblies that encompass analyses, parts, and functions commonly used in the design of space systems. Space missions reuse concepts, methods, requirements, and hardware from prior missions in order to save time and money. A model library can capture this effectively, negating the need to duplicate and tailor previous mission artifacts. Using the analyses in the FireSat example, the Challenge Team constructed a library of reusable parametric models that represented physical laws such as those related to orbital mechanics, heat transfer, and structural analysis. These concepts attempt to capture parts of design that are flexible enough for reuse. This allows for rapid prototyping and design, since these concepts need only be parameterized for the specific mission that is applying the library.

Mission Design and Specification

The FireSat example outlines a fairly simple mission to provide a system of satellites for detecting, identifying, and monitoring forest fires. Expanding on the previous use-

case analysis and requirements (see Delp et al. 2008), the model now incorporates the requirements as measures of effectiveness (MOEs) and measures of performance (MOPs).

The SysML model captures context specifications such as the operational environment, the science target, and the enterprise that will carry out the mission objectives. As shown in figure 1, the mission's flight systems and

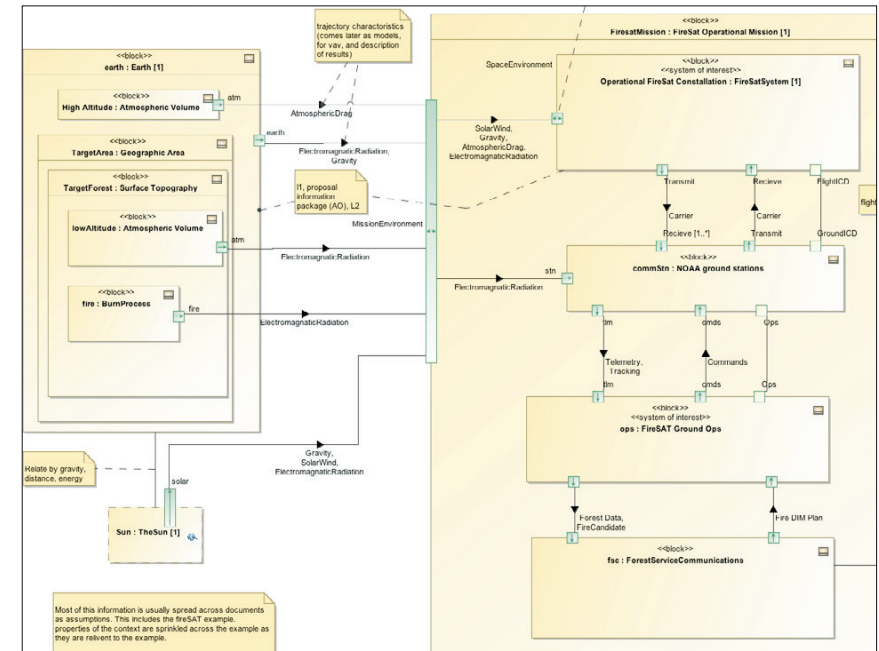


Figure 1. Mission internal block diagram

ground systems are depicted in a collective context. Ports describe the entire end-to-end set of interfaces: from fire and instrument to data downlink to forest-service notification. Since the model is formally described, it can also be automatically checked to ensure that the connections between these elements are correct and consistent. SysML notation makes this simple — no exotic software is needed to parse natural language.

This SysML internal block diagram depicts a high-level interface schematic for the whole FireSat domain. The earth and its various properties relevant to forest fire detection are related to the FireSat Mission and its components and enterprises. This mission-level view captures our desired FireSat system as a black box exposing only interfaces in the context of the operational mission. Every physical body, system, and enterprise that FireSat will interact with is captured in the model. Context sets the stage for behavior and metrics. A significant MOE for the mission is responsiveness: how quickly an existing fire is detected, identified, and reported to the forest service. Modeling these param-

ters as parametrics in SysML (instead of “shall” or “must” statements) now facilitates the flexibility to analyze the mission to determine if the given requirements can be accomplished. Conversely, we can use parametric models to define more-demanding requirements for frequency of coverage and less-demanding requirements for responsiveness (latency) based upon the

assumptions and constraints used for writing conventional requirements. The model allows us to perform these trades at any level of the system, from the product level to the subsystem level, to the level of individual pieces of hardware. At any of these levels the MOEs will report the effects of our trades on requirements. These expressions can now be tied to the orbit design.

Figure 2 shows the end-to-end response time depicted as a sum of all the mean times between the detection of a fire on the ground and the time the forest service receives the alert. This gives a clear quantitative understanding of how the FireSat constellation of satellites should perform with the other elements of the system.

Mission Orbit and Trajectory Design

Observation strategy affects the responsiveness MOE through the ground coverage MOP. The ability to detect fires quickly depends on ground-coverage rate, which in turn depends on number of satellites, their altitudes and orbits, and the payload used to collect data. Physics models of orbits are the foundation upon which different physical architectures can be compared against the MOEs. These MOEs encompass power, mass, and thermal concerns, such as the health and safety of the spacecraft, and the performance of the payload. To provide this foundation, example orbit and trajectory designs are currently being incorporated into the FireSat model, including satellite orbit parameters and equations of motion. Earth geometry as viewed from the spacecraft is also modeled for mission analysis. Coordinate systems are considered in order to appropriately describe the frame of reference in which analysis is performed. Model elements are associated with a reference frame, and transformations between reference frames (i.e., rotations and translations) are related using constraint block relationships. In figure 3, orbit analysis is described in an earth-centered inertial reference frame, while earth geometry viewed from

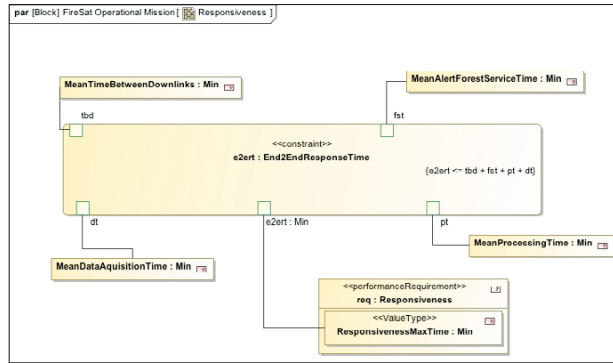


Figure 2. Responsiveness Measure Of Effectiveness. This SysML parametric diagram depicts how the MOE of maximum response time can be used to constrain the figure that sums the times from the different parts of the system’s behavior. Such a requirement is key to various trades in FireSat.

space is in a spacecraft-centered “RPY” (roll, pitch, yaw) frame. The properties of the coordinate systems are related to the appropriate model elements, as depicted in figure 3.

Since the entire mission must achieve a certain responsiveness, this translates to a requirement on FireSat depicted here as daily coverage quantity. The orbit design will directly affect how responsive FireSat is based on the rate of ground coverage.

These models are part of the Space Systems Library now and are usable for any mission.

Future versions of the FireSat model will benefit from this foundation as a reusable method for characterizing the design space and trading different physical architectures. From this exercise it is easy to envision a complete model of solar-system physics, astrodynamics, and even kinematics.

The FireSat system-of-interest block gives a complete black-box specification for functional goals and measures of performance, as well as persistent information stores, states, and interfaces. From this level, trades and other analyses that tie to the measures of effectiveness will allow changes in a physical design to reflect in the performance measures.

Logical and Physical System Architectures

To specify a spacecraft design that will accomplish the operational architecture modeling, developing the logical (similar to functional decomposition) and physical architectures is the next step. These functional architectures can be traded for how much and what types of capability the system requires. Similarly the physical alternative architectures can be traded against how they meet a particular functional scope. SMAD’s example of the FireSat power system provides traditional functional- and physical-design information. The purpose of the power analysis in the SMAD book is to determine the required size of the FireSat solar array and required battery

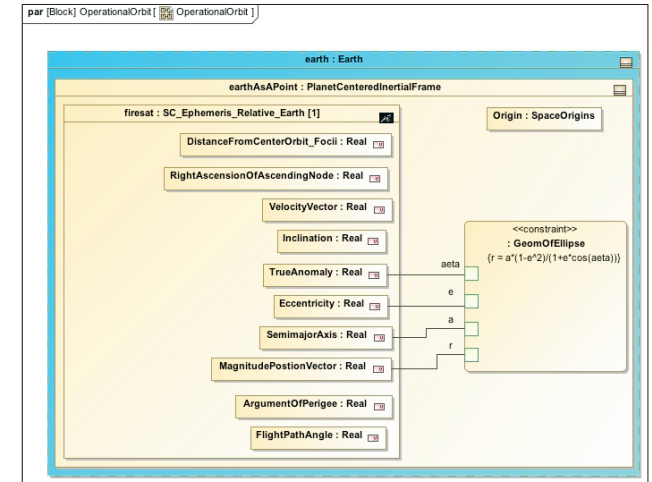


Figure 3. Orbit model. This SysML parametric diagram uses model-library definitions of earth, combined with kinematic reference frames and Kepler’s laws to depict a very basic orbit-geometry framework for a single spacecraft. This model can be used to separately capture any number of specific orbit configurations for trade study or design. Such data can be used to feed any number of execution tools for simulation analysis and trade. The earth sets the stage for the frame of reference that depicts the abstraction of the spacecraft and the earth (as the origin). For reference frames to make sense, the object the frame abstracts must be owned by the object—in this case the earth.

capacity in order to meet the power-system requirements. The book gives multiple types of power distribution schemes, solar cell types, and battery types. For the purpose of FireSat, we chose one combination, as shown in figure 4. This illustrates the complexity of the system being modeled and the trades that will be needed in order to choose a particular system.

The power system for FireSat can be viewed with an eye to many concerns. This parametric view shows how parts of the power system are constrained and what parameters are under constraint. The parametric model of the power system

is probably one of the most powerful examples of MBSE for space systems. Quantitative constraints captured from the SMAD book in the Space Systems Library illustrate how fundamental concepts common to space systems can be modeled and reused as part of a causal analysis. The example involves solving for eclipse and daylight time given power and area requirements and solving for area and power requirements given a particular orbit's daylight and eclipse time. These diagrams can be used to define many alternative combinations for the power system as well as the chosen baseline. It also shows how properties of the chosen orbit and mission-level information affect the power system.

2007–2009 Conclusions and Future Plans

In each example explored by the team, model-based systems engineering showed several advantages from documents. These advantages include clarity and communication. In the cases of figures (mission IBD and parametrics) we can clearly see in a single view all of the information that would ordinarily be a collection of disjointed documents. Instead, the model provides the information of concern as first-class citizens: this information is explicit with clear boundaries and semantics, and does not need to be assumed or referenced from another document.

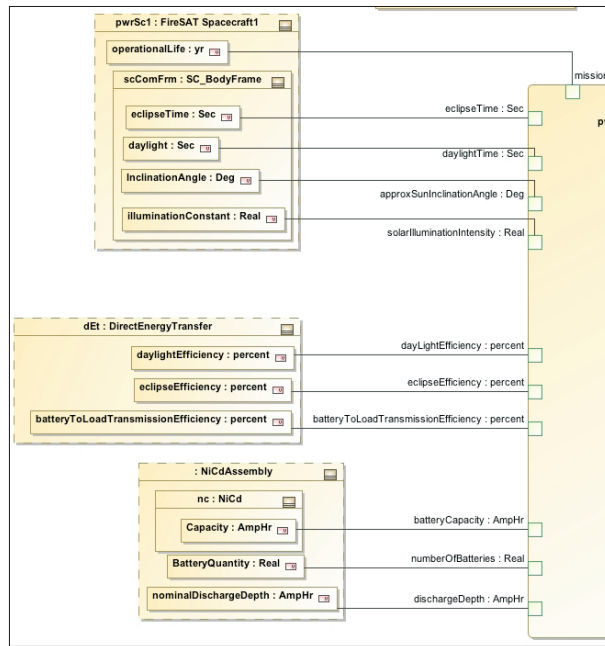


Figure 4. Power-system analysis. This SysML parametric diagram depicts a portion of the Firesat solar array trade as modeled. The constraint block that has been cut off contains all of the equations from the trade, along with the relationships and how it is connected to the power-distribution unit; the battery; and parameters from the spacecraft and the mission that affect power.

The information as shown in the model is the literal artifact. No churning will be required to update either the diagrams or the documents generated from the model. If the mission information changes, it will be propagated automatically.

The analytical relationships between responsiveness, coverage, and power production are explicitly related by parametrics. It may be intuitive that these are related based on orbit geometry and dynamics, but in the model, these equations and parameters are clearly documented in a way that can be just as easily used for calculation. Any changes to, for example, the orbit or number of spacecraft will propagate to our desired metrics.

Figure 4 shows a complete specification for the FireSat system. In one compact view, everything necessary to understand the system is presented. The document generated from this block would be multiple pages long and cover numerous pages in the book. In non-MBSE approaches, there would be a slew of documents containing the same information. Capturing the system in such complete detail without having to dive into the mire of all that text is revolutionary.

The library and use of orbit design and power analysis offers the advantage of reuse in a way that even the book itself cannot accomplish. This is a subtle but important point. These models contain not only the analysis but also the description of the analysis, since SysML is a descriptive modeling language. These foundations do not change, and assumptions captured and tested can now be used across a variety of missions without needing to start from scratch. For FireSat this would mean no ambiguity about the nature of the orbit design. The MBSE approach captures every aspect of the project, from flight software to launch vehicle to mission operations, and provides an unambiguous definition of how the spacecraft is meant to be flown. Finally, since all of the text narrative can be captured inside the model elements, the documents themselves can be generated from the model. The MBSE approach really involves comparing documents to the combination of documents *and* models.

The next year of effort for the FireSat project will include an attempt to incorporate a complimentary 3-D modeling tool called Satellite Tool Kit. This will allow us to explore systems-level modeling across tools and a comparison of 3-D animations vs. 2-D blueprints of space systems. Another area of interest to the team is the exploration of incorporating schedules and resources into the model. This will allow a systems model to better predict combined effects on cost schedule and risk in addition to product and performance topics. 📍

References

- Delp, C. L., with C.-Y. Lee, O. de Weck, C. Bishop, E. Analzone, R. Gostelow, and C. Dutenhoffer. 2008. The challenge of model-based systems engineering for space systems. *INSIGHT* 11 (5): 14–18.
- Friedenthal, S., A. Moore, and R. Steiner. 2008. *A practical guide to SysML: The systems modeling language*. Burlington, MA: Elsevier/Morgan Kaufmann.
- Larson, W. J., and J. R. Wertz, eds. 1999. *Space mission analysis and design*. 3rd ed. El Segundo, CA: Microcosm.

Integrating System Design with Simulation and Analysis Using SysML

Russell Peak, russell.peak@incose.org; Chris Paredis, chris.paredis@gatech.edu; Leon McGinnis, leon.mcginis@gatech.edu; Sanford Friedenthal, sanford.friedenthal@incose.org; and Roger Burkhart, roger.burkhart@incose.org

Modern life depends on the correct and efficient operation of complex technical systems. The OMG Systems Modeling Language (OMG SysML™) is one response to the challenge of effectively integrating a broad range of disciplines across a spectrum of system lifecycle activities—from product conceptualization through design, manufacturing, distribution, operation, support, and finally to end-of-life processing. SysML provides the ability to create formal models that express system requirements, structure, behavior, and parametrics, and that interoperate with other types of descriptive and analysis models to support a broad range of systems modeling. This article describes work that demonstrates this ability in the context of two mechatronics-related testbeds: (1) designing hydraulic excavator systems and associated manufacturing processes, and (2) developing and operating mobile robotics systems. INCOSE's Modeling and Simulation Interoperability (MSI) Team as part of the Model-based Systems Engineering Challenge did the work. This article summarizes the results from the over-130-page Phase 1 report (covering August 2007 to July 2008),¹ and highlights progress from Phase 2 (August 2008 to July 2009).

In the Phase 1 work for the product domain of hydraulic excavator systems, both mechanical and hydraulic systems are designed and simulated, and SysML models are used to integrate design and analysis models both within each design discipline and across disciplines. In the manufacturing domain, the disciplines that are integrated using SysML models include capacity planning, factory layout, process engineering, and production planning. The design and manufacturing domains are integrated using a SysML model of the engineering bill of materials and a common model of measures of effectiveness.

The primary achievements of the MSI team in the Phase 1 excavator case study are these: (1) capturing information about the

1. R. S. Peak, C. J. J. Paredis, L. F. McGinnis, S. A. Friedenthal, R. M. Burkhart, et al., "Integrating System Design with Simulation and Analysis Using SysML: An Excavator Testbed" (Phase 1 Final Report [version 1.2] of INCOSE Modeling and Simulation Interoperability Team, 2009), available at <http://www.pslm.gatech.edu/projects/incose-mbse-msi/>. (Check this Web address for additional items, including the Phase 2 report, expected in the winter 2010 time frame).

structure and behavior of the product and processes in a form that is readily reused for trade studies and design evolution; (2) automating key analysis steps by enabling tight integration between design authoring tools (e.g., CAD and SysML) and analysis tools (e.g., finite element analysis and discrete event simulation); (3) improving communication across disciplines; and (4) enhancing requirements traceability.

The results of these focused case studies have potential for broader impact: (1) the demonstration of MBSE and SysML may encourage a broader community of systems engineers to further explore SysML; (2) the software prototypes of interfaces between SysML and disciplinary design and analysis tools are potentially reusable in other domains (and some² have already been commercialized for that purpose); and (3) the SysML model examples serve as potential templates and reference models for other projects.

The Phase 1 work also identified three important areas for further work: (1) strategies for knowledge capture, including domain metamodels, profiles, and model libraries; (2) graph transformations as a powerful generic approach to model interoperability; and (3) a generalized philosophy and strategy for model integration called MIM—the Model Interoperability Method. Phase 2 continued to explore these areas and other extensions including a testbed for mobile robotics.

Background

Contemporary systems engineering is evolving to a model-based approach to address the complexity of large-scale heterogeneous systems. However, the model-based approach introduces its own challenges related to model interoperability and model management. A model-based systems-engineering approach must support collaboration and interoperability at several levels: across global organizations, between disciplines involved in the systems development effort, among design teams within a given discipline,

2. See for example <http://www.intercax.com/sysml/>, which has been aided by the Georgia Tech VentureLab incubator program.

between design and analysis efforts, and between development and manufacturing. As development and manufacturing become more intensely model-driven, tools and methods for systems engineering must be capable of managing and integrating this collection of models.

Project Context

This project leverages OMG SysML as a systems modeling language to address both model interoperability and model management. SysML provides a formal graphical language that has enough expressivity to be applied across diverse disciplines during system development. Thus, SysML holds the promise of directly enabling the necessary integrations through comprehensive system models, and also directly enabling integration and interoperability across a broad spectrum of models.

The work summarized here explores this promise. Our Challenge Team's project applies a model-based approach, using SysML modeling tools and other design and analysis modeling tools to the design of an excavator as one demonstration platform. The structure of the resulting SysML model is based on that outlined by the Object-Oriented Systems Engineering Method (OOSEM).³ In the system-development process, two primary product-design disciplines are engaged—fluid power systems design for manipulating the excavator arm, and structural design for the bucket and boom. In addition the integration of design with manufacturing is explored, which translates the engineering bill of materials (EBOM) into a manufacturing bill of materials (MBOM) with make/buy decisions, fabrication, and assembly operations.

This project was initiated with the overall objective to define the methodology, tools, requirements, and practical applications that demonstrate how to bridge a SysML system specification and design model with multiple engineering analysis and dynamic simulation models. Supporting objectives led to the definition of four specific tasks:

- Define a methodology for integrating a system model with multiple engineering analysis models and dynamic simulation models.
- Define requirements for the SysML specification, for SysML tools, and for analysis tools that are needed to support such a methodology.
- Demonstrate this methodology with several representative design and simulation models.
- Develop a roadmap for follow-on work.

3. For more information, see the Web site of INCOSE's Object-Oriented Systems Engineering Method (OOSEM) Working Group on INCOSE Connect at <https://connect.incose.org/>. As an example application of OOSEM, see the residential security-system case study (chapter 16) in Friedenthal, Moore, and Steiner's *A Practical Guide to SysML: The Systems Modeling Language* (Burlington, MA: Elsevier/Morgan Kaufmann, 2008).

These tasks were executed during Phase 1 of a twelve-month collaborative effort between Lockheed Martin Corporation and the Georgia Institute of Technology, with additional related work supported by Deere & Company and other efforts. The results of this combined effort are being used to help advance the practice of systems engineering within society as a whole and to support educational objectives at Georgia Tech and beyond. Phase 2, which has recently been completed, extended these techniques and the excavator testbed, and also added a mobile robotics testbed.

Figure 1 illustrates the excavator testbed and the strategy adopted to achieve the project objectives. Three distinct categories of commercial-off-the-shelf (COTS) tools are employed: SysML tools, traditional descriptive tools (product CAD, factory CAD, and Excel), and traditional analysis tools (Excel, FEA, math solvers, and simulation solvers). In addition, the project team developed the necessary interfaces to integrate the SysML modeling tools and other tools using a combination of COTS interfaces (e.g., VIATRA and ParaMagic) as well as custom interfaces developed in C#, Java, and Visual Basic.

Phase 1 Results

Figure 2 illustrates the resulting models and model interfaces from a perspective of model interoperability patterns. On the left are the COTS descriptive tools (in the box labeled *aO. Descriptive Resources*), and on the right are the COTS solver tools (labeled *eO*). The models in the middle boxes (labeled *bO*, *cO*, and *dO*) all are implemented as SysML models (figure 3). The box labeled *bO* is the federated systems model, which is primarily a descriptive model that collects together various *aO*-type models and augments them where needed. In this testbed the *bO* model combines both the system of interest (the excavator product) and its manufacturing system. Through the course of the project, reusable analysis and simulation building blocks that are context-independent (generic) were identified and collected into libraries, illustrated in the box labeled *dO*.

