What's Model Driven Engineering (MDE) and How Can it Impact Process, People, Tools and Productivity

Mark R. Blackburn, Ph.D.
Systems and Software Consortium, Inc.

Abstract

This paper is associated with a four-part Webinar Series presented in 2008 that discusses Model Driven Engineering related topics that are now relevant to System and Software Consortium (SSCI) members. Some SSCI members were early adopters of Model-Based approaches, and SSCI has expertise and lessons learned from working with these early adopters dating back to 1995. Customers are now asking SSCI members to use MDE approaches and tools. However, for some SSCI members there is a large knowledge gap and they don't know how best to adopt MDE or even get started. There are concerns related to modeling techniques, organizational process changes, tools, and project estimation and cost. In addition, there are good and bad ways to integrate model-developed components with those developed using other more traditional approaches.

A key objective of this paper and Webinar series is to clarify what types of models can be used to support lifecycle activities, so that members can better understand where they need to invest to achieve immediate cost saving or long-term benefits. Therefore, the focus of MDE from the perspective of this paper is what information can be derived from model and associated modeling tools that contributes to the development, verification, evolution, maintenance and management of the software systems that our members develop.

Contents

This paper, like the Webinar series, is a trial. The paper describes information presented at the Webinar series sessions. During the Webinar, tools were mentioned, but these are provided as examples. The Consortium is not recommending any particular tool or technology, but rather seeks to make members aware of the capabilities of different tools. [1]

What Do You Think?

The author is interested in any comments you have regarding the use of the Webinar Series and coverage of Model Driven Engineering, especially concerning the following topics:

- Was the Webinar Series a useful mechanism to disseminate information or do you prefer other means such as classroom training?
- The organization of this whitepaper is aligned with the Webinar Sessions – do you have any other recommendations that would improve the delivery of this information?
- What other Model Driven Engineering topics would you like covered in a future Webinar, paper, or other delivery mechanism?

Please send your response to these questions and any other comments to ask-ssci@systemsandsoftware.org, referencing the title of this paper in the subject line.

1

"Over time, code gets complicated, and you want to be agile and change it," he said. "This (modeling) is definitely an area that's open for improvement." – Bill Gates Final Keynote

June 3rd 2008
http://www.informationweek.com/news/software/development/showArticle.jhtml?articleID=208401781

Microsoft may not have been considered a major player in modeling for many years, at least what might be considered the formative years of modeling, but Gates in his final keynote address talked about how modeling will transform software development for Microsoft customers, especially in how the software development lifecycle is managed.

Introduction

Some System and Software Consortium (SSCI) members were early adopters of model-based approaches, and SSCI has expertise and lessons learned from working with these early adopters dating back to 1995. Customers are now asking SSCI members to use Model Driven Engineering (MDE) approaches and tools. However, for some SSCI members there is a large knowledge gap and they don't know how best to adopt MDE or even get started. There are concerns related to modeling standards, modeling techniques, organizational process changes, modeling tools, and project estimation and cost. In addition, there are good and bad ways to integrate model-developed components with those developed using other more traditional approaches.

The evolution of modeling standards[2] that cover enterprise (e.g., DoDAF, MoDAF), systems (e.g., SysML, MARTE), software (e.g., UML) and hardware provides a common basis for the development of interoperable modeling tools. Tool integration standards related to work by the OMG, INCOSE, and AP233, as well as Eclipse for open source development have resulted in many tools to cover various aspects related to modeling, simulation, code and document generation, and analysis. However, model and tool integration is still challenging, and member companies want to understand more about the specifics of modeling tools and applicability to specific domains and types of system (e.g., embedded versus IT/enterprise).

Even with the influence and availability of model-based tools within the university systems where new graduates have the modeling skills, domain knowledge and process guidelines are required for developing complex systems. Open source and commercially available tools are maturing, but MDE may not be right for all projects. For SSCI Member organizations, to minimize program risk, it is important for new MDE users to understand tool limitations and issues, but it is even more important to determine how specific tools and modeling approaches can be aligned with existing processes and the skills of people.

[2] See Session 4 for more information on Department of Defense Architectural Framework (DoDAF), UK Ministry of Defence Architectural Framework (MoDAF), System Modeling Language (SysML), Modeling and Analysis of Real Time and Embedded Systems (MARTE), and Unified Modeling Language (UML).

Finally, some members have stated that their modeling efforts have not provided the significant results that they have expected. Too often the selected modeling approach has resulted in models that were barely more than cartoons, often related to models that represent only structural system aspects. Models must minimally represent structural and behavioral system aspects in order to automate code generation, but modeling approaches to support the system interactions such as timing, scheduling, and resource allocation are being integrated with MDE approaches and tools. Ongoing research is providing insight into complex problems such as parallel computing and concurrency. The results will lead to improved model-based code generation and model analysis required to provide greater assurance of the dependability of today's complex distributed systems.

MDE Webinar Series and Organization of Paper

This paper provides supporting documentation for the four-part Webinar series. Figure 1 provides a perspective on the Webinar sessions content. Session 1 provided an overview of how modeling types, process from a return on investment (ROI) and tool technology are related. In addition, session 1 introduced modeling concepts, terminology, approaches, process and organization implications, benefits, limitations, risks, and the state of the tools. Session 2 used a modeling maturity model to discuss ROI and key practices, including modeling guidelines and recommendations useful for project proposals and model adoption. Session 3 described tool capabilities, and the importance of tool integration that is needed to achieve full lifecycle support. Session 4 included more advanced topics such model integration and challenges from various engineering domains (e.g., software, hardware, mechanical, etc.) up through the system and system-of-system (enterprise) levels, while making some predictions on where MDE is going in the future.

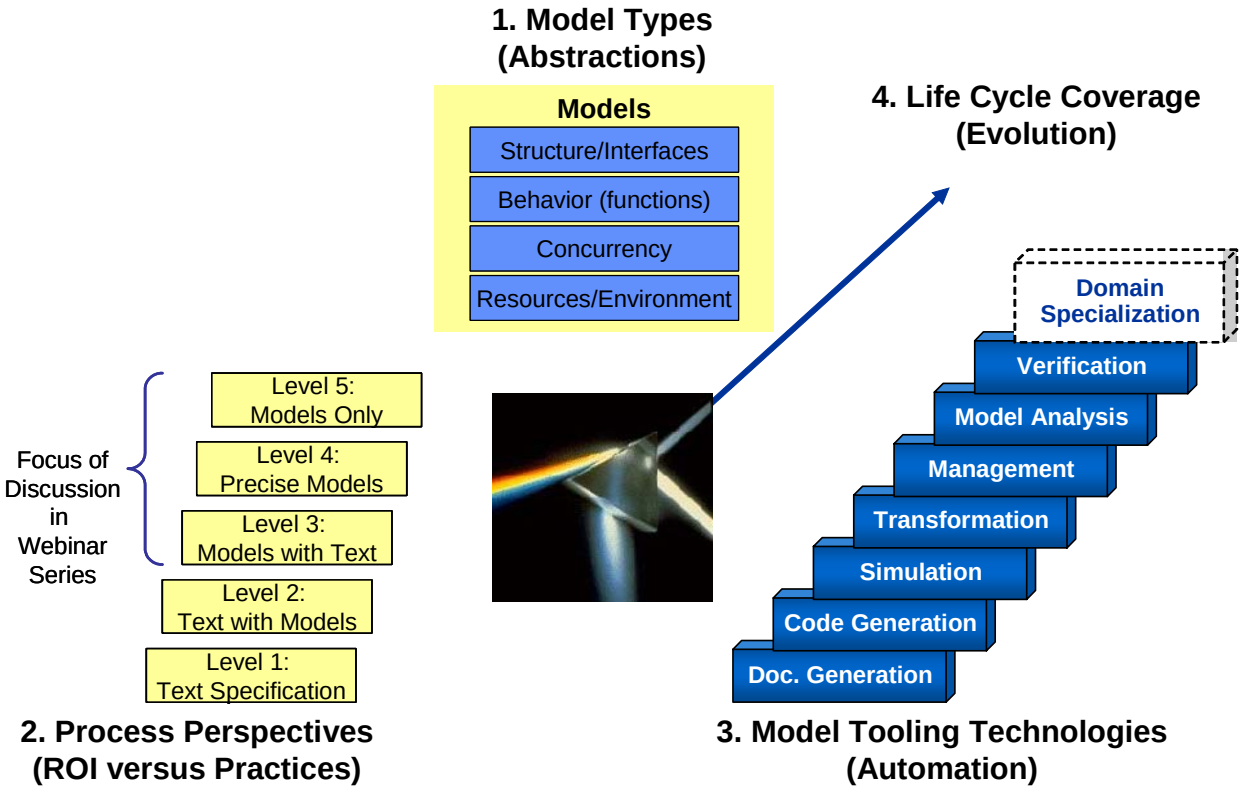


Figure 1. Perspective on Model Driven Engineering Presentation

The webinar sessions topics were:

- Session 1: What's MDE and Why Should I Care?
- Session 2: How Does MDE Impact My Process?
- Session 3: What's Happening with MDE Tools?
- Session 4: What's Next to Come with MDE?

The paper presents the webinar material roughly in the same order as the information was presented in the Webinar sessions. Many of the key topics presented in Session 1 are further explained in greater detail in one or more of the other sessions. The presentations and podcasts from the Webinar can be obtained from the Members Only section of the SSCI website www.systemsandsoftware.org under Training, Event Archive, and Webinars menu.

Who Should Read This Document?

This paper, especially Sessions 1 & 2, is applicable to most of the SSCI member personnel involved in software system engineering including:

- Directors and managers
- Capture managers & teams
- Program and project leads
- System and software engineers: architects, developers, integration and test
- Process developers
- Customers

Sessions 3 & 4 provide more technical information related to lifecycle coverage, evolving standards, and leading-edge tools. These sessions should provide all readers a high-level perspective on tools and technology, but the final few subsection will probably be more beneficial to technologists.

Session 1: What's MDE and Why Should I Care?

This session provides a broad overview of areas related to MDE and provides high-level information and key technical details needed to understand the difference between models, modeling, and model driven engineering.

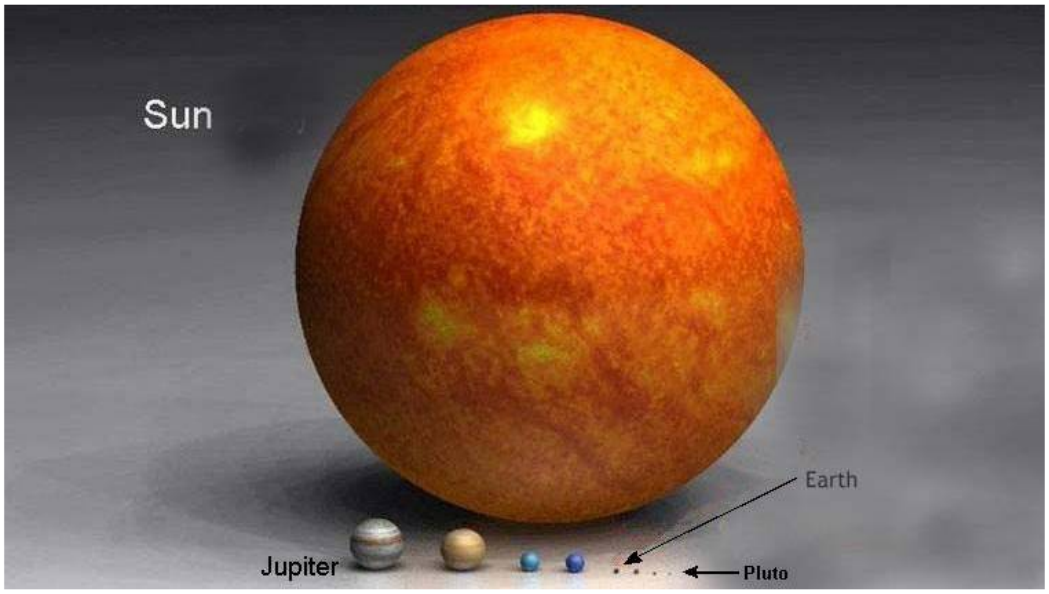
What's Modeling About?

One key aspect of models and modeling is abstraction, which supports communication through different views with various levels of details. Details of importance can be emphasized while other details are not described. For example, a mobile of the solar system as shown in Figure 2

shows the number of planets and might show the relative position of the planets, but it does not accurately show the plant's size or distance from sun. Figure 3 provides a different perspective on the planets of the solar system and emphasizes the relative size of the planets. To get an accurate perspective of a problem or solution often requires several views with some type of formal description of the relationship between the views. For example, the distance from the sun to each planet needs to be described using consistent units (e.g., miles).

www.thisisauto.com/.../wa07005i/l/ID_mobilel.jpg

Figure 2. Mobile

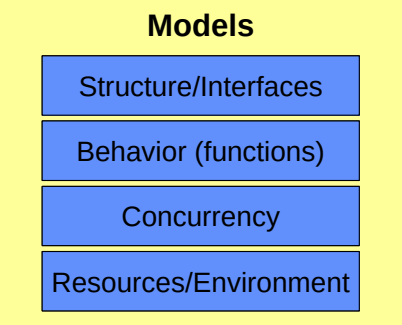


http://elronsviewfromtheedge.wordpress.com/2006/08/23/and-you-thought-size-mattered/

Figure 3. Relative Size of Planets

What's Model Driven Engineering About?

MDE is about the use of relevant **abstractions** that help people focus on key details of a complex problem or solution combined with **automation** to support the analysis of both the problem and solution, along with the mechanism for combining the information collected from the various abstractions to construct a system correctly. Some of the key abstractions can be categorized into types, such as:



- Structure – systems, subsystems, components, modules, classes, and interfaces (inputs and outputs)
- Behavior (functionality)
- Timing (concurrency, interaction)
- Resources (environment)
- Metamodels (models about models)

Some of these abstraction concepts have existed and evolved with programming languages, but within a programming language the combination of these views may be lumped or tangled together (e.g., spaghetti code). Details such as the protocols for communicating, concurrency concepts such as threads, and specialized interfaces to hardware might be combined with domain-specific functional details such as financial computations for tax processing, control law processing for aircraft, or weapon delivery rules. Through good development practices programs can be better structured and layered, but models provide a means of systematically separating these views, because certain types of models are constrained to permit only certain types of information. MDE automation relies on automated means for analyzing the views, deriving information from one-or-more views, and ultimately pulling sets of views together correctly to produce some type of computationally-based system.

Historical Context of Modeling

Models and modeling are not new. Without going too far back in history, third generation languages such as C raised the level of abstract over assembly language. Computer-Aided Software Engineering (CASE) tools provided other abstractions with some tooling, but the vision of full automatically generated software was more challenging than could be addressed with those types of models. The Unified Modeling Language (UML) attempts to unify the best of all modeling practices with standardized views and diagrams, but the goal of UML-to-code has some challenges that are discussed in Session 3. For specific domains related to control laws, the reality of model-to-code has been realized. For example, at a Consortium member event in 1996 members discussed the realization of transforming models to code as reflected in Figure 4. Models represent requirements or design information independent of language, platform, and architecture. Models are translated into implementations using tailorable code generation to specific architectures and languages. Configuration parameters are input to the code generators to specify platform details needed for the target code. The control law software for the F16, F22, and F35 (JSF) aircraft has been produced using tools such as the Mathworks Simulink and National Instruments MATRIXx dating back to the 1990s. The models are a valuable asset that continues to evolve even though the underlying platform continually changes. Models are key intellectual assets of the company.