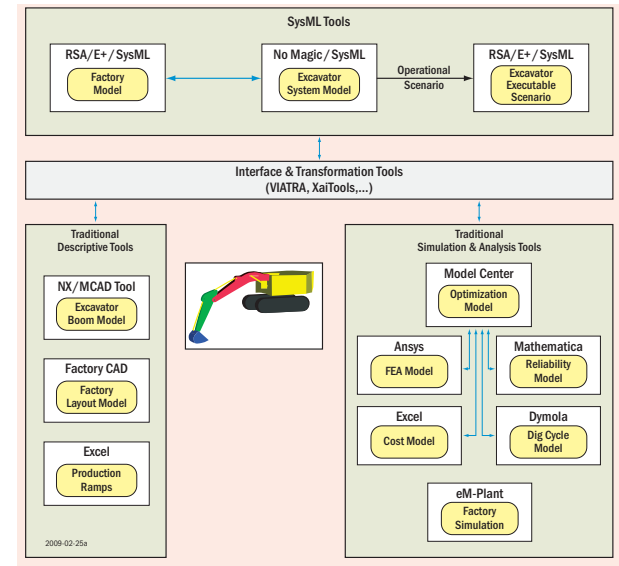


Figure 1. Excavator testbed (Phase 1): tool categories view

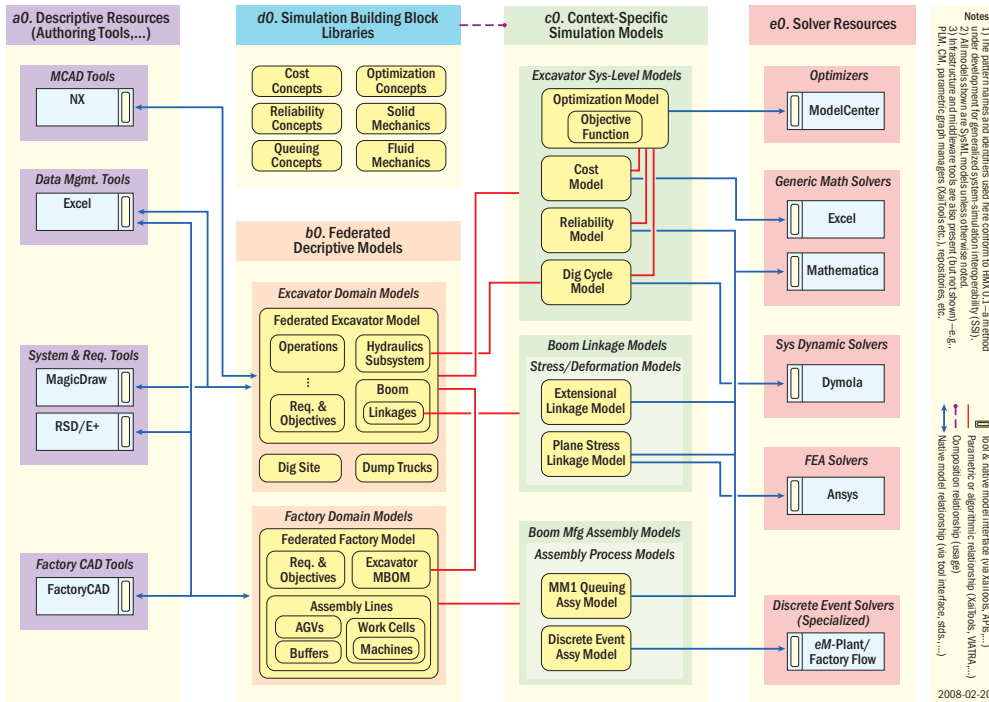


Figure 2. Excavator testbed (Phase 1): MIM model interoperability patterns view

Each context-specific simulation model in figure 2 (models in the box labeled *c0*) applies selected generic *d0* building blocks to the *b0* system for a specific purpose—typically to calculate values to verify one or more requirements or performance objectives. Each *c0* model is executed utilizing one or more *e0* solvers, which are typically general-purpose COTS solvers, but may also be specialized company-proprietary codes. The *c0* model pattern is the focal point for capturing knowledge about domain-specific analysis intent including idealization decisions. Depending on the nature of the *b0* system aspect being analyzed, these *c0* models range from fixed-topology analysis templates (which analysts create directly) to variable-topology analysis templates (which automatically generate a model with simulation topology that is specific to a particular design instance). In our excavator testbed the models for boom linkage are examples of the former, and the dig cycle hydraulics model is an example of the latter.

Each arrow in figure 2 represents a specific interface that required development, implementation, and testing by the project team. SysML modeling and interface development represented the major part of the R&D effort for the project. To demonstrate the model integration illustrated in figure 2, a series of scenarios was created for the excavator example. Figure 3 contains several thumbnail highlights from these scenarios (see the Phase 1 report for the actual figures and further

explanation). The initial scenario represents a design requirement (a target rate for moving dirt), and a marketing requirement (a target rate for selling excavators). The hydraulic and structural teams exercised a design process to achieve a satisfactory product design, and the manufacturing team translated the design into a manufacturing plan, capacity plan, and operational plan. The design process was then confronted with changed requirements. A higher required rate for moving dirt was found to necessitate a redesign of both the hydraulic and the structural subsystems; this revised product design then required a redesign of the manufacturing process. Finally, a higher target sales rate required a further redesign of the manufacturing process to support the corresponding increased manufacturing rate. Based on these results in the context of the excavator domain, we have demonstrated how to bridge a SysML system specification and design model with multiple engineering analysis and dynamic simulation models—i.e., the overall Phase 1 project objective has been met and exceeded.

Significance

The significance of these results is not in any single design decision or supporting engineering analysis—all of these could be done individually without

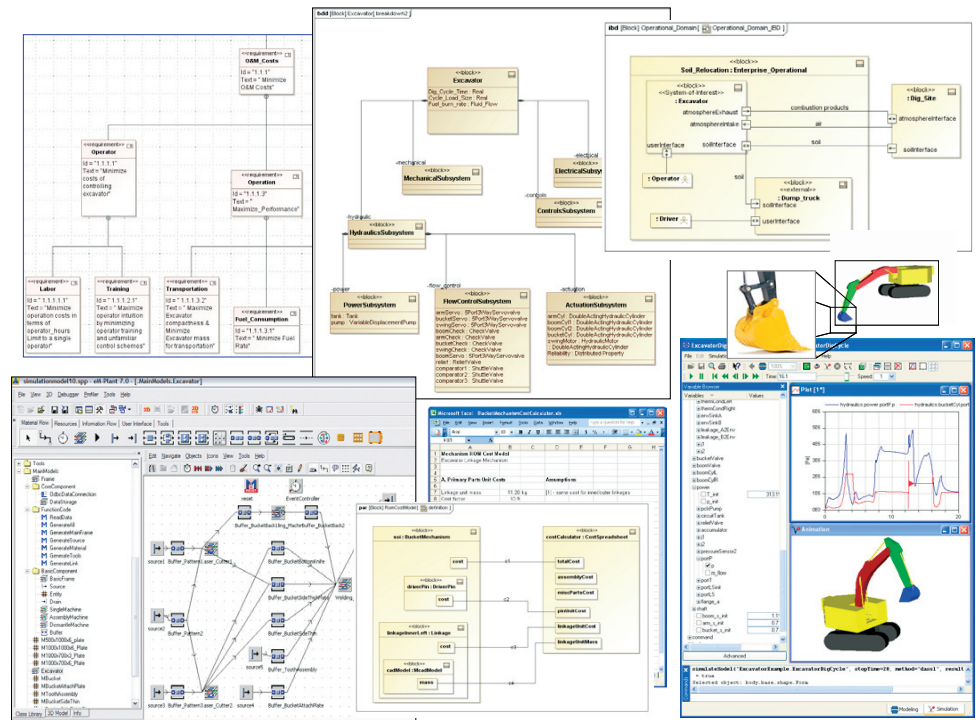


Figure 3. Excavator testbed (Phase 1): sample SysML diagrams and native solver models

the SysML modeling and interface development, albeit not as effectively. Rather, the significance is in the formal capture of modeling and design knowledge in a manner that enhances the integration of design and analysis as well as the reuse of knowledge and information. Integration fully or partially automates time-consuming manual processes and thus enables faster design analyses with less effort by the designer, which can result in faster design cycles and increased design analysis and trade-space exploration. The impact of knowledge capture and reuse is to enhance the capability of designers in terms of both design speed and design quality. The impact of improved model management is better visibility and communication across the entire design-manufacturing cycle, leading to fewer errors, earlier problem identification, and faster problem resolution.

This knowledge capture, integration, and reuse occurs at two levels. First, at the domain level, knowledge capture takes the form of libraries of concepts, modeling elements, and interfaces that are directly reusable in the design of other excavator products or other excavator manufacturing processes (and often in other domains beyond excavators). The use of these libraries does not necessarily require expertise in SysML; the captured knowledge can be accessed by potential users in “wizard” forms or in tools with which they already are familiar. Second, the captured knowledge takes the form of explicit system models that integrate the design across multiple product-design disciplines and between product design and manufacturing. The demonstration scenarios show that this knowledge has been captured in a manner that enables very rapid and inexpensive redesign of both the product and the associated manufacturing processes.

Phase 2 Highlights

Phase 2 (recently completed) has several facets including (1) extending the excavator testbed, (2) enhancing fundamental techniques, and (3) adding a new mobile robotics testbed. The graph transformation work has explored the capabilities of the MOFLON tool and its advantages over VIATRA, which was used in Phase 1. The EBOM-to-MBOM translation work has also applied MOFLON and tested its usefulness in that context. More simulation tools have been added into the factory simulation and parametrics testbed aspects, including AnyLogic and Matlab/Simulink. A new way of visualizing and interacting with SysML parametrics has been introduced: a flattened graph that gives a characteristic “panoramic” DNA-like view of the model. The new mobile robotics testbed demonstrates how SysML activity models can support system operations at both the planning and execution levels (including live updates and execution on physical units). Look for these and other additions in a public version of the Phase 2 report expected in 2010.

Future Work

Based on these experiences, a number of important next steps have been identified.

Generalization of model interoperability method. General-purpose solutions with enhanced robustness are needed in a number of areas:

- In the Phase 1 demonstration, the conversion from the EBOM to an MBOM was formalized in SysML, but the conversion itself was performed manually. Whether or not the conversion can be fully automated is an open question, but it is clear that a suite of tools could be created to assist in the conversion and to make it a more repeatable process.
- Graph transformation technology has proven to be an excellent means for implementing some of the key interfaces between different models, and it holds great promise for making a more efficient process for developing these interfaces. Additional work is needed to better understand this technology (e.g., regarding breadth and scalability) and to develop better tools for using it in this context.
- When a number of different teams work concurrently on a federated model (*b0*), there will naturally be inconsistencies between the federated model and its source submodels as various aspects change over time. Thus, methods are needed to manage these inconsistencies—to recognize and identify them, to provide temporary “work-arounds,” and to ensure resolution (e.g., semi-automated synchronization capabilities).
- Reusability represents a very important opportunity both for generic and for domain-specific modeling and integration. Fundamental questions—such as how to identify opportunities for reusability and how to organize libraries to best facilitate reusability—represent important areas for further work.

Workflow. As federated system models become more elaborate, and more design teams are involved, methods and tools for workflow management and control become critical, not only to automate the linkages between computerized tools and files, but also to deal effectively with version and access control, inconsistencies, and so on. Incorporating workflow models into federated system models may provide an entry point for more effective workflow management, and this represents an opportunity for novel research and development.

Deployment. The move toward MBSE and SysML is in its relatively early stages (analogous to the early days when moving from physical drafting first to 2-D CAD and then to 3-D CAD). Achieving broad deployment (and realizing the associated benefits) will require two kinds of efforts. First, education and training are needed

Peak et al. *continued*

to create the necessary pool of human resources. Leveraging the experiences of projects like this is one way to enhance education and training (e.g., by developing effective teaching materials based on these examples). Second, it is important to continue productizing the results of projects like this (e.g., as is being done by InterCAX2), including SysML-based interfaces to specific tools (e.g., the interfaces to NX and ModelCenter depicted in figure 2).

Summary

This project demonstrates the ability to integrate SysML with a broad range of conventional models for design and analysis. These results indicate that this combined technology does indeed hold great promise to enable model-based systems engineering (MBSE) for large-scale, multidisciplinary, complex system-design projects. This must be augmented by new methods for organizing and managing models, by enhanced interoperability methods, and by a broad range of new tool interfaces. The work summarized here and the opportunities identified for further work represent an important step towards achieving true MBSE by utilizing a SysML-based approach for model interoperability. ①

A Modeling Approach to Document Production

Steven Jenkins, steven.jenkins@incose.org

Model-based systems engineering is often contrasted with document-based systems engineering, but the use of models does not eliminate the need for documentation or negate its utility. There is and will continue to be a systems-engineering role for narrative prose that explains the analyses, trades, and judgments that validate a system design. Contract law, moreover, attaches special significance to writings; contract statements of work will include natural language for the foreseeable future. Rather than view them as opposing styles, JPL has undertaken to integrate model and document orientations. The research described in this article was carried out at the Jet Propulsion Laboratory of the California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Documents as Modeled Entities

The first step in integrating system modeling and document production is to develop a model structure for documents. If we have such a structure, then the creation of a document can be seen as a transformation that converts a system model into a document model. Fortunately, major industries that produce voluminous technical documentation (e.g., information technology, aerospace, automotive, law) have invested for more than thirty years in the development of formal standards for document structure. The widely-adopted Extensible Markup Language (XML) (Worldwide Web Consortium 2006), for example, arose from an earlier Standard Generalized Markup Language (SGML) (ISO 1986) and yet earlier Generalized Markup Language (GML), originally developed at IBM for legal publishing.

Of course, there can be no “standard document” that suffices for all purposes. Instead, modern markup languages provide a set of elements and composition rules by which a particular class of documents can be defined. XML, for example, provides the ability to develop arbitrary element

schema, by which we may declare that there is an element called *article*, that an article may contain *sections*, sections may contain *paragraphs*, and so on. The XML standard then allows us to mark up an actual document with these elements, and to validate that the marked-up document complies with the rules in the schema. If we had mistakenly included, for example, a section within a paragraph, an XML validation utility can tell us the document is not a valid instance according to our schema. XML transformation tools can then convert the XML into HTML, PDF, or a variety of other formats using rule-based transformations.

One essential feature of the “markup” approach to documentation is that nearly all aspects of the visual appearance of the document are specified in external style sheets. This frees authors from the burden of complying with institutional style requirements, and allows the same content to be reused in specialized formats (e.g., for the visually impaired). Let’s look at a specific XML standard for documents.

The DocBook Standard

The Organization for the Advancement of Structured Information Standards (OASIS) is a not-for-profit consortium that “drives the development, convergence and adoption of open standards for the global information society.” One such standard is DocBook (OASIS 2006), a system for writing structured documents using SGML or XML. DocBook includes two key components: a set of document type definitions or XML schema that define the structure of a legal DocBook instance, and a set of Extensible Stylesheet Language (XSL) Transformations (Worldwide Web Consortium 1999) to process DocBook instances, including conversion to HTML and PDF. Both components are extensible: the user may create a custom document type definition based on DocBook, and extend (or create) style sheets to convert to some unique presentation format. Because DocBook is based entirely on widely used XML and related open standards, capable free software tools are available for the entire processing chain, as well as commercial offerings for specialized requirements. DocBook is a mature standard; version 5.0 was released in 2008. DocBook has been used most prominently in the information-

technology industry. A number of computer vendors and large software projects include DocBook documentation.

There are alternatives to DocBook. One is the Darwin Information Typing Architecture (DITA) (OASIS 2009). We have not experimented with DITA, but its emphasis on separation of content from context and its support for content reuse may be applicable to the general problem of machine-integrated documentation. In summary, mature standards and supporting software exist for constructing, analyzing, validating, and rendering large, complex documents of the types typically produced in a systems engineering process. The following sections address the use of one such standard (DocBook) and its associated software for generating engineering documents from system models.

Incorporating Document Models into System Models

Suppose we have a system model containing a tree of *components*, each of which represents an item in the product breakdown structure. A System Description Document may describe some subset of these components, so we need to extend the ontology (or class structure) of our system model to include the concept of *document* and the association *documents* between a component and its associated documents. We then define the concept of *document element* and the association *aggregates* that relates one document element to another. Any given document instance will typically form a strict tree of document elements, but a more general association allows a

single subtree to be aggregated within more than one document (see figure 1).

Using these concepts we can construct a tree structure representing the nested content structure of a System Description Document and associate it with a specific system component. In the simplest possible case, we would populate each document element with document fragments containing literal DocBook content, and then develop a software engine that traverses the tree of document elements and integrates this content into an actual DocBook instance by, for example, nesting lower-level sections within higher-level sections (which is how DocBook indicates subsectioning). While this approach can indeed construct a document, it is no improvement over direct editing with a word processor.

A substantial feature increase can be gained by adding semantics to document elements, so that the traversing software engine can execute instructions that generate DocBook content as it executes. In particular, giving the engine the ability to execute model queries directly means that the document content itself can be generated directly from model content.

Consider a simple example. Using our product-breakdown-structure model from above, suppose that each component includes three properties: an identifier, a name, and a description. Name and identifier are simple strings, and description is a string consisting of one or more DocBook paragraph elements describing the component. And of course, each component also contains an array of references to its child components. We can then define a recursive “component description” function (in pseudo-code) as follows:

```
function component_description(comp) {
    section_start(comp.identifier, comp.name)
    paragraphs(comp.description)
    for child in comp.children {
        component_description(child)
    }
    section_end
}
```

This function creates a tree of DocBook sections whose structure matches some branch of the product breakdown structure, each section describing a component. This document fragment might be included under a top-level section titled “Physical Architecture.”

If we give our software engine the ability to invoke this and other functions, we have a very general capability to generate documents whose complexity and content are determined entirely by the complexity and content of our system models. To be complete, we define three properties for a document element: *entry code* to be executed by the engine on encountering this document element, marked-up DocBook *text* to be inserted literally, and *exit code* to be executed after processing all children of the document element.

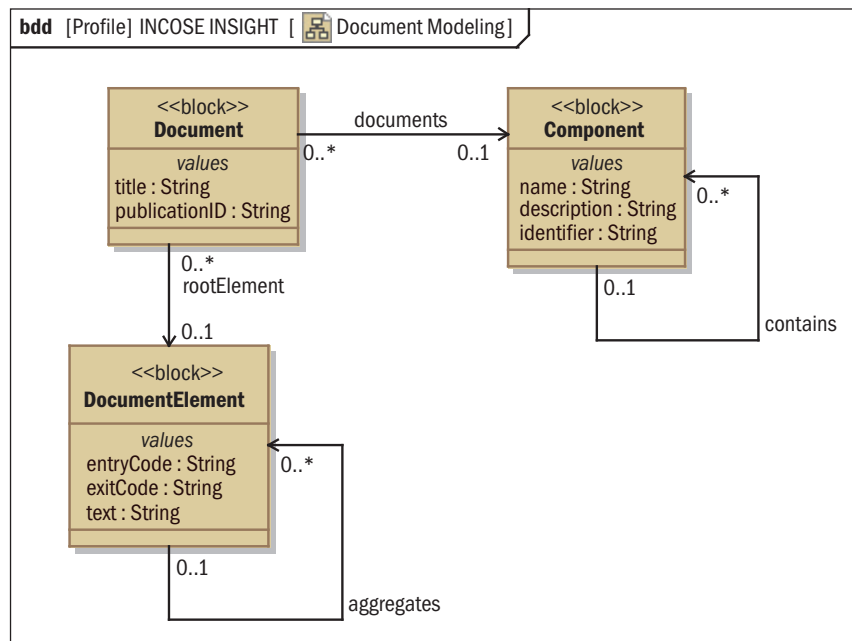


Figure 1. Document model concepts

One obvious benefit to this approach is that a properly constructed document model cannot be inconsistent with the system model. If the system model changes, the generated document will change accordingly. There is no separate update step for documentation. (Of course, the narrative text in the system model should be updated if necessary, but that's simply sound practice.)

With a little thought, we can come up with generators for document fragments commonly appropriate for other model elements and their relationships: interfaces, functions, requirements, risks, processes, work elements, and others. We can then conceive of a document architecture with multiple related document types, each defined by a unique composition of reusable elements. In addition to simplifying the generation of document products, the approach also enhances readability by employing common conventions for common purposes.

Analysis Model Elements

Adding a description property to individual model elements eases the construction of documents that explain *what* a design is. In order to better describe *why* a design is what it is, we introduce the concept of analysis. An analysis is simply a narrative that explains some aspect of a design, and has explicit relationships to the model elements involved in that aspect. The physical decomposition of a system into subsystems, for example, could be described in an analysis and linked directly to the system and its subsystems. A System Design Document generator might then query the model database for any analysis (or perhaps a specialized physical decomposition analysis subclass) related to that system, and insert the resulting narrative into the document at the appropriate point. Any arbitrary set of analysis types (e.g., trade study, risk assessment) can be distinguished through subclassing.

Designing Document Families

The wide availability of personal computers and desktop publishing has, regrettably, made it easier to simply “start typing,” often at the expense of a thoughtful analysis of the concepts to be expressed and the best ways to express them. Hallmarks of professional publishing, such as the disciplined use of metadata, design for readability, cross-referencing, and proper mathematical typesetting, are too easily neglected in this manner of working.