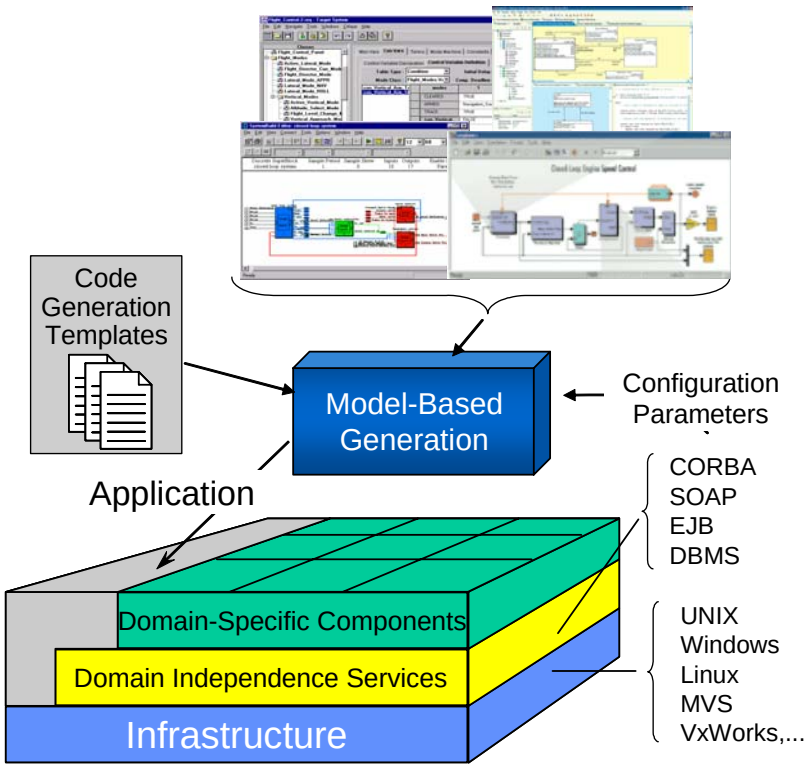


Figure 4. Model Based Code Generation

Next Disruptive Technology

This form of model-based generation (the term used in 1996) was focused on code generation. In a Raytheon article in 2003, MDE was called the ***Next Disruptive Technology*** that would have impact in 5 to 10 years. The article that contained quotes from leaders in the field stated:

> The disruptive technology to enable this development is predictive mathematical modeling of software systems. Such modeling will lead, for example, to model-based generative programming, producing executable code that is never touched by human hands, even—especially—in maintenance phases. A true "disruptive" technology is one that is adopted on its inherent virtues even if its payoff is not yet realized in application or commoditization. It provides a temporary step backward in order to facilitate greater gains later. Mathematical model-based development will cost the industry in retooling and retraining, but it will be to the software industry what the assembly line was to the automobile industry. In related ways, model-based process compliance will show similarly tangible returns, which may not be realized until the V&V stage. ***If a disciplined model-based development process is adhered to for the entire development lifecycle (there's another step backwards), V&V tools will close the development loop, doubling as requirements management tools, and the development cycle will become a living dynamic process with stability and correctness properties of its own.***

In working with SSCI organizations, some users are often focused on code generation. It is important to note in the bolded text that this vision includes not only code generation, but also model analysis, verification and validation, requirements management, and configuration (or model) management. A conceptual view of an environment to support this vision is shown in Figure 5. Conceptually, the domain experts communicate with modelers (or ideally produce

models themselves). Model analysis, such as model checking, proof of properties (e.g., safety, timing) can be validated early. Different transformation mechanisms produce code for various platforms and languages, with fully automated testing to ensure that the generated code operates properly in the different target environments, with full requirement-to-test traceability, and configuration management based directly on the source model. There are many process steps that need to be addressed and modeling technologies that need to be advanced and integrated to support this view. Additional perspectives are provided throughout the paper.

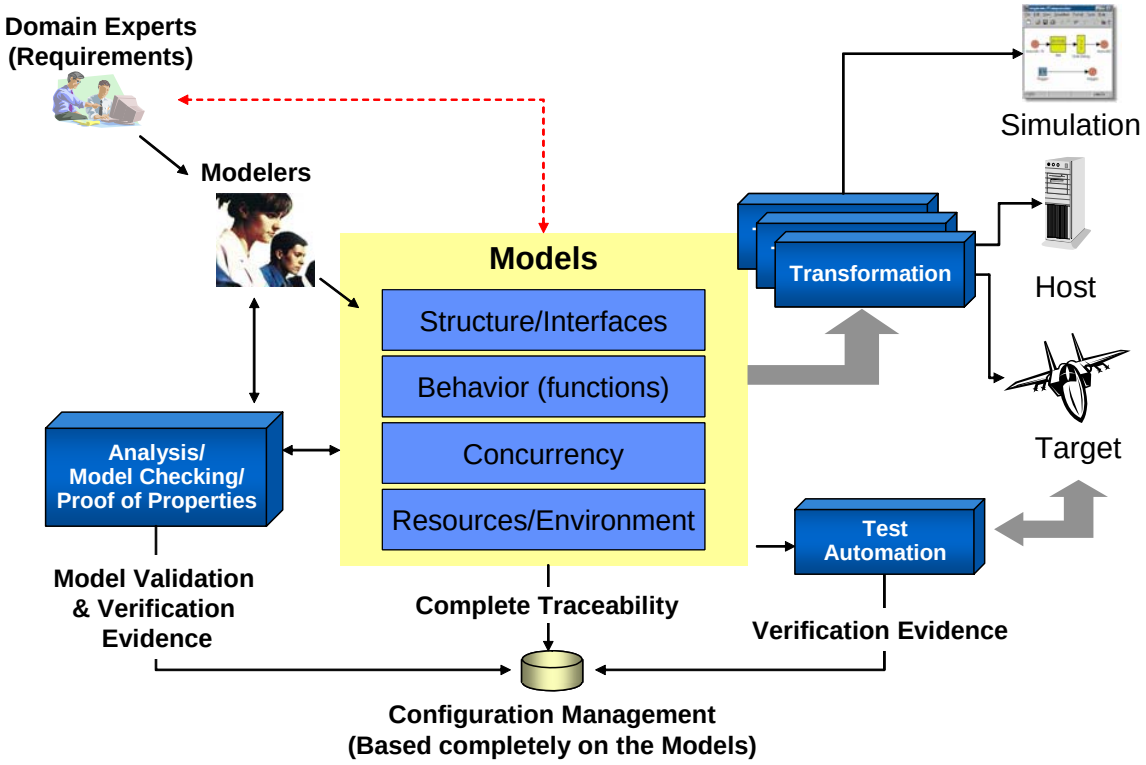


Figure 5. “Near” Idealized View of MDE

Domain-Specific Code Generation of Functional Threads

In the 1990s, several SSCI member companies were using automatic code generation from MATRIXx models as reflected in Figure 6. Most now use Simulink for these same types of applications. An important point to note is that the models were used to produce some of the code from the system. Too often first time users are often misled into thinking that model-based tools can be used efficiently to generate all of the code in the system. Some of the code generators might be good at producing code for some aspects of the system, but not good at other aspects of the system such as real time control. For example, as reflected by Figure 6 code internal to the threads represents control system computational type code, and this code is embedded and wrapped by manually produced code that is used for controlling the various tasks of the system. Members need to be more aware of what types of code can be produced and used. These characteristics and the different levels of model-based capabilities are discussed in terms of model maturity.

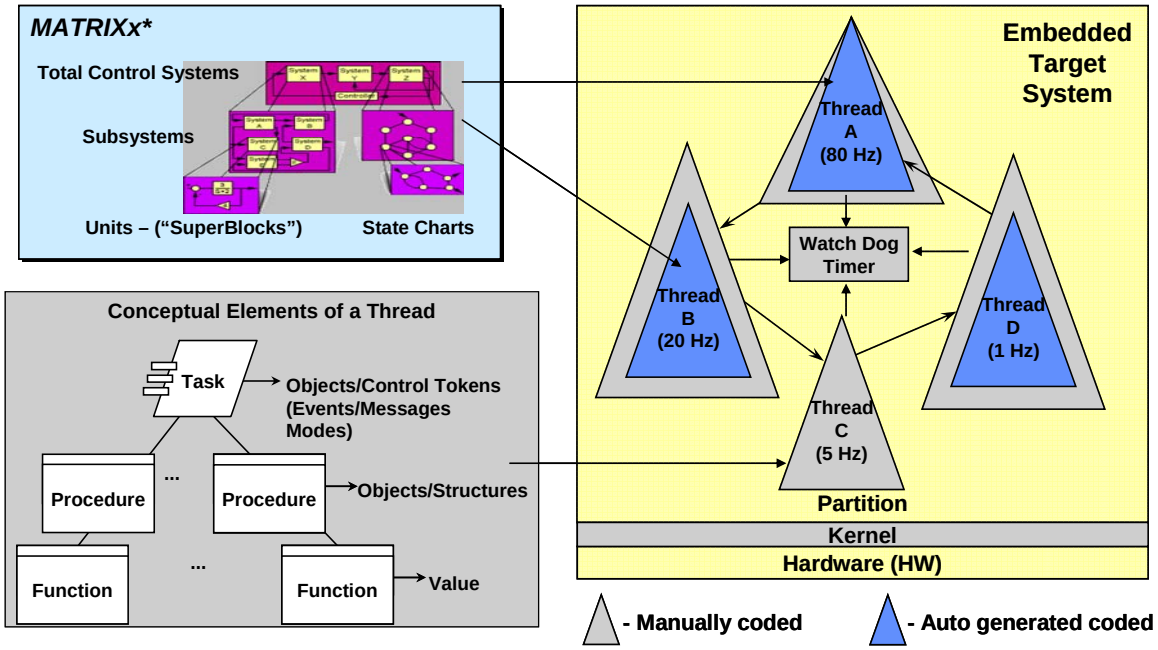
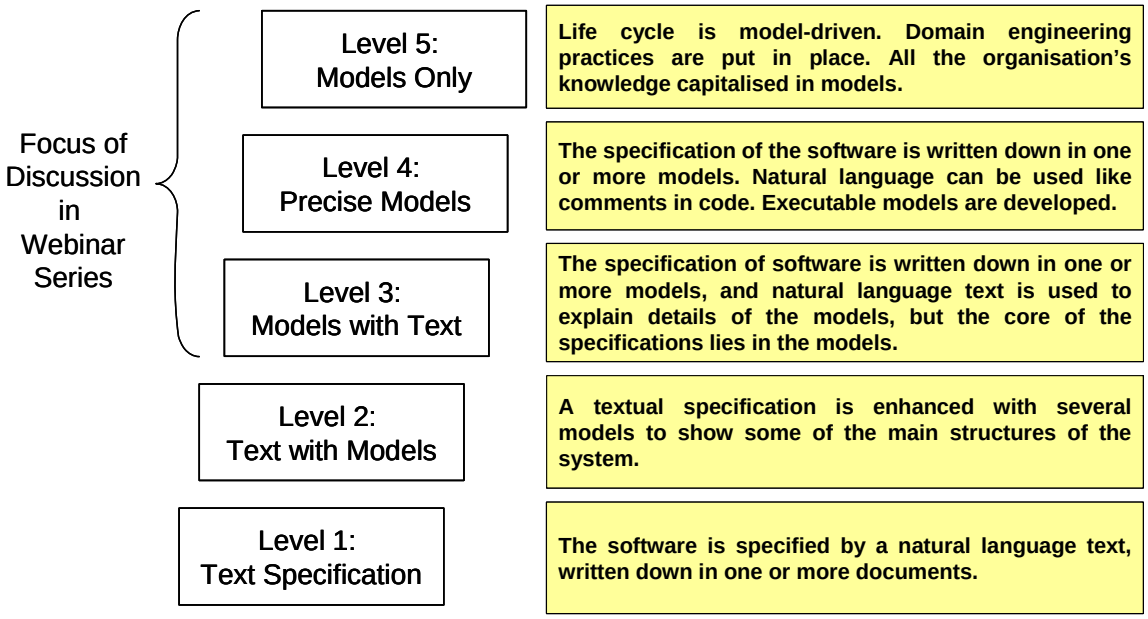


Figure 6. Control System Code Generation

MDE Maturity Model

There are numerous proposals for a Modeling Maturity Model (MMM) that follows in the spirit of the Capability Maturity Model Integrated (CMMI®) and Software Capability Maturity Model (SW-CMM®) for discussing model maturity. The levels shown in the model, especially the higher levels, rely more heavily on formalized models that provide some form of tool related automation and methodologies.



Sources: Jan Aagedal, SINTEF, September 2006, Anneke Kleppe and Jos Warmer in their book MDA Explained Addison-Wesley

Figure 7. Modeling Maturity Model

The types of models and associated tool automation often directly relate to the completeness of the artifacts required to produce a dependable deployable system. For example, the modeling approach associated with Figure 6, based on Simulink or MATRIXx models are often highly functional and specified using modeling notations (aka modeling constructs) that are known to control systems domain specialists. This type of modeling may not apply to other types of systems in other application domains such as financial or information technology. Therefore other types of model notations or views might be required. An example in the data modeling domain is Erwin, which has been used for years to support database design. Organizations that develop using an object-oriented (OO) method and language might be new to modeling or considering adoption. The UML is now supported by tools and modeling approaches that might be more applicable to developers that have been using OO design methods.

Too often members have thought that simply using models will result in higher ROI, but unfortunately that is not always the case. There are a number of limitation and issues with using UML and the associated tools for automatic code generation. Session 2 discusses more about modeling maturity, and in particular discusses how to use concepts like the MMM from an ROI perspective and from a process practices point of view that defines key modeling practices required to maximize the ROI.

Unified Modeling Language

UML is a general purpose modeling language, and attempts to unify many modeling practices. It allows almost everything to be modeled, and because of that it has grown to be extremely large and complex, including an 800 page specification. UML provides a set of diagrams to depict software structures graphically, as shown in Figure 8. Diagrams are developed as separate entities that express different aspects of software, however UML cannot fully define the relationships between diagrams and detailed behavior is difficult to define in UML. Consistency across diagrams is largely left to be resolved by the designer, and without detailed behavior code generation can be limited to structural aspects of the code. As a result tools often combine structural aspects of UML models with manually developed code to specify the behavior. This can result in the need for synchronization between manually developed code and the models.

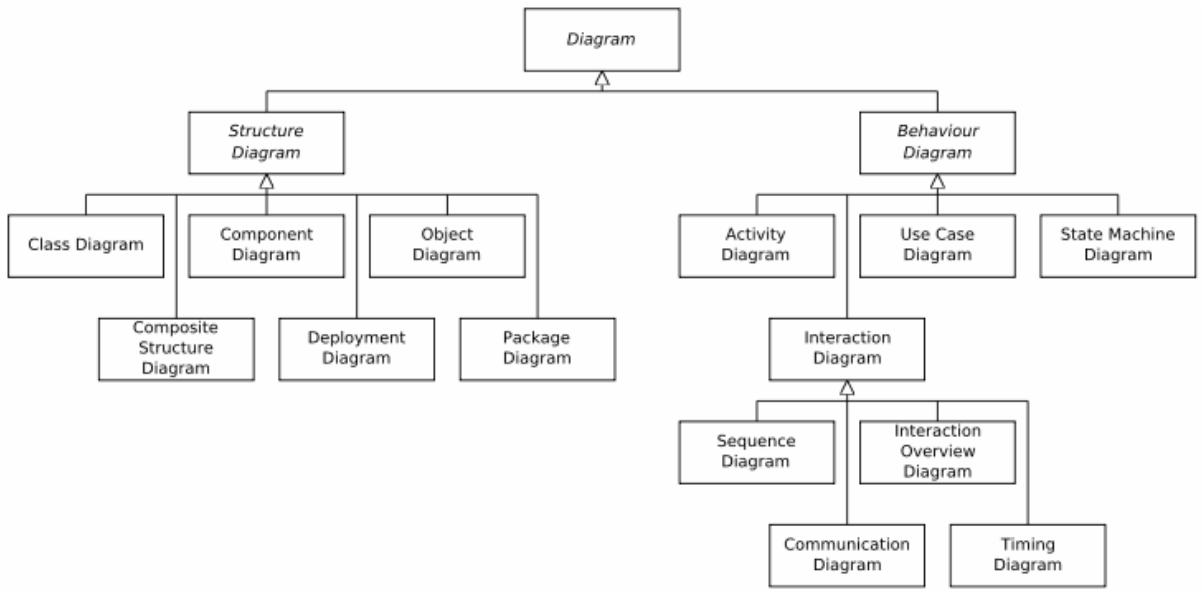


Figure 8. UML Diagrams

There are other approaches for specifying behavior such as the use of action languages that are used with UML diagrams in place of code. Domain Specific Languages (DSLs) are also emerging, because UML cannot provide inherent support for every application domain. DSLs are more precise, more constrained, with clear semantics. Tool support makes DSLs a reality, and metamodels are often used to define these more specialized languages. UML is sometimes used for defining a DSL. DSLs are not new, for example Backus Naur Form (BNF or EBNF) with tools yacc and lex is a good example of a DSL and associated set of tools that have been around for decades. Simulink might be one of the best known DSLs. DSLs and metamodels are discussed in greater detail in Sessions 3 and 4.