The document-model approach offers a division of labor that exploits the strengths of two distinct types of contributors. The systems engineer or design engineer is freed to concentrate on the correctness, consistency, and clarity of the system model, the only requirement being to follow standard markup conventions in any narrative content in the system model. A context-sensitive editor (e.g., a what-you-see-is-what-you-get applet that understands the DocBook content model)

can enforce these conventions in a natural, non-intrusive way. Correspondingly, the document designer can create well-thought-out document structures and express them in rigorous forms that permit software applications to create compliant instances with essentially no human effort. In addition, he or she can create or apply institutional stylesheets that reduce struggling with visual style.

Other Benefits

Because the document generator creates content that corresponds directly to model elements, it can easily insert index references to those elements. One document we generated in our prototyping contains over 7,700 separate index references, which considerably increases the value of the document as a design specification. It is also straightforward to generate tables showing associations of interest: requirement to component, requirement to requirement, component to function, etc. The document mentioned above, for example, contains more than 150 pages of software-generated tables representing 47 different pairwise data associations. Generation of glossaries is similarly straightforward: DocBook specifies a simple mechanism for tagging glossary terms. Processing software can look up definitions in one of several repositories, and construct a glossary containing definitions for those terms (and only those terms) actually used in a document. Similar considerations apply for lists of applicable documents and bibliographies.

Implementation Details

There are three main parts to our implementation: the model database back end that handles queries against the system and document models, the DocBook generation engine, and the DocBook processing tool chain that produces HTML, PDF, and other presentations. Our prototype applications use one of several Web Ontology Language (OWL) repositories as a backend (Worldwide Web Consortium 2004). We are investigating SysML tools as back end repositories — one alternative is to build on a specific tool's API, another is bulk export and conversion to an existing OWL repository. The DocBook generation engine is written in Ruby, primarily due to the author's preference. Scripting languages in general provide convenient tools for constructing domain-specific languages, but there is no reason to prefer any particular implementation language. The processing tool chain for producing HTML is based on the *xsltproc* free software XSL Transformations utility and DocBook HTML style sheets. The PDF tool chain uses a locally written DocBook-to-LaTeX translator (Lamport 1994) and free software from the TeXLive distribution. We will likely replace our translator with *dlatex*, a free *xsltproc*-based software utility, and customized XSL and LaTeX style sheets.

Prototype Results

In the context of larger efforts on model-based systems engineering, we have built a system model consisting of approximately 8,000 OWL statements and defined document models for two different classes of requirements documents and a project-implementation plan. The resulting eight documents total over 640 pages in PDF form, and are generated end-to-end in PDF and HTML form in less than eight minutes on commodity desktop hardware. Processing scales directly to multiprocessor systems. Our efforts in the future will focus on adaptations to SysML (Object Management Group 2008) modeling tools and to developing document families for JPL projects.

References

- ISO (International Organization for Standardization). 1986. *ISO 8879: Information processing; Text and office systems; Standard Generalized Markup Language (SGML)*. Geneva: ISO.
- Lampert, L. 1994. *LaTeX: A document preparation system*. Boston: Addison-Wesley.
- OASIS (Organization for the Advancement of Structured Information Standards). 2006. The DocBook document type. <http://www.docbook.org/specs/docbook-4.5-spec.html>.
- _____. 2009. Darwin Information Typing Architecture (DITA XML). <http://xml.coverpages.org/dita.html>.
- Worldwide Web Consortium. 1999. XSL Transformations (XSLT) Version 1.0. Ed. J. Clark. <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- _____. 2004. OWL Web ontology language overview. Ed. D. McGuinness and F. van Harmelen. <http://www.w3.org/TR/owl-features>.
- _____. 2006. XML 1.1. 2nd ed. Ed. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. <http://www.w3.org/TR/2006/REC-xml11-20060816>.
- Object Management Group. 2008. *OMG Systems Modeling Language (OMG SysML™)*. <http://www.omg.org/spec/SysML/1.1>.

MBSE for European Space-Systems Development

Harald Eisenmann, harald.eisenmann@incose.org; Juan Miro, juan.mirocarretero@incose.org; and Hans Peter de Koning, hanspeter.dekoning@incose.org

Systems-engineering methods, practices, and tools have been successfully used in European space programs over the last decades. The common reference for the systems-engineering process is the European Cooperation for Space Standardization's E-10 series of standards (see, for example, ECSS 2009b). Although many kinds of models are used to support the development and operation of space systems, the process still very much relies on documents to capture all project information, in particular for the major reviews. In line with INCOSE's *Systems Engineering Vision 2020*, there is a vision that wider and more integrated application of model-based systems engineering will enhance the effectiveness and efficiency of space-system development. In particular MBSE is expected to facilitate and improve early validation and verification, to enhance data consistency, to help develop increasingly demanding and complex systems, and to enable the successful development of systems of systems.

A number of activities have been initiated in the last ten years to help realize the vision. In some areas there has been good progress, so that mature operational solutions are now in place, while in other areas there is still a long way to go. The most significant advances have taken place in two areas: (1) early mission definition and conceptual design in concurrent engineering processes, and (2) system verification for spacecraft control and data handling,

where functional simulation models are used for system-level integration and verification.

Today, most individual domain-specific engineering disciplines — such as power, propulsion, mechanical, thermal, optical, aerothermodynamics, space environment and effects, radio frequency communication and sensing, guidance and navigation, attitude and orbit control, avionics, software, data handling, operations, as well as project management, logistics, and cost estimation — all have their own well-established modeling methods and tools for design, analysis, or simulation. However the tools are often not well connected, exchanging or sharing data is cumbersome, and there is a lot of data duplication with the risk of inconsistencies. In order to achieve the next level of improvements, engineers need an integrated model-based approach at the system level that ensures overall consistency, timely provision, and proper consolidation of all data in the system's lifecycle. This kind of coordination is a typical systems-engineering task.

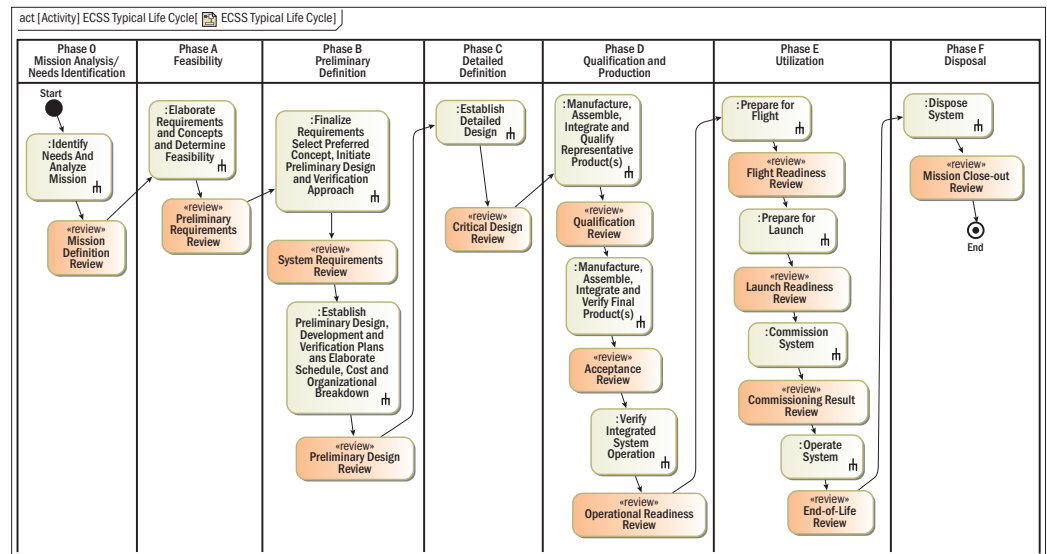


Figure 1. Typical space-system lifecycle from ECSS-M-ST-10C, depicted as a SysML activity diagram

Understanding Systems Engineering for European Space Programmers

Most of the definitions of systems engineering stress multi- or interdisciplinary aspects and full lifecycle considerations. Figure 1 shows the typical European space-system lifecycle (defined in ECSS 2009a), which is quite similar to the NASA or *ISO 15288* lifecycles. In addition, ECSS (2008) defines a recursive customer-supplier model that can be used at any level in the supply chain. One organization can be both supplier to the next-higher level and customer for the next-lower level. One of the critical systems-engineering tasks along the lifecycle (in Phases A and B) is the derivation of requirements for the next-lower system-element level from requirements for the next-higher-level system element. This requires a very robust and reliable system-level design, in particular stable function and product trees. The ECSS documents define *function tree* as the hierarchical decomposition of the system performances into functions and subfunctions, which, when all are fulfilled, complete the overall system mission. ECSS defines *product tree* as the hierarchical structure that depicts the product-oriented breakdown of the project into successive levels of detail, down to the configuration items that are necessary to deliver the required functions.

These decompositions are used to trace the system-level requirements to its intended realization, as well as the intended realization to the derived requirements on the next-lower level. The system-level design must also reflect the constraints identified in the different domain specific analysis activities as well as constraints imposed by integration and verification.

In the different lifecycle phases the following trends can be observed. In Phases 0 and A, the process is driven by all kinds of system budgets—such as overall mass, volume, and power consumption—and performance measures. The European Space Agency has made a particular effort to standardize the key parameters and a basic data model for system decomposition. This model with parameters is implemented in the tools in the Concurrent Design Facility at ESA/ESTEC (<http://www.esa.int/CDF>); this model is increasingly being adopted by similar facilities of other organizations across Europe to ease exchange and collaboration (see ECSS forthcoming 3).

In Phase B the system design is elaborated by using different domain-specific engineering tools. The most mature tool support exists for electrical and mechanical aspects of the system design. In these domains, powerful analysis tools support system-level design, and data-exchange interfaces between design and analysis tools exist. For functional and operational aspects, no dedicated tools are currently available. Requirements-management tools and classical product-data-management (PDM) tools (for project-configuration control) are well established and frequently used.

To better support systems-engineering tasks, several things are needed. First, there is a need for improved modeling of functional aspects, e.g., overall function identification and decomposition, decomposition into operational modes for

different system levels, and reference-timeline definition. There is a lack of support for functional system simulation. Such simulation is needed for improved design consolidation and validation, and operational design definition. It would also pave the road toward functional system simulation in Phases C and D.

Second, comprehensive system modeling will always require multiple tools. There will never be a single tool capable of addressing all needs. Therefore interconnection and integration of the tools is very important. Open interfaces for data exchange and for application-programming are essential. However, experience shows that many COTS tools have significant restrictions with respect to tool integration.

Third, even where powerful analysis tools exist—e.g., for thermal, mechanical, attitude, and orbit control—the effort to create the models and share common data between them is quite large and often involves a lot of manual work. Here also improved interfaces are needed.

Phase B is concluded with the preliminary design review (PDR), which releases the specifications for the next-lower-tier subcontractors. During the subcontractor work in Phase C, the activities on the system level can be summarized as follows. First, the preliminary design baseline is the subject of a refinement during the subcontractor work. Refinement in particular of the interfaces (not only mechanical, electrical, or information but also operational) is performed. This requires continuous analysis of and adaptation to new details and change requests. An integrated, comprehensive system model representation would ease the maintenance of the design baseline.

Second, a characteristic of European space programs is that the suppliers are selected during the program based on the specifications released with the PDR. In the course of the design refinement, it is crucial to set up efficient interfaces to the subcontractors, in order to allow data exchange beyond documents. Although in some cases exchange through spreadsheets already turns out to be a real improvement over simple documents, it would significantly ease the process to have lean, efficient, and commonly agreed-upon data-exchange interfaces.

Third, in parallel to the consolidation of the design baseline, the system integration and verification facility is prepared. Here a specific challenge is to set up, verify, and validate the facility without yet having a system element serving as a mock-up. A simulator can provide a virtual system that is functionally representative and can be used as the mock-up. The preparation of the functional system simulator can effectively start with the PDR, when the overall system design is consolidated. The requirements for the functional system simulators can then be derived, taking into account the verification approach. A comprehensive system model would ease the development and configuration of such simulators.

In Phase D, the space-system elements developed by the subcontractors are integrated and verified. A key element here is a model-based representation of the

functional aspects of the system in a functional system simulator. Many different configurations of the functional system simulation are required, for example:

- Software-in-the-loop (SITL, SWIL, or SIL) and hardware-in-the-loop (HITL, HWIL, or HIL) for progressive integration and verification of the software down to the target hardware and environment
- Open-loop and closed-loop configurations for the attitude- and orbit-control loop
- Complete software simulation and integration of real equipments in the simulation loop
- Hard-real-time configurations for HITL and significantly faster than real-time configurations for SITL.

The work comprises integration testing, verification of software elements, and also validation of onboard control procedures and ground-station operation-control procedures. Although this practice is now state-of-the-art and applied in all projects, the development of this kind of simulator and its integration in the test bench or electrical ground-support equipment requires a significant engineering effort, partly because the reuse of design simulation is seldom attempted and a link of this simulator to the design baseline is not implemented.

In late Phase D, during mission-operations preparation, and in Phase E, the mission-operations team uses the functional system simulator—or an evolved version of it—to develop and validate control procedures or to investigate and resolve anomalies.

Classification of Models in Use

In ECSS (2004), a *model* is defined as a “physical or abstract representation of relevant aspects of an item or process that is put forward as a basis for calculations, predictions or further assessment.” Historically the term *model* is also used to identify particular instances of the product to be delivered, such as the qualification model or the flight model. In the space industry, models of both kinds have always played an essential role in the validation-and-verification approach. An important part of the verification-and-validation plan is the so-called model philosophy, which describes what models will be used when and for what purpose in the life cycle. Evolving computer technology increasingly enables the creation and use of virtual models for many purposes. In discussing MBSE it is essential not to restrict models only to simulation models. The following enumeration provides a taxonomy of the different kinds of models in use:

- Product instance models
- Data (meta)models
- Static or structure models
- Modeling infrastructure models
- Dynamic or behavior models
- Process models

Figure 2 shows how these kinds of models are related to each other.

Product instance models are the classic hardware-oriented models. They comprise first of all the flight model for a one-off or FM1, FM2, etc. and the traditional predecessor, the qualification model. The qualification model is used to

verify the design before the actual manufacturing of the flight model. If the risk is assessed as acceptable, there is today a trend to combine these models into one proto-flight model. While above models always include functional elements of the system, there may be additional models such as a structural thermal model, which is used to test structural and thermal aspects. There is also an increasing trend to rely on an electrical functional model that combines elements of the qualification model and the proto-flight model with a virtual element, the functional system simulator.

Static or structure models capture the evolving specification and definition of the system along the lifecycle. Basically these models are data sets (instance data) in the databases of the supporting tools. With MBSE these models form major artifacts, which need to be exchanged and shared between the project’s participants. They capture information that is common to all disciplines involved in a project. These models include the following information:

- System requirements as specified by the customer and possibly refined by the supplier.
- Next-lower-level (derived) requirements as specified to lower-tier suppliers, based on the design process.
- Functional decomposition of the system of interest, representing a model of the problem space. The focus is on *what* the system shall achieve, not on *how*. The function tree is the core element. Often the interfaces between the different functions are elaborated as well. This representation is also used to capture detailed engineering properties as identified in different analyses.
- Physical architecture and decomposition on how the system of interest will be built and modularized. The focus is on how the design problem will be solved and how complexity is managed. The core information is the product tree that

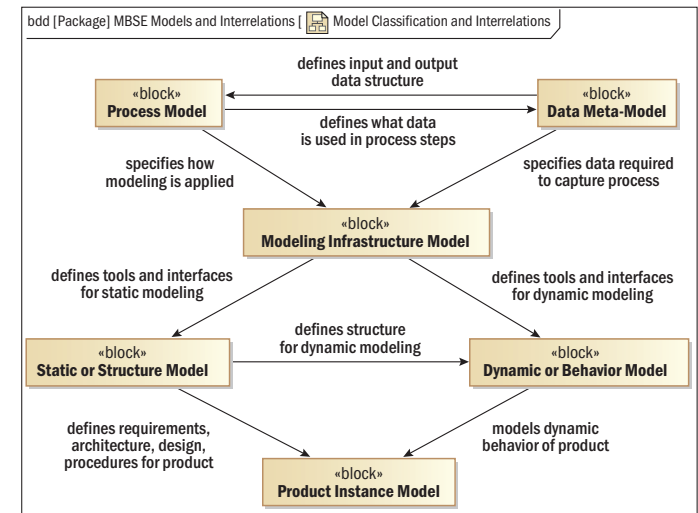


Figure 2. Interrelations between different kinds of model

is also used to set up the organizational breakdown of the supply chain. This needs to be complemented with definition of all interfaces between the different elements and the 3-D spatial configuration. Eventually this decomposition will contain the complete product structure. Associated to its elements are all the actual properties of the envisaged solution.

- Operational decomposition into the modes that are required to command and monitor the system of interest. The operational modes may be attached to any system element. Operational modes may also be hierarchically nested over the different system hierarchies.
- Operational definitions capture how the system will be used, e.g., the sequences in which particular elements are switched on and off. This is closely related to the operational decomposition.
- Assembly, integration, and verification definitions capture the information that is required for the bottom-up assembly, integration, and verification of the system. This information is closely aligned with, and must be consistent with, the functional and physical decompositions. It is a basic input for the project logistics and planning, in particular of all tests.

Dynamic or behavior models are virtual models that represent expected or actual behavior of the system of interest, or aspects of that system. They are executable through use of an appropriate computer tool. They are always idealizations of reality, with different degrees of fidelity, and usually allow for analysis of behavior over time. These models can be classified into the subcategories of domain-specific analysis models and functional system simulators.

Domain-specific analysis models are independent models that are used by the individual engineering disciplines, in support of the overall systems-engineering process. Typically the models focus on the aspects that are important for a particular discipline. Other aspects are neglected or only captured rudimentarily. For example, for attitude and orbit control, a model is formed around the control loop with all its contributors like controller, sensors, actuators or disturbances, while the 3-D physical geometry is only represented schematically or completely abstracted into the relevant centers of gravity and moments of inertia. In some domains the conversion between a design (definition) model and an analysis model is automated or semi-automated with manual assistance, while in other domains it is still fully manual. This implies a significant effort for the model's creation and maintenance. Often the structure of the analysis models is decomposed along the system functions of the corresponding subsystem. The product structure is often not captured. Only selected properties of the system design are assigned to the functions. The analysis models are used for two purposes: (1) during the design phases they are used to understand the problem, identify constraints, and assist in

finding design solutions, and (2) during the later phases they are used to verify the system that is realized (verification by analysis). Models for design and for verification are not necessarily the same, but the latter may evolve from the former.

Functional system simulators can be considered as a virtual representation of a system that can be used or operated like the “real” system. Telecommands are received and processed, corresponding telemetry is provided. Typically the flight software or at least elements of it are in the loop. The functional system simulator follows the system design on (at least) the top decomposition level: therefore top-level elements (equipments) typically have a direct virtual representation. The equipment models closely reflect their behavior according to the corresponding specifications. This applies in particular to operational and electrical interfaces, which are represented on the protocol level. Besides the virtual representation of the system architecture, the physical behavior of the system as a whole is also represented. Typically, at least the following physical behavior is represented: rigid body dynamics, power, and thermal. Therefore the functional system simulator is by definition a multidisciplinary representation. The configurations of a functional system simulator typically comprise the following:

- Functional validation of the attitude- and orbit-control algorithm
- System-level software integration and verification
- Hybrid configuration (electrical functional model) with hardware and software in the loop
- Operations simulator for the ground segment

Data models and meta models capture the structure of data in computer applications. As mentioned above, the static or structure models are stored as data sets inside the databases of the supporting computer tools. As most tools have proprietary native data structures and at best restricted data import/export interfaces, their integration is very costly. However, for effective integration of the tools one needs a precise definition of *what data* is used or required along the lifecycle. This information can be used to map native tool data structures and develop data-exchange solutions. Having powerful (ideally standardized) data models in place is crucial for selecting or integrating tools effectively. As these data models define the structure of other model data, they are often called metamodels. Typically data models are defined in dedicated languages, such as UML class diagrams (with the Object Constraint Language), *ISO 10303 EXPRESS*, Object Role Modeling (ORM), or XML schema.

Modeling infrastructure models is needed for the deployment of effective model-based systems engineering, which requires the integration of the supporting tools in a *modeling infrastructure*. This infrastructure can be regarded as a kind of “super-tool,” which needs to be configured according to the needs of a specific project. The

infrastructure itself needs to be designed, implemented, operated, and maintained, and should therefore itself be the subject of a systems-engineering approach that extends across different projects and their development teams. Such an approach would maximize the effectiveness of investments—both in infrastructure and training—and promote reuse and knowledge sharing across different projects.

Process models can be used to capture and improve engineering processes and workflows. Proceeding with the integration of tools also enables more comprehensive integration of the engineering processes. We expect that through the use of emerging tools for business-process modeling and execution, MBSE-based processes can be improved as well.

The Way Ahead in Model-based System Engineering for European Space Programs

Based on the successful application of systems-engineering processes for many European space programs, ESA and the major European space-system integrators started ambitious research-and-development activities to enhance systems engineering with a set of supportive models and enable MBSE. Although the attempt was to support the complete systems-engineering process from early analysis through final verification, the focus has been on *modeling and simulation*. Demands for providing more powerful, flexible, and modular functional system simulators mainly caused this. ESA developed (for example) the System Simulation and Verification Facility, Generic Project Testbed, and EuroSim, while Astrium developed the Model-based Design and Verification Environment.

Meanwhile these simulators, their infrastructures (for development and runtime) and the simulator engineering processes are consolidated. However, the lack of reliable system models becomes more and more obvious. *Modeling and simulation* often just meant modeling *for simulation*, as the only purpose of the modeling was the development of the simulator. In general, *static models* that represent system structure and are shared between disciplines have not yet been achieved. This kind of integration remains a major issue. In some areas like mechanical engineering and production (often called PLM), there have been significant advances which integrate the mechanical subset of disciplines, but in other areas like functional systems engineering, the available solutions cannot be considered adequate, and important parts of the process are only supported with typical office tools (like word processors and spreadsheets). The way forward is to continue with, on the one hand, a top-down systems-engineering approach with conceptual, semantic, data models to help realize a true MBSE infrastructure, and on the other hand, a bottom-up software-and-simulator-engineering approach starting from the actually realized infrastructure.