Confusing Terminology

The specialization of modeling approaches, standardization, and tool support has led to a number of different terms that relate to MDE. The “model driven engineering” or MDE is not currently trademarked and therefore it is used to characterize the general set of model-based practices. There are many related MDE approaches, and the following provides a non-exhaustive list of a few commonly used terms:

- MDA®: Model Driven Architecture®
- MDD™: Model Driven Development
- MDSD: Model Driven Software Development
- MDSE: Model Driven Software Engineering
- MIC: Model Integrated Computing
- DSL: Domain Specific Languages
- Software Factories
- MBT: Model Based Testing

There are different tool companies that support these approaches too. For example, MDA is an OMG standard, and many companies were involved with the formalization of this standard. In 2008 there are about 60 companies listed on the OMG website that provide MDA-related tools or services. MDA is based on a set of OMG standards such as:

- UML – Unified Modeling Language
- MOF - Meta Object Facility
- XMI – XML Metadata Interchange (XMI®)
- OCL – Object Constraint Language
- CWM - Common Warehouse Metamodel

MDA proposes two separate parts of specification:

- Platform Independent Model (PIM) are used to specify structure and behavior related to a specific domain or application
- Platform Specific Model (PSM) is a specification of implementation of the functionality on a specific technology platform.

Transformations are iteratively used to transform PIMs into PSMs and finally deliver running systems. More details are provided on tools and transformation related to MDA in Sessions 3 and 4.

Session 2: How Does MDE Impact My Process?

SSCI members that consider adoption of MDE often have higher levels of maturity as defined by the CMMI® and SW-CMM® guidelines. Although a high level of process maturity is not required to get started with MDE, those concerned with process maturity need to understand the process impacts before they start using MDE on programs. These considerations include not only the typical development-related processes and artifacts, but also important information that can go into program proposals that identify program milestones and deliverables, documentation, configuration management, as well as the need for education and training requirements of both the company team and potentially the customer. Some of these process related issues are not well understood by the proposal teams.

For organizations that use a more agile process, MDE is often iterative in nature with continuous builds and testing supported by automatic code generation and systematic test generation. An MDE approach can complement an agile process while providing significant productivity benefits with added rigor to support complex system development. This session includes information on how agile teams most often use modeling.

This section of the paper includes:

- Process-related considerations impacted by the use of MDE
- Potential changes to lifecycle schedule and deliverables that can impact proposals
- How and why to conduct pilot projects to reduce the risk of adoption
- The need incorporate modeling standards
- The importance of modeling reviews
- Project types that might not be appropriate for MDE

Model Maturity: ROI versus Key Practices

Figure 9 adds a few examples of modeling approaches that are associated with the MMM levels initially shown in Figure 7 to explain capabilities versus ROI. Consider the following examples:

- The MATRIXx example shown in Figure 6 identified as MATRIXx 1996 in Figure 9 might be considered about a Level 4, because precise models are used to produce the code.
- The JSF Simulink 2002-2008 example identifies a modeling approach that might be a slightly higher Level 4, because Simulink models are more expressive in 2008 than they were in 1996.
- UML-oriented modeling tools and methods vary significantly and some of the earlier versions (e.g., UML 1 and UML2) used models integrated with text or code to produce the code and the ROI can vary significantly.
- The xUML (i.e., called executable UML) is a UML-variant supported by a few tool vendors and the tools combine UML diagrams with a programming language independent action language to capture the behavior of a model formally; these tools often produce code directly from the models and the action language can be targeted to different platforms thus providing higher ROI than UML based tools that use code to specify the functional behavior.

- The survey data provided in Figure 10 suggests that agile teams often do use models, and because the whiteboard sketches are considered valuable as reflected in Figure 11, those types of models help structure the communication of an agile team during early phases of development. Although these whiteboard models are informal, they add value as reflected in Figure 11. It is not necessary to have a high level of modeling maturity to get ROI.
- As shown in the upper right of Figure 9, full model-based code generation approach with model analysis and automatic test generation, will likely be considered mature and provide high ROI, although such methods and tools probably do not exist. For example, models of concurrency that capture the distributed characteristics of the system would be needed, and this is still a topic of research discussed in Session 4.
- DSL and MDA approaches will cover the spectrum; although many DSL are focused on code generation or other types of formal analysis, some system-level DSL like SysML might only formalize the descriptions of the system structure.

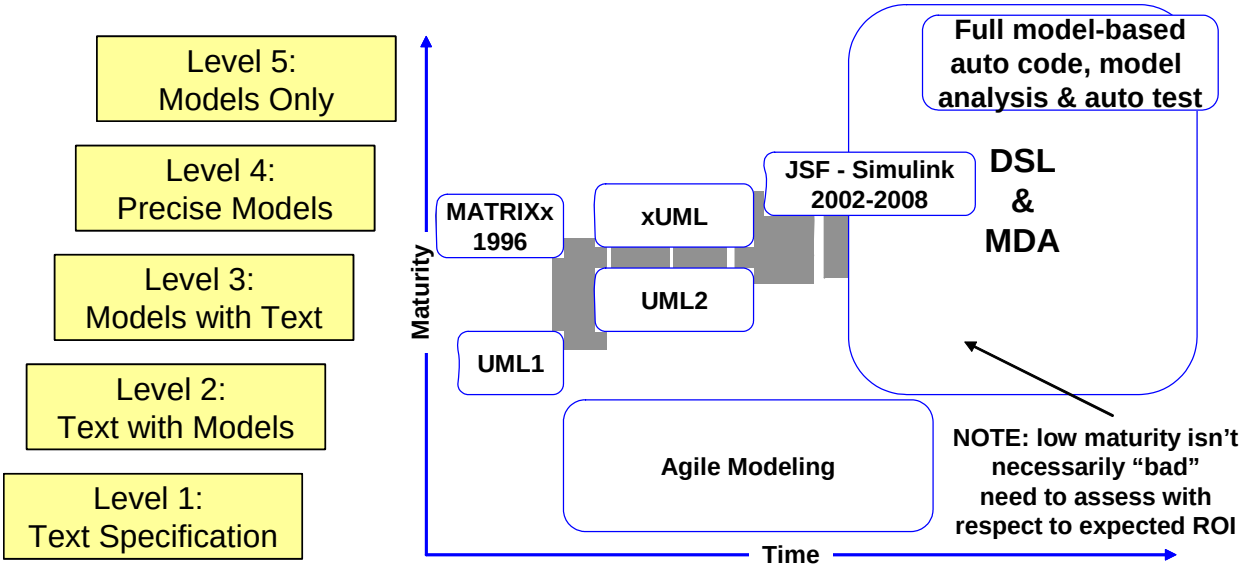


Figure 9. Model Maturity versus Time

Low maturity is not necessarily bad, but what members need to know is that if they are using a lower maturity modeling approach, they need to be sure that they are not making claims in proposals, contracts or project estimates that assume a high ROI. Consider the following example scenario of a project that might be using UML models supplemented by text and code.

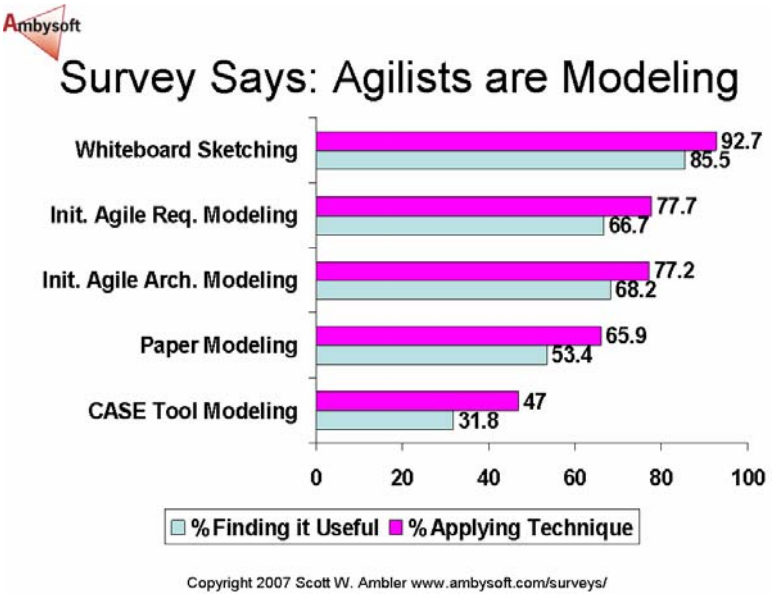


Figure 10. Agile Survey on Use of Modeling

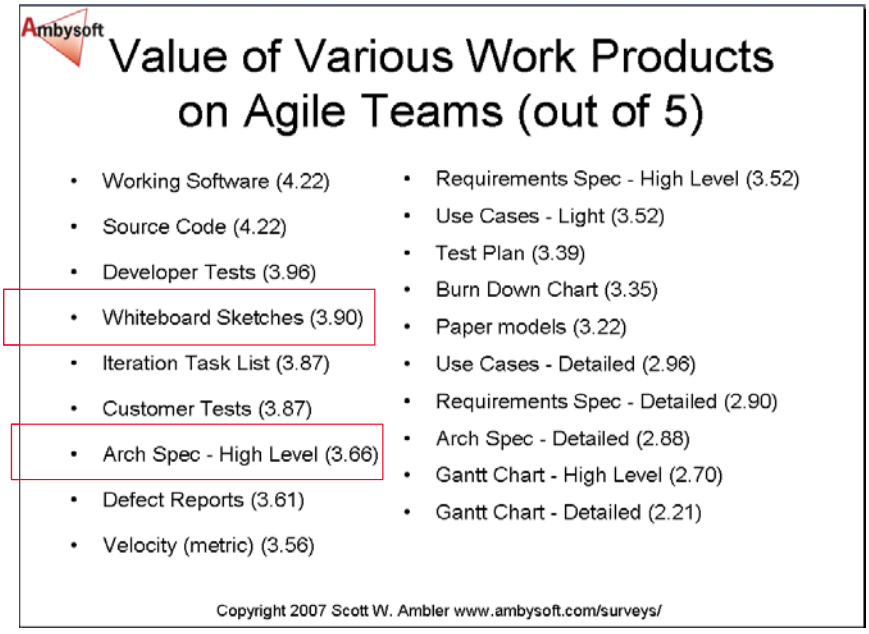


Figure 11. Value of Agile Work Products

Example Project Scenario

As discussed in Session 1, some organizations that may have tried MDE, and in particular some type of UML-based modeling process and tool might have lived through the following scenario.

1. Project uses models that are initially a proper reflection of software being built
   - Models show structure of actual code (e.g., class diagram)
   - Provide documentation for detailed design and code

2. Transition of model to code is done mostly manually. The models represent structure of code through class diagrams, but the behavior is not sufficiently formal and thus a programming language is used to complete the behavioral parts of the model.

3. With the passage of time, and continual changes, the models do not reflect the actual code anymore. The code is updated until the customer is satisfied or the code is changed when new requirement or bug fixes are required.

4. The code is now the product, because keeping the models up-to-date is often considered to be unimportant and too time consuming.

5. The models that were once perfect as code documentation are now useless.

Continuing with the theme of the following example, Figure 12, based on a diagram originally created by John Daniels[3], reflects on some of the issues that often occur with UML modeling where the behavior is described in code. This particular process of synchronizing the code with the models is sometimes referred to as round trip engineering. As reflected in the scenario, when opposing forces such as time and schedule pressures begin to impact a project, the focus stays on the code and not the models. Armed with knowledge of these common pitfalls, there are a few recommendations and guidelines that potential new users should consider during the process of planning for MDE adoption.

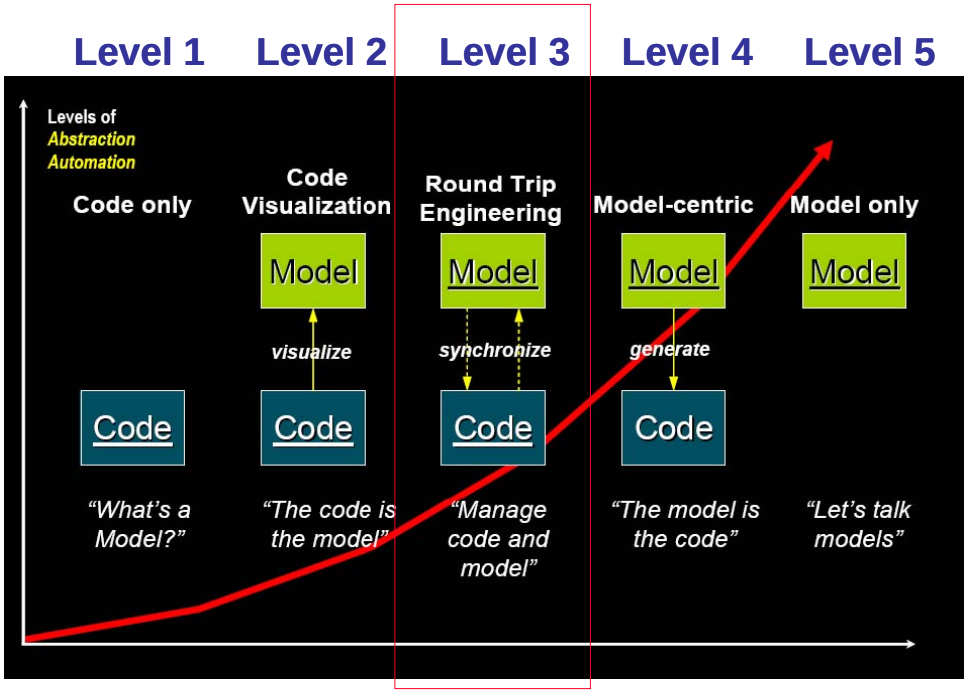


Figure 12. Another Perspective on Model Levels

[3] MODELWARE – 511731 – D2.9 MDD TCM Guidelines – Revision 1.5

Process Guidance - Tools and Methods

MDE tools and standards are evolving, but methods and usage standards are not inherent in the tools, and organizations must define methods, standards and tool usage that align with their project and organizational drivers. In addition, they must align their ROI expectations when developing proposals and cost and schedule estimates. There are a number of effective ways for adopting and preparing for the use of MDE. A non-exhaustive list of guidelines follows:

- To start, use pilot projects to thoroughly understand the MDE tools and methods.

  Begin with some type of system application, or subsystem that is well understood so that process comparisons can be made. The comparison may be subjective, so that is why it is important to use some type of existing system, ideally one that was recently completed.

- Think about a modeling methodology that aligns with existing processes.

  Many of the tools do not prescribe a particular method, but based on the ways they produce code, or documentation, may impose some process constraints that may not align with your organizational processes. Consider tool alternatives and select a tool or tool that fits in with process and technology change objective and constraints of your project and organization.

- Define methods for your project that constrain the tools to meet your specific needs. Advanced users can create or extend tools that use languages such as the Object Constraint Language (OCL) that checks to ensure guidelines on developed models are satisfied.
- Plan for integration of system/hardware modeling with software modeling tools.

  Understand the interfaces between the different hardware and software modeling tools and methods to ensure that inconsistencies that can cause late integration problems are addressed before integration.

- Define lifecycle objectives based on understanding the tool capabilities and current state of organizational capabilities.

  For example, for a first time user it might be reasonable to assume that the modeling will be used for structural code generation and full document generation from models.

- Understand auto-code generation capabilities.
  - Understand what aspects of structural and behavioral models contribute to code.
  - Identify code generation parameters or template, because different parameters can significantly impact the way code is produced (e.g., code may be optimized for real-time performance as opposed to memory space efficiency).
  - Identify modeling constructs that produce “good” code, and add that information to methods guidelines and standards.
  - Create baseline models for re-validating code generator from version-to-version of the tool, and update and maintain those validation models with each new tool version.
- Understand the run-time environment that might be assumed by the code generator.

  - Assess the impacts of run-time overhead, and identify if or how the run-time code can be configured, or determine how to produce code that does not rely on a run-time environment. As reflected by Figure 6, it might be necessary to wrap generated code with run-time control such as tasking or threads that has been created through manual code development.

  - Plan to address concurrency issues external to code generation. See more in Session 4 on the subject of concurrency.

- Determine how model artifacts support required document generation and assembly.

  - The organization of the modeling artifacts can impact document generation. Configuration management of the model can also impact the document generation.

  - Establish how requirement-to-test traceability is documented, and understand how links must be created through the model views and associated model artifacts.

  - Ensure that the customers understand the new types of documents and deliverables.

  - Take into consideration requirements for integrating system, software, and hardware documentation deliverables. For example, how are the interface control documents related to the models?

- Understand how testing is going to be performed.