A number of initiatives are under way today to evolve and promote the use of MBSE methods and tools in early project phases and during verification activities. MBSE is the focus of a multidisciplinary research-and-development initiative called

Virtual Spacecraft Design under ESA contract in the frame of its Technology Research Program and General Support Technology Program. It is part of a three-year plan (2008–2010), which is expected to yield tangible results in the near future.

Within *Virtual Spacecraft Design* a conceptual data metamodel for the whole space-system lifecycle is further elaborated. An initial version has been established and is being finalized for publication (see ECSS forthcoming 2). This data metamodel has been validated with a number of industrial scenarios, and is also already partially used in actual space projects. An overall comprehensive validation is currently under preparation. It also includes the quantities, units, dimensions, and values model defined in SysML v1.2. Another task is the implementation of a space-system data repository called the Space System Reference Database that provides integrated data-management services. The repository complies with the conceptual data metamodel. It contains a systems-engineering data hub that also acts as a central store for data that is common to all engineering disciplines, and for results consolidated at system level. Its functionality comprises persistent data storage and retrieval, version-and-configuration control (including branching and merging of design options), data consistency and completeness checking, support for automated model transformation, and report generation. Furthermore, a comprehensive set of tools called the Space System Design Editors was developed as well as the Space System Visualisation Tool. Figure 3 shows an architectural overview of the Virtual Spacecraft Engineering Environment of the Virtual Spacecraft Design project. The software-engineering approach for the project itself is highly model-driven.

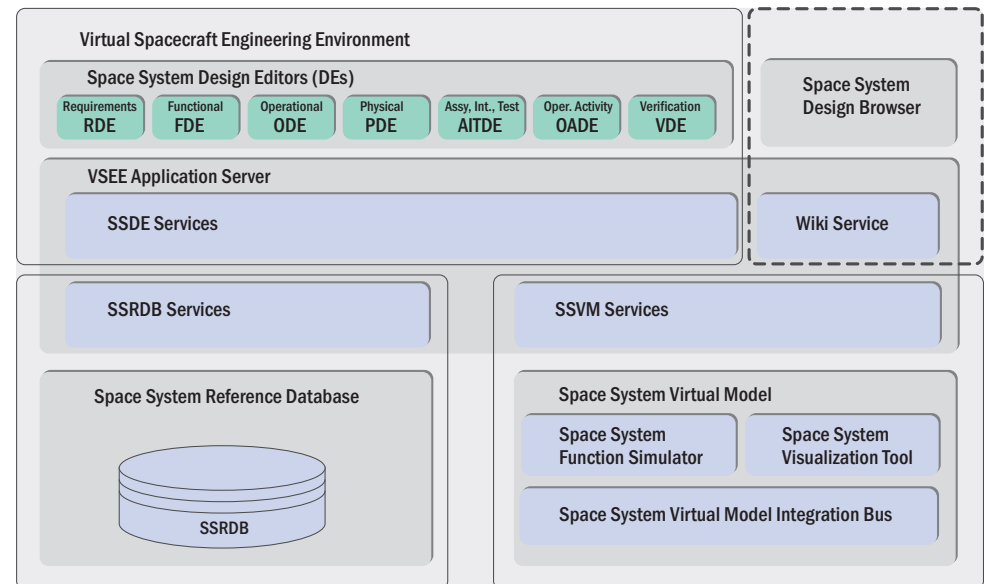


Figure 3. Overview of the European Virtual Spacecraft Engineering Environment (currently being developed as part of the Virtual Spacecraft Design project)

SysML vs. SysML Tools vs. MBSE

While SysML is sometimes considered as equivalent to MBSE, our view is that SysML is one of many means that can be used for MBSE for space systems. SysML is adequate to represent the overall high-level design of the system to be developed. These high-level models are appropriate in the early lifecycle phases (O and A), to elaborate requirements, develop concepts, assess feasibility, and perform trade studies. However, once the project reaches Phase B (preliminary design), the models require so much detail that they are best handled by engineers using their discipline-specific modeling tools. We propose that engineers maintain a high-level model in SysML that reflects the common systems-engineering data during the whole lifecycle, such as system budgets, critical design parameters, and the main decomposition trees together with the interface definitions. We do not consider it useful to implement in SysML the complete space-system design to the lowest level of detail, since the sheer amount of data would render it unusable to the systems engineer. Specific-discipline engineers would typically be unfamiliar with SysML and be more efficient with their discipline-specific representations. The ongoing INCOSE MBSE Challenge for Space Systems is seeking to apply SysML to the representative textbook example called FireSat (Larson and Wertz 1999). The results will provide useful hints on the best practice for using SysML for space-systems engineering.

Today we see the primary application of SysML in early space-system specification and trade-offs, by using its standardized graphical notation for high-level system design, directly fed from the space-system data repository. Research into the feasibility and effectiveness of this approach is currently in progress, among others in the Virtual Spacecraft Design project mentioned above. SysML can fill the gap between the established requirements-management tools and the discipline-specific tools for modeling and simulation. Such usage appears attractive, in particular the SysML graphical notation, known as the “concrete syntax.” However, the current implementation of SysML in existing tools raises doubts with respect to the feasibility for application in Phase B, C, and D. The underlying metamodel of SysML (its abstract syntax) is very rich but complicated and carries a lot of weight from its UML and Meta Object Facility (MOF) heritage. Mapping other data repositories to and from SysML data repositories is not straightforward. In this respect it is interesting to note and investigate the possibility for a kind of “light-SysML” approach based on the much simpler Ecore metamodel established in the open-source Eclipse project.

Most current SysML tools are adaptations of existing UML computer-aided software-engineering tools, which have advantages in terms of the quality of the user interface and the maturity of graphics manipulation, but also disadvantages because the software-engineering background and software concepts keep shining

through. For engineers that do not have experience with UML, the learning curve can be quite steep, and a number of constructs may not be intuitive. This is a potential barrier to the uptake of SysML by systems engineers. Another important limitation concerns data exchange. A systems-engineering tool needs to connect with many other different tools. For this, efficient and high-quality data exchange is necessary. Currently the reliability of data interchange based on SysML or XMI is very limited, and the exchange of diagram layouts is not yet supported at all. Also, incremental changes to data are very hard to achieve through XMI. These issues are well known and being worked on by the UML/SysML community, but it will take quite some time before robust industrial solutions are available. This is a critical success factor.

SysML tools (like all UML-based tools) have the built-in capability—called profiling—to customize and extend the standard language and notation with user-defined additional concepts. On one hand it is very powerful and flexible to have this kind of low-cost customization capability, but on the other hand, if profiling is used too easily or heavily, it can lead to many cumbersome usability aspects, in particular for data exchange and use by different organizations. Nevertheless, SysML seems to be the best candidate for capturing and maintaining a system synthesis representation that can be maintained throughout the system’s lifecycle and provides a standardized graphical notation as well as a mature editing interface. The recently updated UPDM specification, which promotes SysML in many of its diagrams for system-of-systems development, reinforces SysML’s position.

Enabling Technologies

One of the key elements for pursuing MBSE is the availability of a conceptual data metamodel that captures the semantics of all data relevant to systems engineering in a certain application domain, in our case space-systems engineering. The model needs to be defined in a formal, computer-readable way, for two reasons. First, the model is sufficiently large that without formalism and data-modeling tools it is almost impossible to make it consistent and correct, and to evolve it in a controlled way over time. Second, it should be possible to use the model in a model-driven-architecture (MDA) framework (see <http://www.omg.org/mda>).

In our experience the MDA approach is very efficient and robust for the development of adaptations to MBSE tools, such as user interfaces and data-exchange adapters. Very large parts of the required software can be generated automatically, and automated consistency checking of implementations also becomes possible. We would identify the following use cases for the formal conceptual data metamodel:

- Support to the overall definition of an MBSE methodology for a specific application domain
- Design-editor development and customization, e.g., profile definition for a SysML tool

- Generation of data-exchange interfaces and adapters
- Generation of database schemas in different implementation languages
- Reference model to enable central or distributed data repositories for systems engineering that can act as a “data hub” for the tools of all participating disciplines

For space-systems engineering an ECSS technical memorandum with a conceptual data metamodel (specified in UML class diagrams) has been developed (see ECSS forthcoming 2). This model is currently being used and validated in ESA research-and-development projects.

Besides the conceptual data model, a formalized and generalized representation of *value properties* is also essential to succeed with MBSE. A value property—which is a SysML term—is a property with a simple value, like a design parameter or a physical quantity like length, mass, power, force, volume, speed, or electric current. Today in many tools and databases, such properties are still represented as “string” types or even hidden in larger text fields, and often just as values without any measurement units or physical dimension, which severely impedes correct usage and can be a source of serious errors. An agreed-upon and standardized value-property model will greatly improve this situation. On the basis of earlier work on STEP data-exchange standards, ESA has contributed to the development of a very powerful and comprehensive Quantities, Units, Dimensions, and Values (QUDV) data model in SysML version 1.2, annex C (see <http://www.omg.sysml.org>). QUDV is based on, and is fully compatible with, the emerging *ISO/IEC 80000* standard on quantities and units. The same model is adopted in an ECSS document (forthcoming 2).

With the emerging technologies for advanced graphical user interface and editors (such as using the open-source Eclipse modeling framework), alternatives to classic proprietary COTS system-modeling tools become feasible for some applications. Instead of using the conceptual data metamodel to customize or adapt a COTS tool for integration into a model-based systems-engineering infrastructure, a dedicated tool for domain-specific application can be generated for the most part and then manually completed at acceptable cost. Early prototypes have yielded very promising results, and this work will be continued. In the validation activities in support of the new ECSS document (forthcoming 2) Eclipse editors have been automatically generated—implementing the SysML graphical notation. Another example is from functional simulator development where classic tools from SysML and UML have been replaced with editors based on Eclipse Modeling Framework / Graphical Modeling Framework, derived from a conceptual data metamodel for simulator engineering.

Conclusions

A fully operational MBSE process with a corresponding tool set has not yet been realized in space projects today. A number of elements have, however, been implemented successfully, like an integrated design model (IDM) for concurrent engineering in the early lifecycle phases, end-to-end performance simulation for earth-observation instruments, and operations simulation for mission preparation. ESA has started a dedicated set of research-and-development initiatives to further develop MBSE for space missions. All these initiatives will remain ineffective, though, if they are not accompanied by representative pilot applications, where benefits can be measured in the context of a real project and made visible to space-project managers. For this purpose, ESA and the European space-system integrators will need to sponsor shadow application of MBSE elements in ongoing projects. Technology demonstration projects should also be used to demonstrate advanced MBSE development methods and tools in addition to the more traditional focus on innovating the space-system products directly.

Adoption of MBSE methods and tools requires initial investments in time and money by project teams before actual benefits can be obtained. This is why MBSE will only be adopted if reasonably substantiated quantitative evidence of such benefits can be provided. This is a very difficult endeavor, but examples from outside the space sector where such methods have been adopted can be a great help. It is also important to realize that smart technical ideas and approaches are not enough, but need to be supported by mature tools and documented in standards, handbooks, and tutorials, and finally adopted in internal company procedures. Dissemination in seminars, training courses, and technical publications is also essential. Last but not least, systems engineers and project managers need to embrace the ideas and incorporate them into their daily practice. As always, incremental evolution rather than revolution seems the way to go. 🍎

References

- ECSS (European Cooperation for Space Standardization). 2008. *ECSS-S-ST-00C: ECSS system; Description, implementation and general requirements*. This and the following ECSS documents are available at <http://www.ecss.nl>.
- _____. 2004. *ECSS-P-001B: Glossary of terms*.
- _____. 2009a. *ECSS-M-ST-10C Rev. 1: Space project management; Project planning and implementation*.
- _____. 2009b. *ECSS-E-ST-10C: Space engineering; System engineering general requirements*.
- _____. Forthcoming 1. *ECSS-E-TM-10-21A: Space engineering; System modelling and simulation*.
- _____. Forthcoming 2. *ECSS-E-TM-10-23A: Space engineering; Engineering database*. (Note: This establishes a conceptual data metamodel for space-system lifecycle-data repositories.)
- _____. Forthcoming 3. *ECSS-E-TM-10-25A: Space engineering; Engineering design model data exchange*. (Note: Scope is the exchange of conceptual design data between concurrent design facilities.)
- Larson, W. J., and J. R. Wertz. 1999. *Space mission analysis and design*. 3rd ed. El Segundo, CA: Microcosm.

SysML is the Point of Departure for MBSE, Not the Destination

Anatoly Levenchuk, anatoly.levenchuk@incose.org

Engineers today use the word *model* in such a vague way that it means almost the same thing as *description*. When we write *model-based systems engineering*, it could also be read as *description-based systems engineering*. But isn't any engineering description-based, with text, drawings, and formula descriptions? Why did we coin a new term to describe ordinary practice, and stress that this is the future of systems engineering?

I guess the new term would appear to reflect new types of models. However, MBSE is not about any descriptions usually associated with "models." MBSE is not about simulations and emulations. MBSE is not about using Modelica, a non-proprietary, object-oriented, equation-based language to conveniently model complex physical systems containing mechanical, electrical, electronic, hydraulic, thermal, control, electric power or process-oriented subcomponents, nor finite element analysis models. Neither is MBSE about using complex multi-physics models in simulations of cyberphysics systems, where we simultaneously compute several different nature models (Sztipanovits 2008). Nor is MBSE about Reference Process Models that we can see in *ISO 15288* and other systems-engineering standards.

Model-based systems engineering is about generative models. I am using the term *generative* here in the same sense as Noam Chomsky's "generative grammar"; this should not be confused with generative models from statistics. According to Chomsky, "when we speak of a grammar as generating a sentence with a certain structural description, we mean simply that the grammar assigns this structural description to the sentence" (Chomsky 1965: 9). Models in MBSE generate system descriptions in the same sense that generative grammar generates sentences. Generative production systems (Fox 2009) use the same notion of generative grammars—to shape grammars used in a generative design domain. I suppose that such an approach is valid not only to shape languages of mechanical engineering but to all languages that systems engineers experience in interdisciplinary projects.

In essence, model-based systems engineering, as described by

the Object Management Group, is about the use of metamodel layers (description-of-descriptions layers, language layers) to describe systems. Every layer of the model stack (model M0, metamodel M1, meta-meta-model M2, etc.) can be detailed and/or transformed to "generate" another description that addresses interest of one or more stakeholders, and eventually these comprehensive descriptions will be sufficient to the realization, integration, verification and validation of a system. A key characteristic of model-based systems engineering is the support of multiple viewpoints, that is, multiple methods of modeling to provide multiple views, or in other words, multiple groups of description that address different interests of appropriate stakeholders. Fifteen years ago it was not common to accent this multiple-view approach. Modeling was usually mono-modeling that provided one type of view every time, and providing links between relations and objects in different views was a big problem to engineers. The OMG's Unified Modeling Language was a breakthrough that brought the paradigm of multiple viewpoints or views to mainstream software engineering. UML describes five types of diagram to enable the user to capture different aspects of software systems. Moreover, UML is expandable in a formal manner, and after a ten-year lag, systems engineering now has a means to provide these multiple viewpoints (*ISO 42010*) description in the form of SysML.

UML, along with the OMG's (2006) Meta Object Facility (which provides interoperability for all UML-based models), is a core language of the OMG (2003) model-driven architecture (MDA). The MDA community is rethinking their scope and approach from a software-centric to a system-centric view (Cloutier 2006; Dickerson 2007). Key features of MDA include multiple layers of abstraction (metamodeling) and multiple viewpoints with definite correspondence rules, provided by its metamodels. The main disadvantage of MDA is that it is bounded by UML. Why is UML a disadvantage? Because in a multi-disciplinary project we cannot expect that all specialists speak UML or SysML, even if we extend it with domain-specific stereotypes.

Along those lines, the software-engineering community has started down another branch of the modeling movement related to the domain-specific-language (DSL) approach to modeling. Domain-specific languages provide a viewpoint for expert's view to particular domain. The domain-specific-languages approach combines domain-specific metamodels and notation. This is

.....
Systems engineers are just now using SysML to create consistent models instead of domain-specific languages — much like programmers a few years ago used Java only, not Java + DSLs.

different from the MDA approach, which prescribes MOF/UML-based metamodels and notations. Thus programmers should provide separate integrated development environments for each domain-specific language. Then experts can use such a suite of specialized, integrated development environments to model their systems. A corollary is that all contemporary CAD suites can be regarded as integrated development environment suites for “programming” of engineering domain models in domain-specific languages such as piping-and-instrumentation-diagram languages for modeling of hydraulic systems or 3-D graphical “language” for the modeling of spatial shapes. If we want to capture a facility model of a process plant we have a difficult choice between MDA/SysML modeling of hydraulic systems or modeling it with piping-and-instrumentation-diagram notations. Domain-specific languages for contemporary CADs win hands down by capturing the specific concepts and notation used in specific domains. Yet we need to combine all these incompatible domain-specific languages into one coherent facility model.

There exist two options to be consistent among the zoo of different domain specific languages: (1) ontology-based mapping of different DSL metamodels and, therefore, linking domain-specific models in a common data-centric model repository; or (2) using a language workbench.

The ontology-based mapping that is now used by all major CAD vendors uses different upper ontologies and domain taxonomies—for example, *ISO 15926* (ISO 2004a) for process industries, *ISO 18629* (ISO 2004b) for a process-specification language, or *ISO/PAS 16739* (ISO 2005) for building information modeling in construction industries. This is a viable option to cope with legacy systems that support “good old” domain-specific languages for engineering modeling. It is a relatively fresh movement (started from 1994 work on Shell’s downstream data model: see West 2009), originating from the data-modeling branch of software development. Now adopters of this approach are moving toward using the readily available semantic Web tools (AIFB 2009) to perform data modeling and data integration, starting with experiments in logical reasoning for “executing” of ontology-based models. Today most ontologists (or data modelers) are former programmers, software analysts, and database architects; in other words, this field is de facto now part of software engineering, not systems engineering.

Language workbenches are newly emerging integrated development environments that are especially devoted to multiple domain-specific languages (Fowler 2005). Developing a language-independent interpreter/compiler and language-independent graphical editor is a challenging and complex task. But the prospects are attractive: every expert can get her own customized DSL, and all of these domain-specific languages that addresses multiple stakeholders’ interests will work in concert with one another. Moreover, these distinct descriptions that provide separations of concerns can be used for different purposes and in different

ways: they can be validated for consistency, transformed to output language suited for manufacturing tools, or transformed to executable simulation models.

This approach may prove superior to that of model-driven architecture with UML or SysML), because it provides freedom of language choices and still retains a coherent model. Engineering-domain experts will get integrated development environments (editors, interpreters, repositories) for languages that they are accustomed to using (both the metamodel part of these languages and notational part), not stereotypes of UML with predefined UML semantics in predefined UML syntax. Language workbenches permit programmers to easily produce domain-specific integrated development environments while still providing a common repository for different domain-specific models that are produced by different experts in the different languages. You may think about language workbenches as “CAD for CAD” that preserve the freedom of an arbitrary domain language (metamodel + notation) choice. A language workbench is unlike UML models, which permit a choice from UML-defined metamodels only and from UML stereotype notation only. Programmers will add languages to the common language environment (viewpoints), and domain experts will develop model (views) with these languages. Everyone will be doing what they do best.

Today, language workbenches are limited to the software community. Language workbenches first appeared in the form of integrated development environments for compiling multiple domain specific languages to Java or C#, not to “compile” multiple engineering-domain-specific languages to a facility model in contemporary CAD suites. So far, systems engineers have ignored the domain-specific language and language-workbench approaches. Systems engineers are just now using SysML to create consistent models instead of domain-specific languages—much like programmers a few years ago used Java only, not Java + DSLs.

Systems engineers may continue to lag behind the software community in adopting these new modeling approaches from software engineering. However, I believe they should overcome their resistance to change, and begin to experiment with these new technologies. This would result in SysML being merely the point of departure for model-based systems engineering, not the destination point.

References

- AIFB (Institut für Angewandte Informatik und Formale Beschreibungsverfahren). 2009. *Semantic Web*. Wiki for Semantic Web community. Hosted by AIFB at the Karlsruhe Institute of Technology, Karlsruhe, Germany. <http://semanticweb.org> (accessed 11 Nov. 2009).
- Chomsky, N. 1965. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.
- Cloutier, R. 2006. MDA for systems engineering: Why should we care? Paper presented at the Telelogic Americas User Group Conference. Available at <http://www.calimar.com/Papers/Model%20Driven%20Architecture%20of%20SE-Why%20Care.pdf> (accessed 11 Nov. 2009).
- Dickerson, C. E. 2007. Model driven architecture for model based systems engineering. Paper presented at SEDC Integration Workshop. Available at <http://www.lboro.ac.uk/departments/el/sedc/documents/presentations/model-driven-architecture.pdf> (accessed 11 Nov. 2009).

Learning that Fits *Your Life*

Convenient Online and Distance Education that Fits into Your Busy Career



Online Master of Engineering or Certificate in Systems Engineering

Gain skills and training in high demand by employers from a University with more than 40 years of experience providing distance education to working professionals.

Apply or enroll today for flexible and convenient online education in Systems Engineering that meets workforce needs and helps you achieve your career goals.