  Evolution and maintenance can consume seventy percent of the product development lifecycle and testing is often far more critical once the initial releases of the system have been developed and deployed, however testing-oriented support from models is not often addressed during the modeling and tool selection process.

  - During the pilot project, determine if the selected modeling tools provide support for automatic test generation.

  - Determine if the auto-code generation or run-time environment constrains how testing and test coverage is accomplished. Auto-code generation may change the name of model variables in the auto-generated code. Determine if there is information that maps model names to test interfaces.

  - Add guidelines to ensure that the modeling methodology enforces design for testability.

  - If applicable, determine and plan for the tools and methods for documenting test coverage.

Process Methods and Configuration Management

Many of the activities discussed in terms of methods and the associated tooling should be formalized and incorporated into project-specific methods and review procedures.

- Establish standards and reviews for use of the tools.

  Describe modeling constructs that are permitted for use, as well as those that should not be used, because they can result in poor quality code (e.g., performance is too slow, not designed for failsafe behavior of safety critical applications). Ensure the guidelines are created and review processes are used. Too often organizations that understand the importance of code reviews do not follow those same types of guidelines for modeling.

The guidelines should also identify:

  - Code generation parameters and templates, for example, variations in the parameters can have a significant impact on code size or code speed.
  - Naming conventions
  - Project structure and organization
- Determine the modeling artifacts that must be configuration controlled.
  - Model management and model merging may not be as straight forward as code-based and file-based configuration management.
- Establish guidelines for how the tools must be version controlled to ensure deliverables are reproducible from source artifacts maintained under configuration control.
- Plan a process to understand impacts of new tool versions prior to permitting the new tool results to be deployed. Tools can have significant changes from version to version and the outputs produced from one version might be different or unexpected in the next version.
- Assess the potential impact of tool chain integrations.

Proposal Impacts

Some members have stated that the proposed use of modeling has been a contributing factor to winning a proposal award. Other members are struggling to advance their modeling capabilities, because the customers are requesting the use of model approaches on programs. In either case, there are a few things that proposal team should ensure are addressed in the proposals, including:

- Ensure proposal costs and schedules align with ROI that is achievable.

  It is important that the modeling capabilities of the organization align with the expected ROI gains. Often an advanced demonstration team may be involved in the proposal process, but it is important that the development team have proper training and understanding of the guidelines and standards.
- Make sure that the tool acquisitions process has been completed and understand the implications of tool licensing.
- Ensure that milestone schedule aligns with new modeling process.

  Modeling can impact traditional milestones. Typical customer review processes that includes preliminary design review (PDR) and critical design review (CDR) are often based on paper documents. Modeling may use alternative forms. Ensure that the customer understands the information. This may require training for the customer.

  Precise models that replace traditional documentation need more details. Understand how they can be presented incrementally as they evolve rather than as a completed document.
- Prepare subcontractors if the interface between the teams is going to use models. They too must have processes and procedures in place to support usage and development from models.

- Be prepared to train customer and subcontractor on the project-specific use of models and their associated artifacts and measures.

Lifecycle Evolution and Maintenance

Modeling often starts with a new project and the early lifecycle activities such as architecture and design often get the most focus, however some of the biggest gaps in the design of the modeling process don't account for longer term concerns such as testing that can account for a significant percentage of the effort during the evolution and maintenance of a program. Therefore it is recommended that the following topics be considered as early as possible:

- Verification and validation

  Modeling practices are generally focused heavily on code production, but verification is not guaranteed and testing is still required. Models can have defects, and although review processes should be used in an attempt to find model defects, defects can be hidden in complex models. Modeling tools are advancing with additional analysis and test generation capabilities; factor these long-term needs into the pilot project and model tool evaluation process. See Session 4 for more information on model analysis and testing support.

- Integration between systems, hardware, and software

  The methodology and artifacts handoffs should be planned and coordinated. Tools for systems models may not integrate with software or hardware modeling tools. The same issues at system-of-system levels interfaces between systems can have significant mismatch, especially when different teams, disciplines, and subcontractors are involved. See Session 4 more information.

Transitional Pilots

Planned pilot projects that precede the project development are strongly encouraged, but some members have long running projects, and may be forced into updating their capabilities during a program. Transitional pilots can provide stakeholders quickly demonstrated evidence within their organization to commit to updating their process to use modeling on a scheduled deliverable. In addition, these pilots help reduce the risk of over commitment, because they can provide some insight into ROI from modeling processes and tools.

- Transition from a pilot project to a thread of an existing project
  - Select a thread that is likely to change often or have features extend it
  - The most leverage and benefits come from reusing and evolving one or more related models
- Identify the right projects for transitioning from an existing process to a new process to meet schedule
  - Select a project prior to requirement phase so that modeling can start early and help improve requirements, while providing sufficient time to collaborate with design team to improve the interfaces to support testability, which reduces risk of schedule slippage caused by the startup overhead of a new modeling process.

Session 3: What's Happening with MDE Tools?

Tools are an essential part of MDE. Tools formalize modeling information that includes structural information such as architectural elements, interfaces, behavioral information, and other system properties. The formalization permits tools to analyze, transform, trace and simulate model information, as well as synthesize and generate other artifacts such as code, tests, documentation, and reports. This section discusses the various types of tools that support MDE, and discusses commercial as well as free open source tools without promoting any particular set of tools. No one tool provides complete support for the entire lifecycle and therefore tool chains and associated standards are emerging to provide greater lifecycle coverage. This session introduces some advances such as DSLs which are discussed in more detail in Session 4.

The objective of the Webinar and this session in particular is to cover tool capabilities without recommending any particular tool or vendors. Tool and associated vendors are mentioned as examples, but this should not be taken as a recommendation by SSCI.

Tools Targeted to Specific Domains

SSCI members produce systems in many different domains, and some of the modeling tools are targeted towards embedded systems, while other are more applicable to information or enterprise systems. Therefore, when looking at the various tool technologies it is necessary to consider the application domain needs too.

Embedded systems have been developed and verified using models dating back into the 1980s. They often have more variation when it comes to hardware, which is sometimes specialized or custom made for some applications. Embedded systems often have requirements for high assurance, meaning that systematic verification is often necessary. These types of systems may also be used on safety critical applications, and may also require run-time environments that have been proven to meet safety critical needs. The software involved in controlling mechanical devices such as aircrafts or vehicles often relies on control system models such as Simulink as mentioned in Session 1.

Enterprise or IT systems may have in the past used data modeling for the database, but there is an emergence of models being used on these types of system. It’s often difficult to apply models to legacy and mainframe development, but new areas such as web and service-based systems are prime targets for model driven engineering, and some of the tool are targeted to these specific domains.

Historical Perspective of Tool Suppliers

In 2004, the SSCI Board of Directors authorized a project to report on the benefits and risks of “automatic code generation” from modeling tools. A SSCI report (SPC-2004010-MC) covered several different tools as reflected by the representative sample shown in Table 1. Many of the tools in the table exist today, but the owner of the tools may now be a different company due to acquisitions. The purpose of the report was to categorize the different types of code generation tools, and describe the benefits and risks for using and adopting the tools. Many of these tools have continued to evolve and some products provide significant lifecycle coverage. Some also promote certain methods, while others attempt to be method independent.

Table 1. Automatic Code Generation Survey

Tool Name and Company	Category	Comment
Simulink Mathworks	Behavioral	Used for control system modeling such as aerospace, avionics, automotive, and can include state chart diagrams in a model. Model analysis and test generation support
MATRIXx National Instruments	Behavioral	Used for control system modeling such as aerospace, avionics, automotive. Combines state transition diagrams. Model analysis and test generation support available from several
SCADE Esterel Technologies	Behavioral	Used for control system modeling such as aerospace, avionics, automotive, and energy. Combines state chart diagrams. Has some support for verification.
BridgePoint Project Technologies now Mentor	Translative	Executable and translatable Unified Modeling Language (xtUML), with profile that relies on a language that extends UML 2.0.
Statemate Ilogix now IBM/Telelogic	Behavioral	State machine-based, with formal action language. Used for embedded systems with support for test.
Rhapsody Ilogix now IBM/Telelogic	Structural / Elaborative	UML 2.0, with coding framework for C, C++, and Ada. Some support for test execution.
Rose XDE IBM	Structural / Elaborative	UML 2.0, coding framework-based.
Real-time Studio Professional Artisan	Structural / Elaborative	UML 2.0, coding framework-based.
TAU/Developer Telelogic now IBM/Telelogic	Structural / Elaborative	UML 2.0, coding framework-based.
VAPS eNGENUITY Technologies	Behavioral Translative (hybrid)	Virtual Applications Prototyping System (VAPS) is a tool for building data-driven, interactive, graphical user interfaces, or human-machine interfaces.