Colorado State University

Continuing Education

www.learn.colostate.edu/inc • (877) 491-4336

Levenchuk *continued*

Fowler, M. 2005. Language workbenches: The killer-app for domain specific languages? Personal Web site. <http://martinfowler.com/articles/languageWorkbench.html> (accessed 11 Nov. 2009).

Fox, S. 2009. Generative production systems for sustainable product creation. VTT working papers 129. Espoo, Finland: VTT Technical Research Centre of Finland. <http://www.vtt.fi/inf/pdf/workingpapers/2009/W129.pdf> (accessed 11 Nov. 2009).

ISO. 2004a. *ISO 15926-1: Industrial automation systems and integration; Integration of life-cycle data for process plants including oil and gas production facilities. Part 1: Overview and fundamental principles*. Geneva: ISO.

_____. 2004b. *ISO 18629-1: Industrial automation systems and integration; Process specification language. Part 1: Overview and basic principles*. Geneva: ISO.

ISO and IEC. 2008. *ISO/IEC 15288: Systems and software engineering; System life cycle processes*. Geneva: ISO.

ISO and IEEE. 2007. *ISO 42010/IEEE 1471: Recommended practice for the creation, analysis, and maintenance of software-intensive system architectures*. Geneva: ISO.

ISO. 2005. *ISO/PAS 16739: Industry foundation classes, release 2x; Platform specification (IFC2x platform)*. Geneva: ISO.

OMG (Object Management Group). 2003. *OMG Model Driven Architecture. Object Management Group Web site*. <http://www.omg.org/mda/> (accessed 11 Nov. 2009).

_____. 2006. *OMG's MetaObject Facility. Object Management Group Web site*. <http://www.omg.org/mof/> (accessed 11 Nov. 2009).

Sztipanovits, J. 2008. Convergence: Model-based software, systems and control engineering. Paper presented at OOPSLA 2008. Available at <http://www.infoq.com/presentations/Model-Based-Design-Janos-Sztipanovits> (accessed 11 Nov. 2009).

West, M. 2009. Matthew West's publications. Personal Web site. <http://www.matthew-west.org.uk/Publications.html> (accessed 11 Nov. 2009).

22nd Annual

SSTC
Systems & Software
Technology Conference

TECHNOLOGY:



CHANGING THE GAME

26-29 April 2010 • Salt Lake City, Utah

Conference Registration Opens 5 January 2010

Exhibitor Registration Available Now

120+ TECHNICAL PRESENTATIONS

TRAINING AND CERTIFICATION

OPPORTUNITIES AT A REDUCED COST

COLLABORATIVE NETWORKING

TRADE SHOW

SCENIC LOCATION



PLAN NOW TO ATTEND!
WWW.SSTC-ONLINE.ORG

SYSTEMS ENGINEERING TRAINING

PE CERTIFIED COURSES • CUSTOM TAILORED • ON SITE TRAINING

TONEXTM
EMPOWERED BY KNOWLEDGE AND SKILLS

Over 20 years of experience training professionals from small businesses to top Fortune 100 Companies and many government agencies.

Contact us to learn more!

INFO@TONEX.COM

1 - 888 - TO - TONEX

WWW.TONEX.COM



New Additions to the Complex and Enterprise Systems Engineering Series...

Series Editors:
International Council on
Systems Engineering (INCOSE)
Dr. Paul R. Garvey and Brian White
The MITRE Corporation, Bedford, Massachusetts, USA

Cutting-edge techniques you can apply to any workplace or research program

Architecture and Principles of Systems Engineering

C.E. Dickerson
Loughborough University, Leicestershire, UK
D.N. Mavris
Georgia Institute of Technology, Atlanta, USA

- Explains how Model-Driven Architecture (MDA™) and other initiatives will greatly affect the evolution of architecture and SE
- Introduces tools for optimizing model-based architecture and systems engineering practices
- Reinforces understanding with practical case studies

Catalog no. AU7253
ISBN: 978-1-4200-7253-2
November 2009, 6-1/8 x 9-1/4, 496 pp.
Suggested Price: \$89.95 / £54.99

Models and procedures to optimize systems design and performance

Designing Complex Systems Foundations of Design in the Functional Domain

Erik W. Aslaksen
Sinclair Knight Merz, Sydney, Australia

- Presents specific models and procedures for carrying out systems design and optimizing system performance
- Examines the purpose and basic features of design and the development of design methodology
- Defines functional elements and investigates their properties and interactions

Catalog no. AU7533
ISBN: 978-1-4200-8753-6
January 2009, 6-1/8 x 9-1/4, 176 pp.
Suggested Price: \$69.95 / £44.99

Model-Oriented Systems Engineering Science A Unifying Framework for Traditional and Complex Systems

Duane W. Hybertson
MITRE, McLean, Virginia, USA

- Extends existing modeling approaches into an MO that views all science and engineering artifacts as models of systems
- Organizes approaches into a virtual “SE model space”—effectively a container for the accumulating body of SE and SES knowledge in the form of models and patterns

Catalog no. AU7251
ISBN: 978-1-4200-7251-8
June 2009, 7 x 10, 379 pp.
Suggested Price: \$89.95 / £54.99



Order securely online at www.crcpress.com • Free Standard Shipping on All Orders

*Enter promo code **606LA** at checkout to receive your discount.

Offer good through January 31, 2010

Key Issues of Systems Engineering

Presented by the INCOSE Fellows

Introduction by William Mackey, william.mackey@incose.org

When I became chair of the INCOSE Fellows in 2007, I quickly realized that I was meeting with some of the finest minds in the metadiscipline of systems engineering. I began to ask myself and others how we could make our time together in business meetings more stimulating and how we might create the opportunity for all of those experienced systems engineers to share their ideas with regard to the discipline. I asked for a volunteer to collect information regarding the most important issues related to the metadiscipline. For a year, Bill Schoening collected a small database of what we called “The Key Issues of Systems Engineering.” The Fellows identified 35 major issues related to systems engineering with a large number of sub-issues numbering almost 100 issues in all.

The Process

I decided that an experiment was in order, so in 2008 I asked for a vote on which of the issues we as a body should address first. I also asked who might address those issues. The list narrowed to eleven, with Fellows' names attached to each of the eleven issues.

For the 2009 International Workshop in San Francisco, I created ten teams of three Fellows each and requested those Fellows to prepare their views on these defined issues. The discussions at IW09 required a full day and were so in-depth that we were only able to complete the discussion of five of those issues. Everyone was extremely pleased with our experiment, and began to ask how we might share our intellectual day with the rest of the INCOSE membership. My view of the day was that it had been the best sharing of ideas that I had ever experienced in INCOSE including symposia, workshops, chapter meetings, and hundreds of working group sessions that I had attended since becoming an INCOSE member in 1992.

I had assigned a lead speaker for each issue, so I requested the leaders to draft a white paper for each of the key issues we had discussed. Each leader created a draft and got a level of consensus with each of his or her team members. Once the team had agreed

on the content of the white paper, the team sent the white paper to every Fellow for review. Gaining consensus from such a large body is not at all easy, but with a few rules we now have five white papers we will present to you in *INSIGHT*. This issue of *INSIGHT* contains the first white paper, and the remaining key issues will be discussed in future issues. I wish to indicate the following provisos related to these white papers:

- The summaries are the consensus of the Fellows' teams, and not necessarily the Fellows as a whole. The teams solicited comments from the Fellows as a whole and incorporated them where they thought they were valid and pertinent.
- The summaries are not prescriptive; they are simply opinions and starting points for future discussions. They are not position statements, but rather discussion papers to provide insight to stimulate further discussion.
- The summaries do not represent the position of INCOSE nor do they necessarily agree with published INCOSE documents, such as the *Systems Engineering Handbook*. They represent ideas that may or may not be incorporated in INCOSE documents in the future.
- The summaries do not represent direct or indirect criticism of any person or group. Many ideas exist within INCOSE, and only the future will determine the consensus of INCOSE as a whole.

The Key Issues of Systems Engineering

The five issues that were discussed at the Fellows' IW09 meeting and the teams that presented and documented these issues are as follows:

Issue 1. What are the general principles applicable to systems?

Speakers: D. Hitchins (lead), B. Boehm, S. Sheard

Issue 2. What is the “systems approach” and why is it fundamental to systems, systems thinking, systems methodology, systems design, and systems engineering?

Speakers: S. Jackson (lead), D. Hitchins (online), H. Eisner (online)

Issue 3. How can you “prove” that your systems design will solve the customer's problem before you build and prove that design?

Speakers: J. Ring (lead), H. Eisner (online), M. Maier

Issue 4. What is the return on investment (ROI) for using systems engineering?

Speakers: B. Boehm (lead), S. Sheard, M. Maier

Issue 5. What distinguishes complex adaptive systems from other kinds of systems?

Speakers: A. Sage (lead), J. Ring, S. Sheard

What Are the General Principles Applicable to Systems? | Derek Hitchins, profhitchins@incose.org¹

Uncertainty reigns over the question of what might reasonably be considered as a general systems principle, reflecting the confusion between systems-as-a-discipline, systems thinking, systems engineering, project management, engineering, engineering management, operations analysis, defense procurement, etc. Each of these may lay claim to being the source of general systems principles. Identifying the principles of systems and of systems engineering, then, depends upon what you think systems and systems engineering are. For instance, is systems engineering *about* systems, engineering, management, or is systems engineering something else: a unique metadiscipline, perhaps, as viewed by the doyen of systems engineering, Arthur D. Hall III (1989: 3–52)?

Reflecting this identity conflict, candidate principles might include the following:

- *Give the customers what they want.* This is the so-called “service principle,” so specifically neither systems, nor systems engineering: more, perhaps, commerce?
- *Faster, better, cheaper.* This is less a systems-engineering principle, perhaps; more a commercial lean-manufacturing mantra.
- *Brook’s law:* “Adding people to a late project makes it even later.” This is project management rather than systems engineering.
- *Bellman’s principle of optimality.* Concerned with dynamic programming, so possibly with the “best” systems-engineering process?
- *Popper’s principle of falsifiability.* If systems engineering is scientific problem-solving, then solutions should be testable, i.e., open to being proved false.
- *Systems engineering looks upwards and outwards.* This is more about systems-as-a-discipline and systems-engineering philosophy than a general systems principle.
- *Kaizen, the philosophy of continual improvement.* Kaizen is generally viewed as a philosophy, or school of thought, rather than a principle, but this is a possibility.
- *Optimize the whole, not the parts.* A principle for systems design or architecting, so this would be a possible general systems-engineering principle.

1. Early in 2009, a small team of INCOSE Fellows—Barry Boehm, Sarah Sheard, and I—explored this question; this article resulted from our reflections.

- *Form follows function.* Louis Sullivan (1947: 208) famously formulated this well-known principle: “It is the pervading law of all things organic, and inorganic, of all things physical and metaphysical, of all things human and all things superhuman, of all true manifestations of the head, of the heart, of the soul, that the life is recognizable in its expression, that form ever follows function. This is the law.” No confusion there, then! At the time, Sullivan was concerned with the design of skyscrapers—but is it a general systems principle?

In addition to these, I have previously collected several “principles of creativity in systems engineering” (Hitchins 1992: 246–248), associated with the conception and design phases of systems engineering:

<ul style="list-style-type: none"> • Highest level of abstraction • Disciplined anarchy • Breadth before depth • One level at a time 	<ul style="list-style-type: none"> • Decomposition before integration • Functional before physical • Tight functional binding • Loose functional coupling
<ul style="list-style-type: none"> • Functional migrates to physical 	

Some of these so-called principles (practices?) guide abstract problem solving, others guide top-down design elaboration, while still others guide the development of systems architecture and some guide procedure. According to this approach, the concepts of “functional before physical” and “functional migrates to physical” reflect Sullivan’s “form follows function.” But, are they “general systems principles,” or are they confined to only a part of orthodox systems engineering?

What is a principle in this context? According to the *Oxford American Dictionary*, a principle is “a general scientific theory or law that has numerous special applications across a wide field.” If we take this definition, then we are seeking some underlying or extensive, or systemic, characteristic that relates to all systems, to all systems engineering—or, ideally to both. These characteristics might be the way systems form, sustain, operate, adapt, evolve, fade, and die. Such principles are likely to be founded in systems science, the science of wholes and of complexity, which incorporates the physical, natural, and life sciences. General systems principles are hard to find in the literature, however, without some prefix: for example, one finds “operational systems principles,” “control systems principles,” or “distributed systems principles.” Examination of supporting literature shows these terms to refer less to systems in general, and more to operations, control, distribution, and so on. On the other hand, it appears reasonable to suppose that general systems principles will also have relevance to systems engineering.

General Systems Principles

Propositions that are self-evidently true are axioms, and the following appear axiomatic:

First principle of systems. The properties, capabilities, and behavior of a system derive from its parts, from interactions between those parts, and from interactions with other systems.

Corollary to the first principle. Altering the properties, capabilities, or behavior of any of the parts, or any of their interactions, affects other parts, the whole system, and interacting systems.

The first principle, which seems to derive from general systems theory (for which see Bertalanffy 1968), may be axiomatic, but is it helpful? Research into the formation and behavior of systems considers not a single system, but networks of interacting systems. General systems principles emerging from such work should, in principle, prove more useful.

In one approach, known as “systems-lifecycle theory” (Hitchins 2003: 107–121), the following general systems principles are expounded to form an integrated set, which together form a systems-lifecycle map (see figure 1). See Hitchins (2003) for further explanation of how these systems principles apply to the creation, manipulation, demise, and design of systems; I have also explained them in a brief video available at <http://www.hitchins.net/SystemsPrinciples.mov>. Note that the seven systems principles apply *collectively* to any open system or network of such systems.

1. *The principle of system reactions* (after Le Chatelier’s principle). If a set of interacting systems is in equilibrium and either a new system is introduced to the set, or one of the systems or interconnections undergoes some change, then, insofar as they are able, the other systems will rearrange themselves to oppose the change and, in so doing, move to a new point of equilibrium.
2. *The principle of cohesion.* A system’s form is maintained by a balance, static or dynamic, between cohesive and dispersive forces.
3. *The principle of adaptation.* For continued systems cohesion, the mean rate of system adaptation must equal or exceed the mean rate of change of the environment.
4. *The principle of connected variety.* The stability of interacting systems increases with variety, and with the degree of cohesion of that variety within the environment.
5. *The principle of limited variety.* Variety in interacting systems is limited by the available degrees of freedom and minimum degree of differentiation.
6. *The principle of preferred patterns.* The stability of interacting systems increases both with the variety of systems and with their cohesion.
7. *The principle of cyclic progression.* Interconnected systems driven by an external energy source will tend to a cyclic progression in which variety of systems (and variety within systems) is generated, dominance emerges to suppress the variety, the dominant mode decays or collapses, and survivors emerge to regenerate variety.

The principle of cyclic progression and the lifecycle map of figure 1 address the phenomenon of “entropic cycling,” in which order and disorder among complex networks of systems are observed to cycle repeatedly over time. Economic, ecological, and climate systems display this characteristic, as do teams, businesses, and industries, leading me to propose a new law of complexity (Hitchins 2008): “The entropy of open, interacting systems cycles continually at rates and levels determined by available energy.”

How does the map work? The lives of systems move around the map in a clockwise direction. Starting at one o’clock, energy promotes environmental change and variety generation; environmental change promotes adaptation, which also promotes variety generation. Varieties may act and interact with others to form complementary sets and connected variety, promoting a tendency to stability (five o’clock). This in turn leads to preferred patterns, and to systems cohesion, the latter being threatened by dispersive influences, in which generated variety, which has not been taken up as connected variety, may behave destructively — as with pathogens and viruses in organisms and computers, and sociopaths in society.

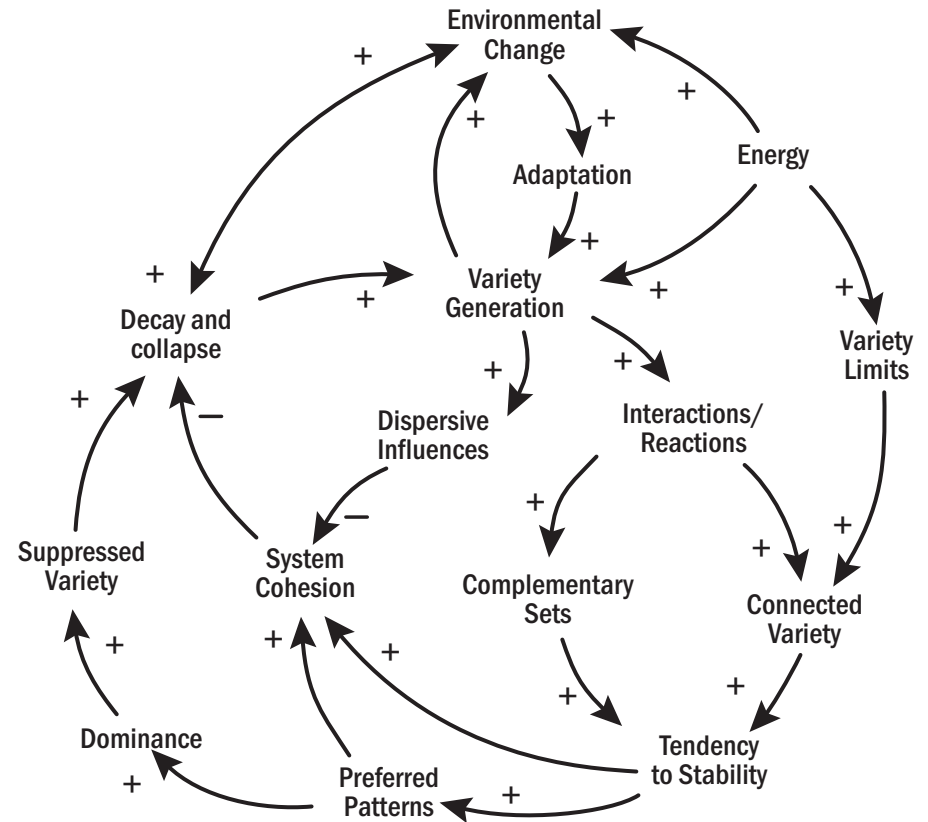


Figure 1. Systems-lifecycle map

Preferred patterns give way to dominance, in which one aspect of the network of systems dominates—commands the bulk of resources, energy, and so on—so weakening the others, leading to suppressed variety; this in turn presages decay and collapse, if system cohesion is inadequate or when environmental change arises, finding the systems with insufficient variety to respond to changing circumstances.

Note the positive feedback loop: decay and collapse, variety generation, dispersive influences, system cohesion, decay and collapse. This positive feedback loop suggests that collapse, when it occurs, may be sudden and catastrophic. We see such situations in the domino collapse of the former Soviet Union, for instance. Collapse is not inevitable, however: restoring variety may, instead, redirect the path of the weakened system back around the connected variety loop again. Such a system is seen to undergo periodic perturbations or upheavals. IBM underwent such an upheaval in the not-too-distant past when it was slow to recognize market environmental change, and the consequent limitations of its computer mainframe range. Some organizations find it beneficial to undergo periodic upheaval (reorganization), to avoid becoming moribund; others absorb new variety by taking over companies. Could this be systems engineering of a kind?

The systems-lifecycle map is not limited to explaining past situations in systems terms: it also offers potential for a different kind of systems engineering. For instance, by preventing dominance and by maintaining and refreshing variety in systems, they may exist indefinitely—without end. Alternatively, encouraging dominance and/or restricting variety leads to moribund systems, poised to collapse. This pattern has been observed in some political systems, and, as (inadvertently?) practiced by some accountants in industry during periods of recession. This “different take” on systems engineering, with its emphasis on variety, dominance, and entropic cycling, offers a new perspective on viruses, and how to deal with them. It may even shed some light on the evolution of organization, of industry, society, and even of life from the primordial soup!

Systems-Engineering Principles

The following are fundamental, guiding principles that have been the foundation of systems engineering, apparently since its inception. In many respects, systems-engineering principles A through D define and characterize systems engineering. It might actually be reasonable to consider them as the four “pillars of systems engineering.”

SE Principle A: The Systems Approach

The systems approach (Jenkins 1969) will be addressed fully elsewhere in this series of articles. Essentially, the systems approach addresses the system of interest

(SOI) in context, as an open¹ system that (1) interacts with and adapts to other systems in its operational environment, (2) contains open, interacting subsystems, and (3) forms part of some wider or greater whole. The systems approach, then, considers an SOI to be *open* and dynamic, and to be comprised of open, dynamic, *interacting subsystems*. It also understands the SOI to exist in an *environment*; to *interact* with, and *adapt* to, other systems in that environment; and to form part of a larger, *wider*, or *containing* system.

SE Principle B: Synthesis

Synthesis brings parts together to act and interact as a unified whole. Parts or subsystems of a system cooperate, coordinate, contribute, and behave synergistically, enabled by their interconnections and interactions. Such patterns or orchestrations of interaction are preserved during design elaboration, development, construction, and integration; otherwise, integration will not reconstitute the original, designed whole. “The ‘essence’ of systems engineering is in choosing (conceiving, designing, selecting) the right parts, bringing them together to interact in the right way, and in orchestrating those interactions to create requisite properties of the whole, such that it performs with optimum² effectiveness in its operational environment, so solving the problem that prompted its creation” (Hitchins 2008: 120).

SE Principle C: Holism

As the contributors to *Wikipedia* explain it, “the properties of a given system (biological, chemical, social, sociotechnical, economic, mental, linguistic, etc.) cannot be determined or explained by its component parts alone. Instead, the system as a whole determines in an important way how the parts behave.”³ The saying attributed to Aristotle puts it more succinctly: “The whole is greater than the sum of its parts; the part is more than a fraction of the whole.” Kast and Rosenzweig (1981: 46) define *holistic* as “emphasizing the functional relation between parts and whole; pertaining to totality, or to the whole. The holistic view is basic to the systems approach.”

Holism pervades our systems thinking and systems-engineering activity: we consider each and every part of a system always as connected, active, in context, not in isolation. Further, we avoid addressing only part of a problem, to avoid exacerbating the whole problem. So, consistent with the first principle of systems, systems engineering addresses the *whole* problem, and creates the *whole* solution. Similarly, systems design and systems engineering seek to optimize the whole

1. An open system is one that exchanges energy, substance, and information with its environment.
 2. *Optimum* in this context means simply “best” or “greatest.” “Best effectiveness” may be cost-constrained, and expressed colloquially as “most bangs per buck,” or “the best value for the money.”
 3. *Wikipedia*, s.v. “holism,” <http://en.wikipedia.org/wiki/Holism> (accessed 20 Oct. 2009).

system, not the parts: it can be shown that optimizing the parts independently of each other may actually de-optimize the whole.

SE Principle D: Organismic Analogy (Organicism)

The organismic analogy (Bertalanffy 1962) compares society and social systems, to the human body, with organic subsystems containing organs. This analogy does not claim that society actually is an organism, but that in some ways it *behaves* as one. One can think of society as having these organic subsystems:

<ul style="list-style-type: none"> • legal, judicial • power, energy, water, sanitation • waste disposal 	<ul style="list-style-type: none"> • education • economic • penal
---	--

Like organisms, such societal systems also display lifecycles (see figure 1). Systems engineering creates organized, purposeful sociotechnical systems, suggesting that systems engineering might generally employ the organic metaphor, rather than the mechanistic metaphor of classic technology engineering. The organic metaphor is consistent with open systems that interact with, and adapt to, other systems in their environment.

William Emerson Ritter coined the term *organicism* in 1919 to denote the concept that (as Wikipedia puts it) “reality is best understood as an organic whole.”⁴ Organicism is close to holism. Organicism stresses the organization, rather than the composition, of organisms or organizations. Practicing systems engineers may find themselves concerned primarily with the conception, configuration, architecture, arrangement, interfacing, behavior, and integration of parts or subsystems into a functioning whole, performing effectively in its operational environment,⁵ and only indirectly with the “internals” (such as technology or structure) of the parts or subsystems. Hence the systems-engineering mantra, “form, fit and function.”

These four guiding principles of systems engineering (the systems approach, holism, synthesis, and organicism) inform and address all forms and “styles” of systems engineering. Indeed, if these guiding principles are not observed, it may be unreasonable, perhaps, to categorize such activities and processes as systems engineering. If, on the other hand, they are observed, then systems engineering will be based on system science.

SE Principle E: Adaptive Optimizing

Complex systems adapt to maintain their performance optimally effective in changeable, problematic situations. For purposeful, manmade sociotechnical sys-

tems, one way to keep abreast of continual change is continual redesign (Hitchins 2008: 436 – 437). Continual redesign addresses the problem space, detecting and addressing changes in situation, operational environment, other interacting systems, and other factors; it continually conceives, designs, and implements or reconfigures the whole solution system to perform with optimal effectiveness in the contemporary operational environment.⁶

Optimal effectiveness analyses may employ, as the objective function, cost-effectiveness (effectiveness divided by cost), casualty-exchange ratios, cost-exchange ratios, or some combination of these and others. Optimal effectiveness may also be examined over a range of operational scenarios, to avoid undue specificity, and, in appropriate situations, may be achieved by changes in training, organization, procedures, strategy, and tactics.

Defense and aerospace organizations presently may redesign the whole after the operational and support systems have been “put to work,” creating a so-called “midlife update,” designed to re-optimize the effectiveness of the whole system, in line with the contemporary, evolving problematic situation – and to take advantage of new technology that has become available in the intervening period. Continual redesign takes periodic upgrade a stage further, to become continual or even continuous, starting before the first delivery of an operational system. The objective is to maintain operational performance in an optimally effective state from the point of delivery and throughout changing situation and circumstance. Continual redesign requires that corresponding operational and support systems are designed for, and amenable to, continual change. Continual redesign is compatible with *kaizen*, the Japanese philosophy of continual (or continuous) improvement.

Continual redesign can be built into “fielded” systems: operational systems may reconfigure themselves to maintain optimal effectiveness. This capability already exists in some remote space systems, where physical human access is not feasible. Similarly, every time the operating system updates on my iMac, it goes into a protracted process of “re-optimizing”: I hope it knows what it is doing...

SE Principle F: Progressive Entropy Reduction

The process of systems engineering moves from problematic abstraction, disorder, and dysfunction, progressively towards order and function, resulting in a tangible, purposeful solution. As systems engineering moves from the problem space towards the fielded solution, the knowledge of the whole, the parts, and their interrelationships moves from the vague, abstract, incoherent, disordered, and incomplete towards the comprehensive, structured, organized, specific, complete,

4. Wikipedia, s.v. “organicism,” <http://en.wikipedia.org/wiki/Organicism> (accessed 20 Oct. 2009).

5. This is “looking upwards and outwards,” consistent with the philosophies of systems-as-a-discipline, systems thinking, and systems engineering.

6. Continual performance and capability improvement of systems in operation is sometimes called *operational systems engineering*, and may be undertaken by customer or user organizations with or without support from industry, as they seek to “get the best” out of their systems in demanding situations.

and tangible. In terms of knowledge or information, this process involves progressively reducing entropy, going from a condition of high entropy (that is, disorder) at the outset to low entropy (order) at the finish.

Satisficing versus Optimizing

Systems engineering may be viewed as a problem-solving paradigm. Decision theorists identify three archetypal ways to address a problem (Ackoff 1981):

- *Solve* the problem — find a correct (or optimum) answer, as in an equation.
- *Resolve* the problem — find an answer that is “good enough,” to “satisfice.”⁷
- *Dissolve* the problem — change the situation such that the problem no longer arises (often the “smart” choice of politicians).

The long-standing debate between satisficing and optimizing is of great significance to systems engineering.⁸ Advocates of satisficing (Simon 1997: 295–298) believe that it has an advantage over optimizing. The theory of bounded rationality (Simon 1997: 291–294) postulates that decision-makers lack the ability and resources to arrive at optimal solutions, so instead they apply rationality only after having greatly simplified the choices available. Consequently the decision-maker is seen as a satisficer, one seeking a satisfactory solution rather than the optimal one.

Simon's theory of bounded rationality is not without its detractors: “This theory does not categorically assert that it is better to satisfice than optimize... If a decision maker could optimize, it surely should do so. Only the real-world constraints on its capabilities prevent it from achieving the optimum. By necessity, it is forced to compromise, but the notion of optimality remains intact” (Stirling 2003: 10).

Satisficing may be seen as a pragmatic approach in constrained situations. A particular instance arises in defense procurement, where the operational environment in which the solution system will operate is continually changing. At the same time, complex defense and aerospace systems may take many years to realize, during which the initial, “most effective” design configuration would no longer be “most effective” under changed conditions and operational environments. Operational systems may be obsolescent upon delivery. Even in the relatively stable 1960s, large command-and-control systems had to be 67% and 95% redone to fit the environmental changes that had occurred during their development (Boehm 1973). Since many defense systems-engineering projects continue to operate in this way (primarily due to contractual constraints on the defense systems-engineering process), it is worth enunciating one final systems-engineering principle.

7. *Satisficing* (a portmanteau of *satisfy* and *suffice*, coined by Herbert Simon) is a decision-making strategy that attempts to meet criteria for adequacy, rather than to identify an optimal solution.

8. See Sniedovich 2009 for a balanced discussion of optimization versus satisficing, Pareto optimization, adaptive optimization, and related issues.

SE Principle G: Adaptive Satisficing

“Successful systems engineering involves a continuing process of adapting the system's requirements and solutions to enable the system to produce mutually satisfactory results for its success-critical stakeholders” (Boehm and Jain 2006). In the context of the INCOSE definition of systems engineering as enabling the realization of successful systems, principle G implies two sub-principles, as given by Boehm and Ross (1989):

SE Sub-Principle G1: System Success. “A system will succeed if and only if it makes winners of its success-critical stakeholders.”

SE Sub-Principle G2: System Success Realization. “Making winners of your success-critical stakeholders requires

1. identifying all of the success-critical stakeholders (SCSs);
2. understanding how the SCSs want to win;
3. having the SCSs negotiate a win-win set of product and process plans;
4. controlling progress toward SCS win-win realization, including adaptation to change.” ⓘ

References

- Ackoff, R. L. 1981. *Creating the corporate future*. New York: Wiley.
- Alexander, C. 1964. *Notes on the synthesis of form*. Cambridge, MA: Harvard Univ. Press.
- Bertalanffy, L. von. 1962. General systems theory: A critical review. *General Systems* 7:1–20.
- . 1968. *General system theory: Foundations, development, applications*. New York: George Braziller.
- Boehm, B. 1973. Software and its impact: A quantitative assessment. *Datamation* 19 (May): 48–59.
- Boehm, B., and R. Ross. 1989. Theory-W software project management: Principles and examples. *IEEE Transactions on Software Engineering* 15 (7): 902–916.
- Boehm, B., and A. Jain. 2006. A value-based theory of systems engineering. Paper presented at the Sixteenth Annual International Symposium of INCOSE (Orlando, FL).
- Hall, A. D., III. 1989. *Metasystems methodology: A new synthesis and unification*. Oxford, U.K.: Pergamon Press.
- Hitchins, D. K. 2008. *Systems engineering: A 21st century systems methodology*. Chichester, U.K.: Wiley.
- . 1992. *Putting systems to work*. Chichester, U.K.: Wiley. Also available at <http://www.hitchins.net/SysBooks.html#PSTW>.
- . 2003. *Advanced systems thinking engineering and management*. Boston, MA: Artech House.
- Jenkins, G. M. 1981. The systems approach. In *Systems behaviour*, 3rd ed., ed. R. J. Beishon and G. Peters, 142–168. London: Open Univ. Press (Harper & Row).
- Kast, F. E., and J. E. Rosenzweig. 1981. Organization and management. In *Systems behaviour*, 3rd ed., ed. R. J. Beishon and G. Peters, 44–58. London: Open Univ. Press (Harper & Row).
- Sniedovich, M. 2009. The satisficing vs optimizing debate. *Decision-making under severe uncertainty* Web site, <http://www.moshe-online.com/satisficing/> (accessed 20 Oct. 2009).
- Simon, H. 1997. *Empirically grounded economic reason*. Vol. 3 of *Models of bounded rationality*. Cambridge, MA: MIT Press.
- Stirling, W. C. 2003. *Satisficing games and decision making*. Cambridge, U.K.: Cambridge Univ. Press.
- Sullivan, L. 1947. The tall office building artistically considered. In *Kindergarten chats (revised 1918) and other writings*, ed. I. Athey, 202–13. New York: Wittenborn, Schultz.



7th European Systems Engineering Conference
EuSEC 2010, Stockholm, Sweden,
May 23 - 26, 2010

Systems Engineering and Innovation

WELCOME TO EUSEC 2010!

Make sure to mark your calendar for the European Systems Engineering Conference (EuSEC) organized by INCOSE Region 3 in Stockholm, Sweden, May 23-26, 2010!

The theme for EuSEC 2010 is “Systems Engineering and Innovation” and it addresses industry, government and academia.

At EuSEC 2010 you will meet a balanced mix of quality technical papers sessions, panels, tutorials, as well as full-day industrial and academic tracks.

We warmly welcome you, your colleagues and everyone you bring along, to a thrilling event in the beautiful Nordic spring in Stockholm!

REGISTRATION IS OPEN!

Register early and take advantage of our Early bird conference fee. Last date for early bird registrations is March 31, 2010. Please, visit our web page for more information about the conference, registration, and Stockholm – the Capital of Scandinavia.

IMPORTANT DATES

Notification of accepted papers:	March 8, 2010
Last date for “Early bird” discount:	March 31, 2010
Final Papers Due:	April 12, 2010
Conference:	May 23-26, 2010

CONTACT

For more information please contact Linda Christersson, E-mail EuSEC2010@congreg.com or visit our website.

www.incose.org/eusec2010

Are You Programmable, Inventive, or Innovative?

Jack Ring, jack.ring@incose.org

In *Harnessing Complexity: Organizational Implications of a Scientific Frontier* (New York: Free Press, 1999), R. Axelrod and S. Cohen describe the use of genetic algorithms as the interaction of twelve concepts. These twelve concepts interact according to the following scenario (numbers added to annotate the twelve concepts):

Agents [1] of a variety [2] of types [3] use their strategies [4] in patterned interaction [5] across both physical space [6] and conceptual space [7] with each other and with artifacts [8]. Performance measures [9] on the resulting events drive the selection [10] of agents and/or strategies through processes of error-prone copying [11] and recombination [12] thus changing the frequencies of the types within the system thereby changing the emergent characteristics of the system and creating a new gap relative to desired performance. (Axelrod and Cohen 1999: 154)

A later survey by Robert Plotkin in his article, “The Automation

of Invention” (*The Futurist*, July–August 2009, <http://www.wfs.org/futurist.htm>), describes how genetic algorithms converge on a product design or discover the content and structure of an intervention system for complex, problematic situations.

Now consider your personal participation in a systems-engineering project. Read the annotated paragraph above again, this time envisioning yourself as one of the agents. Do you interact with the diversity of practitioners, exchanging knowledge and adapting your ideas and practices in pursuit of the best set of trade-offs? What competencies, information base, intuition base, and risk-aversion policy can you apply to producing a descriptive model of the problematic situation, and then a responsive prescriptive model of an effective solution? Do you simply apply prescribed practices regardless of outcome, or can you collaboratively devise and evaluate various alternatives, as do genetic algorithms, estimating the likelihood of an effective system being realized? Better yet, can you rapidly co-evolve with your colleagues to discover a previously unforeseen idea that makes a dramatic difference for the sponsors?



The Use of Systems Engineering Methods to Explain the Success of an Enterprise

Sonia Bhar Ahluwalia, sahluwal@stevens.edu

Systems-engineering methods are gaining momentum for building and maintaining systems throughout industry and government. These methods have also been used to understand the intangible forces behind the success of a system. In this article, I present a case study based on my work as a master’s student at the Stevens Institute of Technology that shows how systems methodologies can be used to pinpoint the reasons for the success of a Fortune 500 company that is renowned as a great place to work.

Google is a simple yet powerful search engine. It connects users worldwide to an abundance of free information, literally at their fingertips. Advertisers have generated nearly 100 percent of Google’s revenues (Google 2008: 20), yet there is another force to be reckoned with—corporate culture. Google’s corporate culture views the

employee as an individual with the unique potential to “change the world” (Google 2009b).

History

Google was begun unintentionally by two Stanford PhD students, Larry Page and Sergey Brin, who were investigating ways to help users locate information on the Internet. They believed that there should be a way to bring order and structure to the seemingly chaotic and random Internet based on the example of citing references in research papers. In order for a scholar’s research paper to gain the respect of other scholars, the author must cite the research of other respectable scholars. The two PhD students believed that the Internet could be understood similarly as a long chain of references

and cross-references (Boardman and Sauser 2008: 12).

Initially, the students wanted to sell their search algorithm to AltaVista for USD 1 million, but the company refused. These students were disappointed by the rejection, not so much because of the monetary loss, but rather, the loss for humanity, who would never experience powerful ways of getting the information they want. Not taking no for an answer, the students dropped out of the PhD program, created their own company, and blazed a new trail for the Internet—which humanity now knows as Google. This USD 200 billion company's signature search engine has proven so useful for searching information that its name has become a common verb (Boardman and Sauser 2008: 14).

A User's Perspective

To understand Google's business model and how corporate culture factors in, we must first acknowledge its principle, "focus on the user and all else will follow" (Au et al. 2008). Google strives to bake user-friendliness into its products and services, which are mainly Web-deployed applications. Hence Google.com greets all its users with a plain, yet versatile interface where someone can type in a search query and potentially unlock the mysteries of the universe (well, according to some, at least).

"Rich pictures" provide the best way to describe how Google serves their users. The Open University System Group's Web site (<http://systems.open.ac.uk/materials/t552/pages/rich/richAppendix.html>) explains that this term refers to a particular kind of visual representation used to simplify complex situations, as opposed to brainstorming and writing ideas, "because our intuitive consciousness communicates more easily in impressions and symbols than in words." A person can make a rough model of a complex situation by drawing objects, and positioning these objects based on their interpreted relationship to one another. An actor is the driving force behind such depictions: the picture thus represents how the entity in question interacts with the overall situation.

The rich picture in figure 1 shows how the actor, in this case the user, interacts with Google (which encompasses the search engine and the company that works behind the scenes). The user views Google as a simple program that provides access to information of different types (depicted by different colors of the rainbow) based on the user's requests ideas. Google empowers ordinary users by making them feel they have the opportunity to accomplish extraordinary things.

Google's Business Model

At first blush, Google seems like a paradox: its simple interface provides the user with a world of free knowledge, while Google rakes in profits. How is this all possible? It's very simple (of course): Google's automated online-advertising program AdWords allows advertisers to create concise three-line text ads and enables them to specify



Figure 1. Rich picture: User's interaction with Google

possible search terms and Web content that closely match these ads. Then, whenever Google runs a program to display the results of a search query, or Web content, AdWords will execute, and determine which ad to display, based on how well the ad matches the search query or Web content, how often users have clicked the link to the ad in the past, and how much the advertiser is willing to pay Google each time a user clicks on the ad. This ensures that high-paying advertisers are not simply "squatting" on precious Google real estate but that advertisers with relevant ads get a higher return on their investment by displaying these ads to users who are most likely to express interest (Google 2008: 12). Figure 2 uses a "systemigram" (Boardman and Sauser 2008: 87) to illustrate this entire process, which ultimately generates profit for both the advertiser and Google.

As you may note, third-party network members are considered part of Google's family of products and services. The thousands of third-party network members comprise of Web-site owners that use Google's advertising program, AdSense, to profit from their Web content. AdSense is a syndication of AdWords that will pull relevant ads for display along with the Web content to these third-party network member Web sites (free of charge to the network member). Every time a visitor to a network member's Web site clicks on a link for an ad, the advertiser pays Google, who in turn shares the profit with that network member.

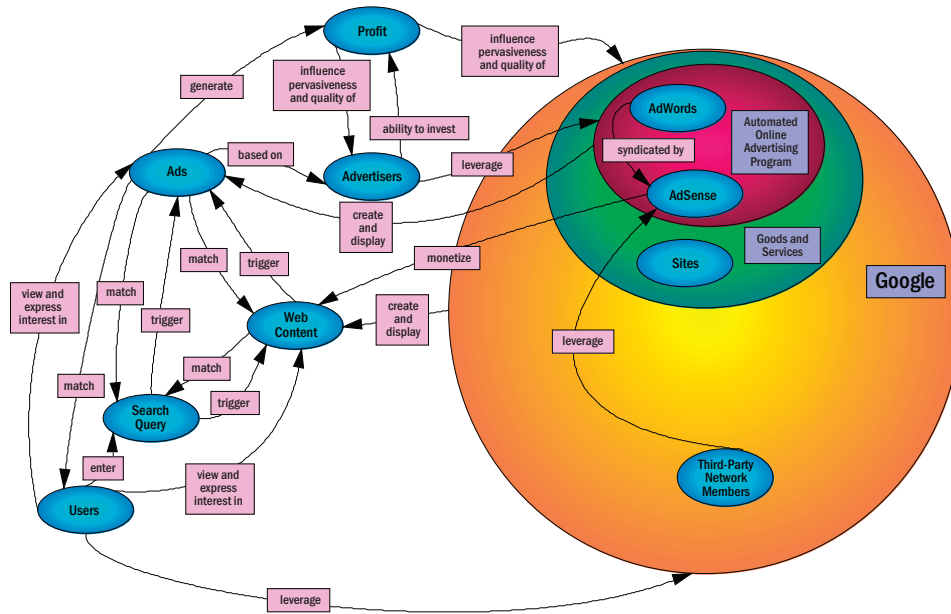


Figure 2. Advertisers drive Google's profit

Google's Corporate Culture

Google competes to attract and retain marketable advertisers by enabling them to create simple ads and ensure these ads are displayed to users who are most likely to respond to them. On the other hand Google also competes to attract and retain highly talented employees. Google was started by two graduate students who, USD 200 billion later, want to maintain a close-knit atmosphere by treating employees as individuals with unique talents and dreams. They believe that knowledge trumps experience and thus hire most employees straight out of college. Google's facilities are a cross between the fun of a theme park and the knowledge hub of world-class universities. Employees bounce technical ideas back and forth while lounging on big comfy sofas and playing with dogs (Google 2009a stresses that Google is a canine-friendly environment where employees can bring their dogs to work).

Google encourages employees to spend 20 percent of their time working on independent projects. Initiatives such as AdWords were shaped under these auspices. Studies in management have demonstrated time and time again that employee appreciation plays a significant role in a company's success, and entire books have been published on this subject (Heskett et al. 1997). Figure 3 shows how corporate culture drives profit.

Amid all the fun, there is the risk that the company might lose site of its ultimate goal, which is to serve the user (Capek 2007). Google addresses this by making it a priority to incorporate user-friendliness into its products and services. This is

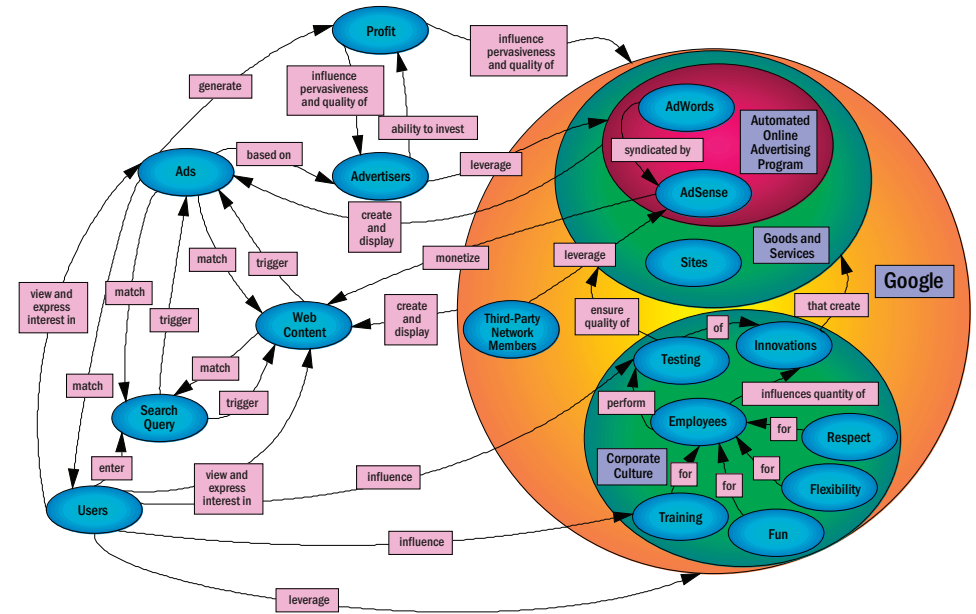


Figure 3. Corporate culture drives Google's profits

accomplished by introducing all new Google employees (also known as "Nooglers") to the User Experience (UX) team. This team is comprised of Google employees located across the globe with diverse skill-sets who teach an orientation course called "Life of a User," which complements the Nooglers' technical training when they first join Google. "Life of a User" immerses the new employees in user-friendliness concepts and principles. UX team members are available to dispense advice on user-friendliness for the hundreds of active projects worked on at any given time. Plus, there are reference materials available that allow development teams to learn and benefit from previous designs (Au et al. 2008).

At later phases of application development, the initial product is posted onto Google Labs, where users can use and test the application to provide feedback. User interactions with applications are tracked using Web analytics, which the UX team studies to understand how applications are being used, what is working, or not working so that development teams can make improvements and upgrades accordingly (Au et al. 2008).

Since employees can choose to work on independent projects that fascinate them rather than projects that address a specific user need, it is imperative for Google to make the applications attractive to users. At the same time, AdWords will "push" relevant ads for display on these applications based on what the user is viewing, so that the users will be more likely to reply to the ads and thus generate profit for advertisers and Google.

Possible Ways to Weather the Economic Storm

The current economic crisis has created a break in the profit-generating models that I illustrated above. Users are being laid off from their jobs, which mean they have less disposable income and are less likely to respond to ads that are relevant to their search queries and Web content they are viewing. In turn, advertisers are getting less return on their investments from their ads, which may cause advertisers to break their agreements with Google.

Because advertisers contribute most of Google's profits, there is a legitimate fear in the company that the amount of advertisements, and therefore the profits, will be drastically slashed. Hence, Google had no choice but to freeze hiring and lay off hundreds of human-resources positions (Das and Lawsky 2009). News of layoffs is a terrible blow to corporate culture because the prospect of losing one's job is always at the back of the employee's mind. Figure 4 demonstrates how the economy could damage Google's business model. Once the bubbles for advertisers and ads are removed, note that there is no longer any indication of how profit will be generated.

However, there are ways to remain resilient and weather this storm. Google's 2008 annual report dealt with many challenges to Google's operations (Google 2008a). Several of these challenges can be addressed, and used to (at the very least) sustain profits.

Modernization and talent are Google's lifeline; CEO Eric Schmidt warned in the

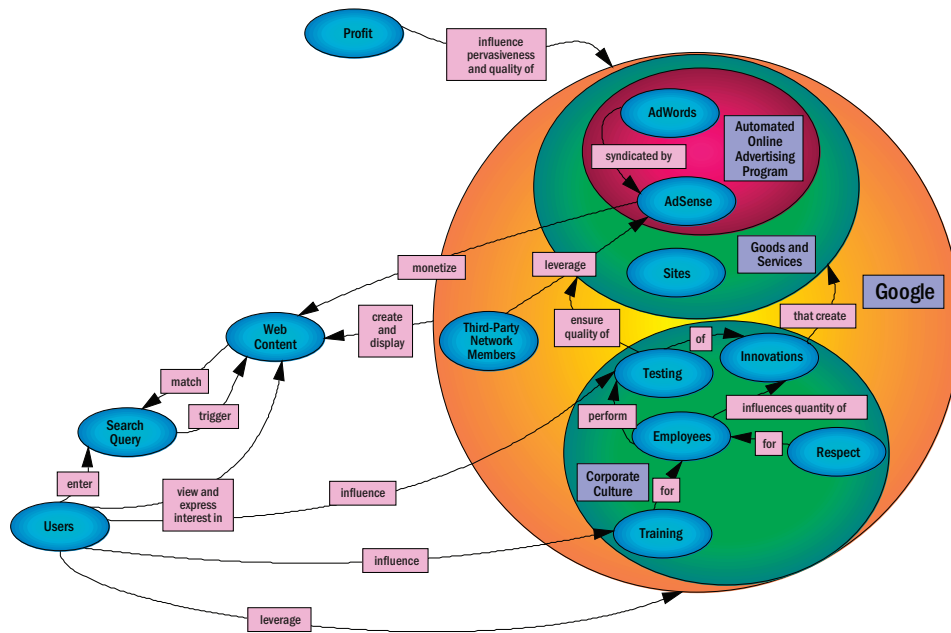


Figure 4. Google's business model without advertisers and their ads. Plus a lack of flexibility and fun for employees.

annual report, "If we do not continue to innovate and provide products and services that are useful to users, we may not remain competitive, and our revenues and operating results could suffer" (Google 2008: 20). He also said, "We rely on highly skilled personnel and, if we are unable to retain or motivate key personnel, hire qualified personnel or maintain our corporate culture, we may not be able to grow effectively" (Google 2008: 27). Currently, Google allows its employees to devote 20 percent of their time to pursue independent projects. Google should raise this to 30 thirty percent at least temporarily. This will further expand the employee's skill set and create peace of mind for the employees in case they are the next to receive a pink slip. This peace of mind may allow the employees to innovate and extend their network of professionals to interact with. An employee with varied skill sets who can think outside the box and is in touch with a large network of individuals across different industries will have the confidence to survive this economic storm. In the unfortunate event that the employee is laid off, the employee will have the opportunity to apply and qualify for more jobs with varying skill sets.

Mr. Schmidt also acknowledged another problem facing Google: "Because our users need to access our services through Internet access providers, they have direct relationships with these providers. If an access provider or a computer or computing device manufacturer offers online services that compete with ours, the user may find it more convenient to use the services of the access provider or manufacturer" (Google 2008: 18). To respond to this problem, Google should consider becoming an Internet service provider (ISP) itself, and provide services that are comparable, or exceed other reputed ISPs.

Another problem Google faces is with regard to international users: "In order to compete, we need to better understand our international users and their preferences, improve our brand recognition, our selling efforts internationally, and build stronger relationships with advertisers. If we fail to do so, our global expansion efforts may be more costly and less profitable than we expect" (Google 2008: 19). Google should encourage international studies by establishing projects that study lesser-known international users and their preferences, for which there is great potential for Google to have market share. It should establish partnerships with university programs that specialize in these areas of study. By this, Google would also be creating goodwill with the university world so when the economy improves it will be easier to hire and retain recent graduates. These efforts may ensure that Google stays afloat until it can restructure its business model to better dodge the punches and blows of the economy, while continuing to focus on the user above all else.

Conclusion

Systems-engineering methods can be used to understand the inner workings of



SYSTEMS ENGINEERING TRAINING

Our SE Courses Include

INCOSE CSEP
 SE Training Level I and II
 Systems Engineering for ITS
 Requirements Analysis
 MIL-1553 Fundamentals
 Product Safety
 Product Development
 Reliability Engineering
 Root Cause Failure Analysis
 SCADA I and II
 CCSDS
 DoDAF, MoDAF, NAF, and TOGAF
 Wireless Systems Engineering
 Six Sigma

Contact Us to Learn More!

INFO@TONEX.COM
 1 - 888 - TO - TONEX
 WWW.TONEX.COM

Ahluwalia *continued*

an enterprise and explain why it has flourished. By the same token these methods have been used to provide solutions for sustaining success in troubling times.

References

- Au, I., R. Boardman, R. Jeffries, P. Larvie, A. Pavese, J. Riegelsberger, K. Rodden, and M. Stevens. 2008. User experience at Google: Focus on the user and all else will follow. Paper presented at the Conference on Human Factors in Computing Systems (Florence, Italy). Available at <http://portal.acm.org/citation.cfm?doid=1358628.1358912> (accessed 7 July 2009).
- Boardman, J., and B. Sauser. 2008. *Systems thinking: Coping with 21st Century problems*. Boca Raton, FL: CRC Press.
- Capek, F. 2007. A break in the service profit chain: Why increases in employee engagement don't improve the customer experience. *Customer innovations: Driving profitable growth* Weblog, 16 November 2007. <http://customerinnovations.wordpress.com/2007/11/16/> (accessed 7 July 2009).
- Das, A., and D. Lawsky. 2009. Google orders first layoffs in its history. *Financial Post*, 15 January. <http://www.financialpost.com/news-sectors/story.html?id=1180389> (accessed July 7, 2009).
- Google. 2008. Google 2008 annual report http://investor.google.com/pdf/2008_google_annual_report.pdf (accessed 7 July 2009).
- _____. 2009a. Google code of conduct. *Google investor relations* Web site. <http://investor.google.com/conduct.html> (accessed 7 July 2009).
- _____. 2009b. Google jobs. <http://www.google.com/intl/en/jobs/> (accessed 7 July 2009).
- Heskett, J. L., W. E. Sasser, and L. A. Schlesinger. 1997. *The service profit chain: How leading companies link profit and growth to loyalty, satisfaction, and value*. New York: Free Press.
- Open University Systems Group. 2009. Rich pictures. The Open University. <http://systems.open.ac.uk/materials/t552/pages/rich/richAppendix.html> (accessed 7 July 2009).

Technical Activities

Announcing BKCASE: Body of Knowledge and Curriculum to Advance Systems Engineering

Alice Squires, alice.squires@stevens.edu; Art Pyster, arthur.pyster@incose.org; David Olwell, david.olwell@incose.org; Stephanie Few, smfew@nps.edu; and Don Gelosh, donald.gelosh@incose.org

In September 2009, Stevens Institute of Technology, together with the Naval Postgraduate School, began the Body of Knowledge and Curriculum to Advance Systems Engineering (BKCASE, pronounced “bookcase”) project. BKCASE is a three-year effort to create a robust Systems Engineering Body of Knowledge (SE BoK) and a Graduate Reference Curriculum in System Engineering (GRCSE, pronounced “Gracie”). Endorsed by the INCOSE Board of Directors, with significant funding from the U.S. Department of Defense and support from the IEEE Systems Council, BKCASE is the response to a call from government and industry for a globally recognized, community-created foundation for the discipline of systems engineering. The BKCASE project hopes to materially influence standard practice, workforce models, certification, and graduate education around the world.

Figure 1 describes BKCASE, showing the project in the upper left-hand corner, and the products—comprised of SE BoK and GRCSE—in the lower right-hand corner. The BKCASE systems diagram describes the project development through a “story” of the relationships between the project and products, the systems-engineering community, and the various products in the community that will be developed based on BKCASE. The BKCASE vision is that competency models, certification programs, textbooks, graduate programs, and related workforce-development initiatives for systems engineering around the world will align themselves with BKCASE.

The SE BoK will define and organize the vast knowledge of the discipline of systems engineering, including its methods, processes, practices, and tools. Within that organization, the SE BoK will point to many thousands of pages of articles, books, Web sites, and other sources of knowledge about systems engineering. The SE BoK will facilitate a common understanding of the core of the field, and will aid fast and efficient knowledge retrieval. The SE BoK will build consensus on the boundary of the discipline and facilitate communication among systems engineers.

GRCSE will be based on the SE BoK and will define the entrance expectations, curriculum architecture, curriculum content, and expected student outcomes for graduate programs in systems engineering. GRCSE will recommend that students

learn about the application of systems engineering in an application domain or business segment. The use of GRCSE for guidance will enable consistency in student proficiency at graduation, making it easier for students to select where to attend and for employers to evaluate prospective new graduates.

The BKCASE team includes invited authors and volunteer reviewers from around the world representing different locales, business segments, professional societies, and areas of expertise. The team has representation from government, industry, and academia. Authors volunteer their time for one or two days per month, attend quarterly workshops, and participate in periodic virtual meetings. Reviewers work as time permits. Once fully staffed, the team will have thirty to forty authors and several hundred reviewers. Some authors and reviewers will work on both SE BoK and GRCSE; others will work on only one product.

Two interim drafts and the final products will be developed in one-year intervals starting in June (SE BoK) and September (GRCSE) of 2010, with version 1.0 products due out in 2012. Both INCOSE and the IEEE Systems Council will be heavily involved from the beginning, possibly leading them to take up maintenance responsibility for BKCASE products and to adopt them in their own products such as the INCOSE *Systems Engineering Handbook* and INCOSE professional certification program. Anyone interested in supporting BKCASE in any capacity, or anyone who has source material to offer, please contact the project leader, Art Pyster, by e-mail at art.pyster@stevens.edu. For additional information on BKCASE, please see <http://www.bkcase.org>.

New Guidelines for Graduate Software-Engineering Education

Mark Ardis, mark.ardis@stevens.edu; Tom Hilburn, hilburn@erau.edu; and Art Pyster, art.pyster@incose.org

A new set of guidelines for graduate software-engineering education was recently published by Stevens Institute of Technology. In 2007 academia, industry, government, and professional societies formed a coalition called the Integrated Software and Systems Engineering Curriculum (iSSEc) project to create a reference curriculum that reflects current development practices and the greater role of software in today’s systems. The guidelines are published as the *Graduate Software Engineering 2009 (GSWE2009): Curriculum Guidelines for Graduate Degree Programs in Software Engineering* and are available at <http://www.gswe2009.org>. Earlier versions of this work used the name “Graduate Software Engineering Reference Curriculum (GSWEREC).”

One of the primary goals of the iSSEc project was the incorporation and integration of systems-engineering knowledge and practices into graduate software-engineering programs. Large systems today include significant software content. The software engineers who work on these systems need to understand better the relationships between hardware, software and human components. INCOSE has been an active member of the iSSEc project, participating in authorship, review, and promotion of the effort.

We are indebted to the many experts who helped create the guidelines. A complete list of those participants and their supporting organizations is included in the report. We are especially grateful to Kristen Baldwin and others in the U.S. Office of the Secretary of Defense for their consistent, generous, and thoughtful support of this project.

History of the Project

In 1989 the Software Engineering Institute (SEI) of Carnegie Mellon University published a landmark report on graduate education in software engineering (Ardis and Ford 1989). Several universities in establishing their software-engineering degree programs used the recommendations in that report. Since then, the way software is developed has changed dramatically. Software’s scale, complexity, and criticality have mushroomed, yet no significant effort has been made to revisit and update the original SEI recommendations.

GSWE2009 builds on the SEI curriculum foundations plus those of other initiatives, such as the *Guide to the Software Engineering Body of Knowledge (SWEBOK)*; Bourque and Dupuis 2004) and *Software Engineering 2004: Curriculum Guidelines*

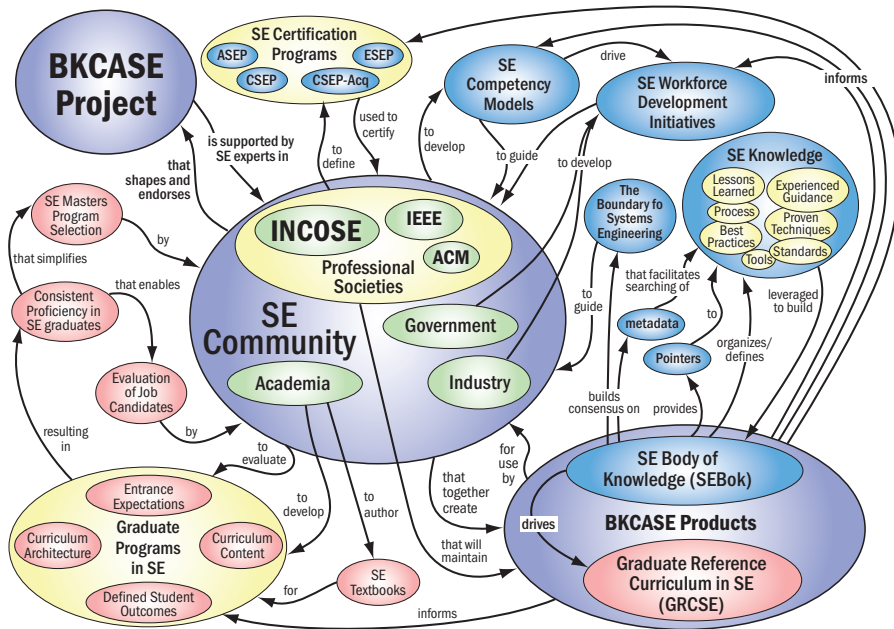


Figure 1. Systems diagram describing BKCASE

Alice Squires 11/2/2009

for *Undergraduate Degree Programs in Software Engineering* (ACM and IEEE 2004). The iSSEc project followed an iterative, evolutionary approach in creating GSwE2009, beginning with the formation of a curriculum author team (CAT). First established in July 2007, the CAT is a set of invited experts from industry, government, academia, and professional associations. CAT membership grew as GSwE2009 matured.

The CAT met in workshops approximately every three months between August 2007 and September 2009, leading to the release of GSwERC 0.25 in February 2008, GSwERC 0.5 in October 2008, and GSwE2009 1.0 in September 2009. The software-engineering community was invited to review versions 0.25 and 0.5 to provide the necessary feedback to develop the current version (1.0). The review of version 0.5 generated more than 800 individual review comments, which were adjudicated for use in creating version 1.0. The detailed comments and their adjudication can be found at <http://www.GSwE2009.org>.

Professional-society participation in the creation of GSwE2009 has been essential to ensuring that GSwE2009 will have the desired impact on global graduate education. Both INCOSE and the U.S. National Defense Industrial Association (NDIA) Systems Engineering Division were early participants in GSwE2009, and each contributed authors. In 2008, the Institute of Electrical and Electronics Engineers (IEEE) Computer Society Education Activities Board became an official participant. In 2009 the Association for Computing Machinery (ACM), the IEEE Computer Society, and the Brazilian Computer Society (BCS) also chose to participate. GSwE2009's success has motivated the start of related efforts by the BKCASE project to create a Systems Engineering Body of Knowledge and a Graduate Systems Engineering Reference Curriculum—each with an “appropriate” amount of software-engineering perspective and content. The BKCASE efforts should lead to version 1.0 products in 2012.

Content of Curriculum Recommendations

GSwE2009 includes the following elements:

- A set of outcomes to be fulfilled by a student who successfully completes a graduate program based on the curriculum
- A set of student skills, knowledge, and experience assumed by the curriculum, not intended as entrance requirements for a specific program, but as the starting point for the curriculum's outcomes
- An architectural framework to support implementation of the curriculum
- A description of the fundamental or core skills, knowledge, and practice to be taught in the curriculum to achieve the outcomes. This is termed a Core Body of Knowledge (CBOK) and includes topic areas and the depth of understanding a student should achieve.

A university considering the creation or modification of a graduate software engineering program should be able to use the CBOK and the architectural framework to design appropriate courses and degree requirements. The outcomes and entrance assumptions should help in determining the expected market and value of the program to potential students and their employers.

In addition, GSwE2009 includes the following:

- The fundamental philosophy for GSwE2009 development as described in a set of guiding principles
- A discussion of how GSwE2009 will evolve to remain effective
- A mapping of expected outcomes to the CBOK and to the total GSwE2009 program recommendations
- A description of Knowledge Areas (KAs) discussed in GSwE2009 that are not yet fully integrated into the current version of the Software Engineering Body of Knowledge (SWEBOK)
- Glossary, references, and other supporting material.

Expected Student Outcomes

Graduates of a master's program that satisfies GSwE2009 recommendations will do the following:

- Master the Core Body of Knowledge (CBOK).
- Master software engineering in at least one application domain, such as finance, medical, transportation, or telecommunications, and one application type, such as real-time, embedded, safety-critical, or highly distributed systems. That mastery includes understanding how differences in domain and type manifest themselves in both the software itself and in its engineering, and includes understanding how to learn a new application domain or type.
- Master at least one Knowledge Area (KA) or subarea from the CBOK to at least the Bloom Synthesis level (Bloom 1956).
- Be able to make ethical professional decisions and practice ethical professional behavior.
- Understand the relationship between software engineering and systems engineering and be able to apply systems-engineering principles and practices in the engineering of software.
- Be an effective member of a team, including teams that are international and geographically distributed, effectively communicate both orally and in writing, and lead in one area of project development, such as project management, requirements analysis, architecture, construction, or quality assurance.
- Be able to reconcile conflicting project objectives, finding acceptable compromises within limitations of cost, time, knowledge, existing systems, and organizations.

- Understand and appreciate feasibility analysis, negotiation, and good communications with stakeholders in a typical software development environment, and be able to perform those tasks well; have effective work habits and be a leader.
- Be able to learn new models, techniques, and technologies as they emerge, and appreciate the necessity of such continuing professional development.
- Be able to analyze a current significant software technology, articulate its strengths and weaknesses, compare it to alternative technologies, and specify and promote improvements or extensions to that technology.

Curriculum Architecture

Figure 1 provides an overview of the curriculum architecture. GSwE2009 identifies the fundamental skills and knowledge that all graduates of a master’s program in software engineering must possess. This is captured in the half-circle area labeled *Core Materials*. These skills and knowledge include such topics as systems-engineering fundamentals, requirements engineering, software design, and ethics and professional conduct.

The next half-circle in figure 1, labeled *University-Specific Materials*, represents materials that an institution might include in order to tailor its program to meet its specific objectives. These will vary by institution or degree program. For example, a program that emphasizes safety-critical systems might have a required course on such systems that would be part of the university-specific materials.

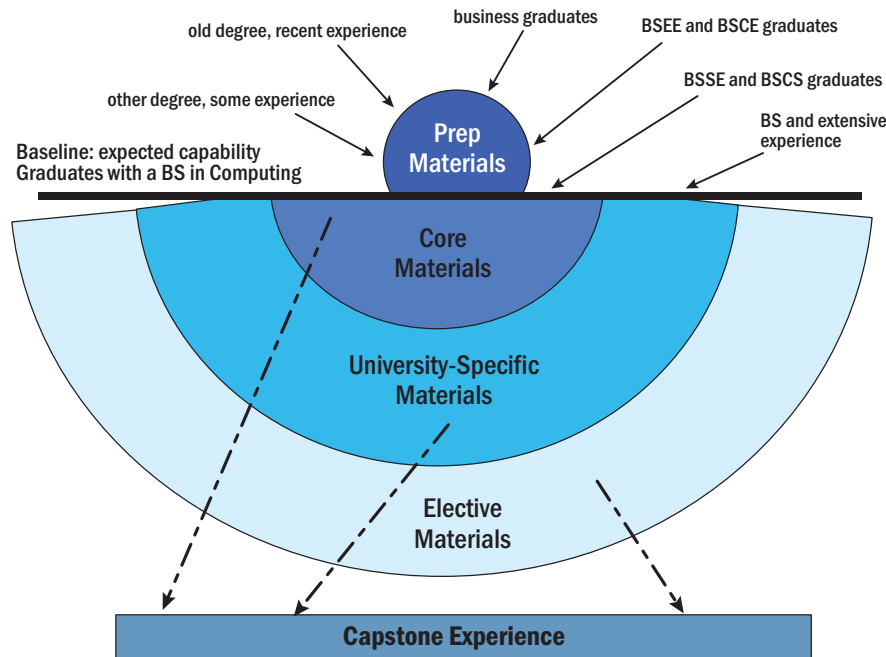


Figure 1. GSwE2009 curriculum architecture

Elective Materials accommodate different interests of individual students, but may still reflect a program focus. For example, a program may focus on information security, verification and validation (V&V), or health-care systems, providing a series of courses that allow a student to gain depth in a technical area.

Core Body of Knowledge

The Core Body of Knowledge (CBOK) includes all of the fundamental or core skills, knowledge, and experience to be taught in the curriculum to achieve the expected student outcomes. The primary source for developing the CBOK was the SWEBOK. Knowledge elements were also derived from the *Software Engineering 2004* curriculum guidelines (ACM and IEEE 2004), the *INCOSE Guide to Systems Engineering Body of Knowledge* (INCOSE 2004), and especially the *INCOSE Systems Engineering Handbook* (Haskins 2007).

Figure 2 shows the knowledge elements of CBOK and their expected relative proportions of the GSwE2009 curriculum. Although specific systems engineering knowledge elements only represents 2–3% of the CBOK, they are considered a

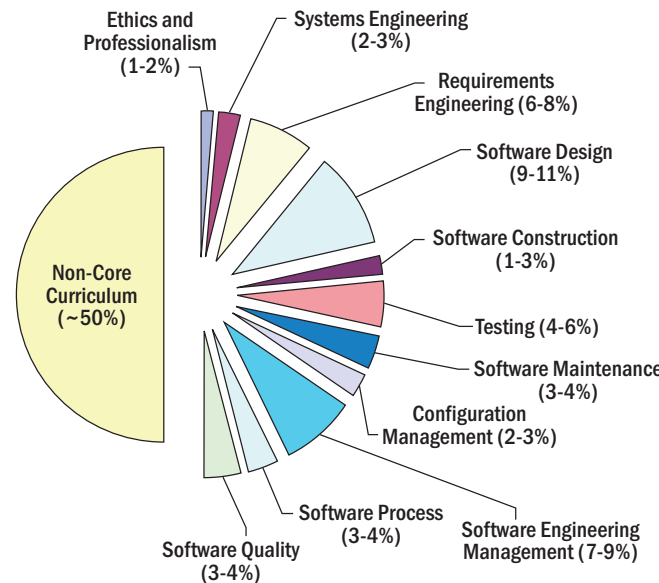


Figure 2. CBOK knowledge elements as percentages of GSwE2009 curriculum

crosscutting concern that arises in many other areas. For example, systems-engineering material would also be covered under requirements engineering, testing, configuration management, and project management.

Systems Engineering in the Curriculum

Of particular interest to *INSIGHT* readers should be the discussion of the systems-


engineering knowledge area and its integration into the curriculum guidelines. You are encouraged to review, for example, section 6.5 on “Systems Engineering Issues” of the CBOK and appendix C.2, “Systems Engineering.”

The iSSEc project has just completed two companion reports to help schools interested in creating or modifying graduate software engineering programs. The first report compares current programs to the guidelines, and the second

Notes to the Editor

Book Review | *Analytical Methods for Risk Management: A Systems Engineering Perspective*

Reviewer's Response to Author's Reply—Mark Powell, mark.powell@incose.org

answers common questions about implementation of graduate degree programs in software engineering. Please visit the GSwE2009 Web site, <http://www.gswE2009.org>, to download the reports. You may also use the site to submit comments or questions. Your help in improving the recommendations is greatly appreciated. 

References

- ACM and IEEE (ACM/IEEE Computer Society Joint Task Force on Computing Curricula). 2004. *Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering*. <http://www.acm.org/education/curricula-recommendations>.
- Ardis, M., and G. Ford. 1989. *SEI report on graduate software engineering education*. CMU/SEI 89-TR-21. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
- Bloom, B. S., ed. 1956. *Taxonomy of educational objectives: The classification of educational goals; Handbook I, cognitive domain*. Place: Longmans.
- Bourque, P., and R. Dupuis, eds. 2004. *SWEBOOK: Guide to the Software Engineering Body of Knowledge*. Los Alamitos, CA: IEEE Computer Society Press.
- Haskins, C., ed. 2007. *Systems engineering handbook: A guide for system life cycle processes and activities*. Version 3.1. Rev. by K. Forsberg and M. Krueger. Seattle: INCOSE.
- INCOSE. 2004. *Guide to Systems Engineering Body of Knowledge*.

INSIGHT

In the October 2009 issue of *INSIGHT*, Paul Garvey took rather vigorous exception to my review of his new book *Analytical Methods for Risk Management: A Systems Engineering Perspective* (review published in the April 2009 issue of *INSIGHT*). Mr. Garvey lists eleven different comments in my review that he considers to be “reviewer inaccuracies.” Strangely, most of his response and commentaries to these “inaccuracies” are largely orthogonal to my review comments. In the interest of INCOSE members who are involved in risk management in their jobs, I believe it is necessary to respond to Mr. Garvey’s rebuttal of my review of his book. But first let me provide some background.

I volunteered to review this book because the title intrigued me. *Analytical methods* in my experience refers to mathematics beyond multivariable calculus, but more generically suggests a quantitative vice qualitative/subjective approach. The term *risk management* covers a lot of territory. So the title, *Analytical Methods for Risk Management*, implied that this book would discuss quantitative mathematical approaches beyond simple four-function math to be used in performing all of the functions of risk management, addressing as a minimum risk identification, risk analysis, risk assessment, risk mitigation, risk tracking, and risk metrics. I have yet to encounter a text that attempts this. Then there is the subtitle, *A Systems Engineering Perspective*. We systems engineers seem to always get the dirty job of risk management on projects. We face a lot of challenges in risk management that the graduate business-school texts on risk management do not address. I anticipated as a result of this subtitle that this book would indeed address a generous sampling of those challenges specific to systems engineering with quantitative analytical solutions. This title heralded much promise for INCOSE members for improving their performance of risk management in their jobs.

The preface of this book reinforced my initial expectations as generated by the title. Terms were used in describing the

intent of the book such as “analytical principles” and “implementation” and “practice.” Then it recommended for its reading a “mathematical background in differential and integral calculus.” The preface, in reference to the body of literature available on risk management, stated that this book “provides managers and systems engineers a guide through the foundational processes, analytical principles, and implementation practices of engineering risk management.” This sounded analogous to what James Martin’s great *Systems Engineering Guidebook* (1997) accomplished for systems-engineering management in general. Fulfillment of the objectives Garvey stated in his preface would greatly benefit INCOSE systems engineers, and would address a number of topics often discussed in the INCOSE Risk Management Working Group.

To prepare for my original review, I read Garvey’s book cover to cover three times and took copious notes. I have scanned this book again considering Mr. Garvey’s rebuttal to my review. I will now address Mr. Garvey’s misidentified “reviewer inaccuracies.” Rather than take up a lot of space quoting my original review comment and Garvey’s rebuttal point, I will use Garvey’s point numbering, synopsize my original review comment, and then respond.

Point 1. *My comment on Garvey’s lack of coverage of the literature in Bayesian statistics*. Garvey’s discussion of things Bayesian, as he points out in his rebuttal, covered seven pages and five problems for the students. However, Garvey’s pages 24–31 merely state Bayes’ law and provide examples of algebraic manipulations of it. Garvey provides five problems for the student to exercise this simple algebra of Bayes’ law, problems one might find in a general undergraduate text on probability and statistics. This sum total of coverage does not constitute a guide to over eighty years of development of modern Bayesian statistics. While Garvey uses the term “Bayesian inference” rather loosely in pages 28–31, there is much, much more to Bayesian inference and statistics than simply exercising the

algebra of Bayes' law. For a sampling of textbook coverage of modern Bayesian inference and statistics, I recommend considering just a few of the many texts available on the subject: Jeffreys (1939), Raiffa and Schlaifer (1961), Schmitt (1969), Box and Tiao (1973), Berger (1985) (a personal favorite of mine), Bernardo and Smith (1994), Gelman, et al. (1995) (used as a text in a number of my courses), Sivia (1996), and Robert and Casella (1999). This is by no means an exhaustive bibliography on this subject, even in my own personal library. Garvey's title and preface led me to expect to see at least a few of these listed in the references in Garvey's book, with some actual discussion of the type of Bayesian "inference" and "statistics" covered in these texts.

Point 2. *My comment praising Garvey for addressing the invalid use of ordinal scales and the product formula.* Garvey clearly identifies that there are some serious problems using the product formula to develop "risk score" isocontours (actually, there are problems using it with both ordinal and cardinal scales). On page 115, Garvey states that "an *impact times probability* approach for ranking risks *within an ordinal risk matrix* should not be used." How else should the reader interpret that statement other than "Do NOT use this formula"? I agree with Garvey very strongly on this, by the way.

Point 3. *My note on Garvey's lack of discussion of the risk aversion artificially introduced by the product formula.* Garvey's rebuttal refers to figure 4.21, which does show the concave isocontours for risk scores as produced by the product formula. Garvey's rebuttal statement about these being the consequence of the "mathematical behavior of the product formula" is true, and fully consistent with my point. Even with "proper calibration" of both axes in the matrix (the scaling of both the probability and consequence axes into something approaching linearity), the product formula still produces these concave isocontours. The product formula artificially imparts "risk aversion" via the math, which is its inherent problem. This becomes glaringly obvious if one borrows a utility-construction method to develop the risk-score isocontours and compares with those produced using the product formula. Risk aversion and tolerance should only be applied by the decision maker, who must decide whether or not to expend resources on mitigation of a risk, not by the math. Garvey's rebuttal refers to page 153, which I reread again looking intensely for hints of "risk aversion," but found nothing. This is a really important pathology of the product formula that is not widely understood. The very fact that Garvey identified that something was wrong with the product formula is in my opinion laudable, something few risk management texts do. A clear explanation of specifically what is wrong can really help systems engineers avoid a number of pitfalls in performing risk management.

Point 4. *My use of the term "risk factors" in my review.* Garvey did not use this term in his book, because as stated in his rebuttal, it is "imprecise." I will contend that it is certainly no more imprecise than the terms *risk scores* or *risk levels*, all used synonymously with *risk factors* across a variety of sources. Back in the 1970s

when I got into risk management, nobody used the term *risk scores*; they used the term *risk factors*. Old terminology apparently sticks with you even when it becomes out of vogue. I refer the reader to old versions of *MIL-STD 882* from back then, and the *DSMC System Engineering Management Guide* (1983) for examples. In INCOSE, we still see the term *risk factors* used quite often, and we all know that it means the same thing as risk scores.

Point 5. *My comment on Garvey's lack of comprehensive coverage of "risk factor formulae."* Garvey found fault with the product formula for calculating risk score isocontours. It is then appropriate that his book address a comprehensive sampling of other risk score formulae, and further explain the issues associated with each. None of them is perfect. The formula I presented in my review as an example of an important one omitted in Garvey's book, sometimes referred to as the *parallel* risk factor formula (analogous to calculating the overall resistance of a network of resistors in parallel), was the primary formula used for risk-factor calculations in the U.S. Department of Defense up until sometime in the 1990s. See the aforementioned *DSMC System Engineering Management Guide*. By the way, this *parallel* formula I mentioned artificially imparts, due to the math, risk tolerance, and is just as faulty as the product formula. Ironically, the weighted probability and impact formula that Garvey repeatedly uses for risk ranking can be used as a formula for calculating risk score isocontours, but it is not obvious that Garvey suggests that it be used for such.

Point 6. *My recommendation that this book be considered for optional readings for a graduate-level risk-management course.* This is a favorable comment for this book. I am sent two or three texts on risk management each year by various textbook publishers to consider for use in my courses in risk management and applied decision analysis. Few of these offer any added value to my courses; much less have sufficient and comprehensive material to use as a central text. I thought Garvey, however, had enough good points in his book to recommend it to other professors for additional readings for their students.

Point 7. *My comment on Garvey's omission of discussion of risks with severe impacts and very low probabilities.* Systems engineers face this problem in risk assessment—the statistical processing of data to calculate the assurance that a risk will be realized above or below a specified level. Garvey's book completely ignored the task of risk assessment in general. Proper quantitative risk assessment requires the use of modern-day Bayesian statistics. Garvey's neglect of risk assessment perhaps explains his lack of coverage of Bayesian statistics. The problem that systems engineers face is that for very low-probability risks, you seldom get many if any *event* data to process statistically. This becomes especially aggravated if the consequence is severe. Garvey's rebuttal referred to pages 147–152, which address risk ranking, something entirely different from risk assessment. Two other important points relative to this comment regarding the preface of this book: (1) the Bayesian

statistics used for proper risk assessments are mathematically intense, requiring a “mathematical background in differential and integral calculus”; and (2) the Bayesian statistics texts I identified earlier as not referenced by Garvey offer valid analytical and numerical methods for solving this problem for systems engineers.


Point 8. *My comment on Garvey’s absence of any discussion of “risk analysis.”* Garvey’s rebuttal refers to section 4.3, and rereading it once again, I saw nothing more than a discussion on risk ranking, as Garvey confirms in his rebuttal comment. “Risk analysis” is that activity in risk management where engineering analyses are performed to understand what factors can produce the consequence and at what level, what the sensitivities of the consequence level might be to various factors, what the sensitivities might be for the probability of the consequence at some level being realized, and what data might be available or needed for a risk assessment. Risk analysis, and the analytical methods employed, is highly dependent on the type of technology involved in the specific consequence being considered: for example, electronic failures must be analyzed quite differently than aeronautical failures. An introduction to the spectrum of analytical methods employed by systems engineers in risk analysis would have significantly expanded Garvey’s book, and have been quite useful and appreciated by systems engineers.

Point 9. *My comment on Garvey’s lack of discussion of the myriad of risk-management standards.* Systems engineers often find themselves working on a wide variety of projects in a wide variety of industries during a career. Risk management on one project may have a certain standard proscribed, on the next project another one, and it is not unusual for the engineer to have to select a standard to use on a project where one is not proscribed. Garvey in his rebuttal commentary justified omitting any such discussion of risk-management standards because of the variety of terminology and methods offered between the existing standards. Systems engineers greatly need a good translation matrix between these standards to be effective in performing risk management over the course of their careers. Garvey’s book could have been of incalculable value had it addressed these various standards, with caveats about the differences between them, especially in the terminology vagaries. Here again I refer to Garvey’s stated intent in his preface to provide “implementation practices.” Dealing with this variety of existing standards of risk-management practice remains a serious challenge for systems engineers, and has been identified as a major problem for INCOSE members. The INCOSE 2005 International Symposium panel discussion I mentioned in my review (and documented in the 2005 International Symposium proceedings) did not solve this problem.

Point 10. *My comment on Garvey’s limited discussion of risk management for enterprise systems.* Garvey’s rebuttal refers to section 4.6 to find this coverage. Upon rereading this section, I confirmed that it does have a top-level discussion of “what an enterprise is.” I did not, however, find a substantial set of methods ana-

lytical or otherwise that would help project managers implement and effectively use a risk-management program within an enterprise. I could find nothing in this section or the entire book to justify the use of risk management in an enterprise. Systems engineering and risk management for the enterprise were major themes for the INCOSE International Symposium in 2007, the proceedings of which were not referenced by Garvey. Garvey did provide some interesting discussion on portfolio management from a capability perspective, and tried to tie it to risk management, but that does not address the scope of risk management for the enterprise.

Point 11. *My comment about Garvey leaving systems engineering on the title page.* The systems engineers in INCOSE face serious challenges in the performance of risk management that are unique to their field. There were many systems-engineering-type words used in a lot of places in the book. But after my third cover-to-cover reading, I was still left wondering where in the book were the analytical methods to help the systems engineer address any of these discipline-specific challenges. The discussion of TPMs seemed a bit strange and out of place in this book on risk management, and that was about the only real substantive systems-engineering content I remembered after three readings.

I eagerly await a text on risk management that fulfills the expectations that Garvey’s title and preface inspired. Despite that, I stand by my comments and recommendations for Garvey’s book as stated in my original review, including those that were favorable. Should any INCOSE members require further clarification of my original review comments or the discussion in this rebuttal response, I welcome e-mail contact at the address above. 

References

- Berger, J. O. 1980. *Statistical decision theory and Bayesian analysis*. New York: Springer-Verlag.
- Bernardo, J. M., and A. F. M. Smith. 1994. *Bayesian theory*. West Sussex, U.K.: Wiley.
- Box, G. E. P., and G. C. Tiao. 1973. *Bayesian inference in statistical analysis*. Reading, MA: Addison-Wesley.
- Defense Systems Management College. 1983. *System engineering management guide*. Fort Belvoir, VA: Defense Systems Management College.
- Gelman, A. B., J. S. Carlin, H. S. Stern, and D. B. Rubin. 1995. *Bayesian data analysis*. Boca Raton, FL: Chapman & Hall/CRC.
- Jeffreys, H. 1939. *Theory of probability*. Oxford: Oxford University Press.
- Martin, J. N. 1997. *Systems engineering guidebook: A process for developing systems and products*. Boca Raton, FL: CRC Press.
- Raiffa, H., and R. Schlaifer. 1961. *Applied statistical decision theory*. Cambridge, MA: Harvard University Press.
- Robert, C. P., and G. Casella. 1999. *Monte Carlo statistical methods*. New York: Springer-Verlag.
- Schmitt, S. A. 1969. *Measuring uncertainty: An elementary introduction to Bayesian statistics*. Reading, MA: Addison-Wesley.
- Sivia, D. S. 1996. *Data analysis: A Bayesian tutorial*. Oxford, U.K.: Oxford University Press.
- U.S. Department of Defense. 1969. *MIL-STD 882: Department of Defense; Standard practice for system safety*. Washington, DC: Office of the Under Secretary for Defense Acquisition, Technology, and Logistics.

Final Thoughts

From the Chief Editor

Bob Kenley, insight@incose.org

This is the first issue of **INSIGHT** published in the new horizontal format. We hope that you find that this format enhances readability on a computer screen and that you take advantage of some of the other features, such as hyperlinks to the table of contents on each page, a thumbnail view of the pages, bookmarks for navigation, and live hyperlinks to the Internet and e-mail applications. If you have any suggestions for further enhancements to the layout of **INSIGHT**, I would like to hear from you.

Our upcoming editions of **INSIGHT** have theme topics on our Council's Technical Operations, the symposium in Chicago, and knowledge management for systems engineering. As always, I am thankful to those who have volunteered to serve as theme editors, and I am delighted to receive proposals for future theme editions.

Upcoming submission deadlines and themes for **INSIGHT**

Issue	Submission Date for General Articles	Theme	Theme Editor
1st Qtr 2010	15 Feb 2010	Technical Operations	Timothy Dilks
2nd Qtr 2009	15 May 2010	The Best of Loughborough: Highlights from the Conference on Systems Engineering Research and SEANET	Roy Kalawsky and Ricardo Valerdi
3rd Qtr 2010	8 Aug 2010*	2010 International Symposium Coverage: Chicago, Illinois, USA	Jack Stein
4th Qtr 2010	15 Oct 2010	Systems Development from Deep Sea to Deep Space:	Mike O'Driscoll and Sam Seymour
1st Qtr 2011	15 Feb 2011	Knowledge Management for Systems Engineering**	Regina Griego

* Submission deadline moves according to International Symposium date

** Please contact the theme editor by 21 May 2010 to propose a theme article.

INSIGHT

International Council on Systems Engineering
7670 Opportunity Road, Suite 220
San Diego, CA 92111-2222

Presort Std
U.S. Postage
PAID
Seattle, WA
Permit #4

INSIGHT