Lifecycle Tool Integration (aka Tool Chains)

There is still a need to provide greater coverage over the lifecycle, going beyond code generation and document generation. Most of the commercial and free open source tools provide varying levels of support for:

- Model development (e.g., model editors)
- Code generation
- Document/report generation
- Traceability – diagram relationships

There is greater variability and less coverage when attempting to address capabilities such as:

- Simulation/animation
- Model analysis (for example, does a model conform to a metamodel?)
- Model checking or proof (for example, does a model satisfy certain properties?)
- Test generation (verification)
- Model management (extending configuration management)
- Model transformation (to leverage other tools)
- Model integration

Tool integration through tool chains is a way to obtain greater lifecycle coverage. The remainder of this section discusses tool capabilities from the point of view of:

- Design models – often focused more on design and code generation
- Requirement models – provide ROI without requiring details to support code generation
- Metamodels – used to support development of DSL and represent rules for source models

Design Model Perspective

Design models ideally describe the behavioral specification used as the basis for code generation. Design models attempt to represents "what's in the box," and are graphical representations of the program, but often require formal text specifications too. They are the key ingredient needed for code generation or synthesis of hardware.

As reflected in Table 1, there are different types of code generation tools that support both OO, and non-OO modeling tools. Some tools do not produce code for the entire application; therefore, the following is one way to classify[4] the different code generating capabilities of the tools:

- Structural (aka Elaborative): Generates code frames and stubs

  Some code generation tools are coding frameworks. The models produce some of the code for the target application, normally the structural aspects of the code, like the modules or classes. Often, these tools provide an integrated development environment where the code can be embedded within the tool (e.g., Rhapsody, Rose). Many of the tools aligning with the classification of the MDA initiative produce skeletons and stubs of the implementation but are not executable applications without the addition of behavioral code that is developed manually.

- Translative: Generates code using translation templates

  Application-independent modeling gives users control over translating models into code. These modeling approaches often use graphics and languages formalism like the behavioral approach. The user often can tailor the translation capabilities.

- Behavioral: Generates code using models and action specifications

  For example, the code generation formalizes the semantics of each model construct, as shown in Figure 13, including determining the dataflow, control flow, data types, functional hierarchies and calling structure. Simulink/Stateflow models represent structural (interfaces) and behavioral information formally, and supports control system and state machine modeling from a graphical set of modeling constructs (e.g., see Simulink Library Browser in Figure 13). They not only support code generation, but often support simulation of the models.

[4] Bell, R. Code Generation from Object Models, http://www.embedded.com/98/9803fe3.htm, March 1998.

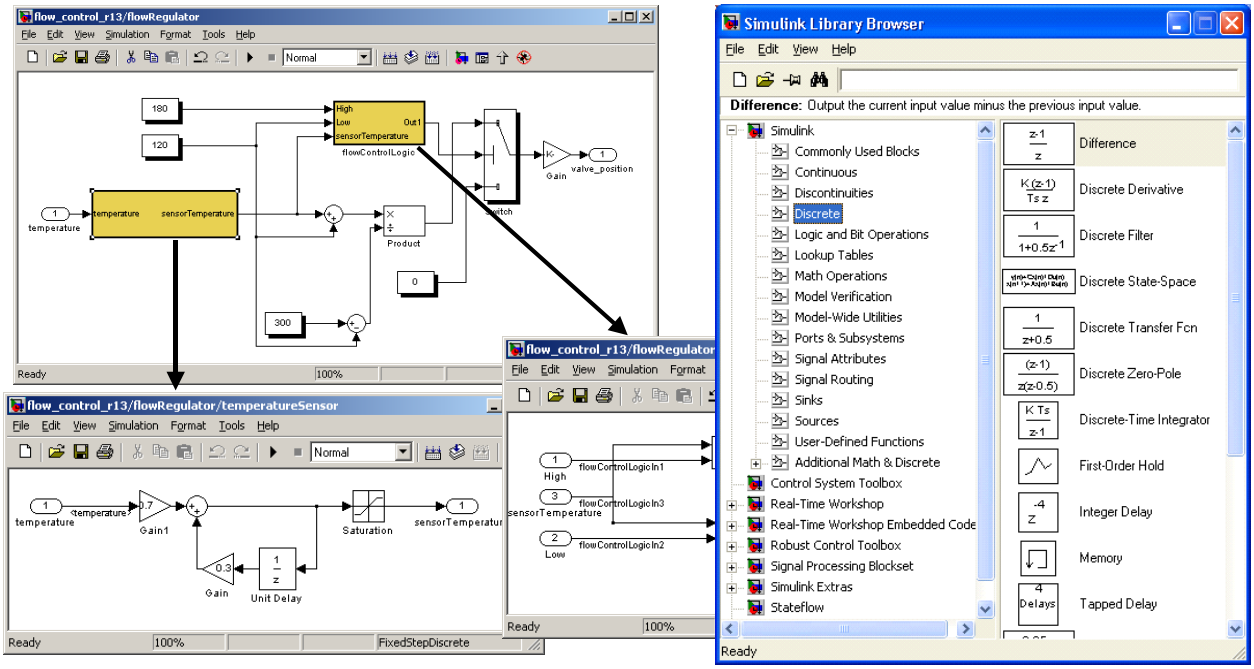


Figure 13. Behavioral Model Example

Model analysis and automatic test generation can be applied to models that can fully describe structure and behavior. These capabilities provide additional ROI, because model defects can be identified early, and testing can be performed continuously during the development, evolution, and maintenance of the design. An example is provided in Session 4.

Model Driven Architecture

MDA is an approach supported by about 60 different tool and service companies that are affiliated with the OMG[5]. There are many variants to design-based tool support and MDA compliance is not required to get ROI from MDE. Figure 14 provides a perspective on the MDA concept with emphasis on the artifacts that must be created as inputs to leverage the tools, and those output that can be produced from the tools. The most fundamental aspects of the MDA concept start with the PIM and when combined with platform specific details such as a language choice, database, and middleware preferences there is one or more transformations to a PSM that could support code generation. Conceptually, this is similar to the process that occurs with Simulink models as is discussed in Session 4, but this is also similar to what has been done in the synthesis of hardware using hardware description languages such as Verilog Hardware Description Language (VHDL). MDA is simply a concept that is associated with OMG standards such as UML and OCL. Therefore, the focus is on the different types of modeling capabilities, not specifically named tools or standards.

As discussed in Session 1, UML is a large and general standard, and platform independent models can be described in a language that is a subset of UML or related variants, and possibly a DSL. The key artifacts associated with MDE, shown in Figure 14, are discussed in the order in which they are often leveraged in the development process:

[5] Companies providing MDA type products and service as of 2008; shttp://www.omg.org/mda/committed-products.htm

1. Documentation is often an output of a modeling approach, and it should be leveraged to reduce the use of textual documentation. Documentation can be produced through some of the formal models that might be used to produce code, but other models that might be UML-based might not be precise enough to support formal model analysis, yet they provide structured information that is part of the documentation for the projects. However, using models for the sole purpose of automating document generation will probably not result in ROI.

2. Traceability is often supported by most modeling tools; tools support linking model elements together, and this can provide the basis for traceability, which should ideally cover requirements through test.

3. Code generation as discussed above in terms of the MDA concept involves the development of a PIM that can be defined based on model editors that support UML or other modeling notation such as a DSL. There are many possible platform-specific details such as a language choice, database, and middleware preferences that can be added based on the different tools and target platform. Although only one transformation is shown in Figure 14, there can be several transformations required to produce code or synthesize hardware.

4. Simulation of a model is possible if the model represents behavioral details. Simulation can be valuable for validation of the system behavior with domain experts and customers of the system.

5. Validation evidence related to checking models for certain properties (e.g., timing) is a broad subject with ongoing research as described in Session 4. Model validation includes identification of defects within the model, but a more basic set of validation evidence as reflected in Figure 14 includes the following scenario. A PIM metamodel defines the rules and constraints for a particular DSL, and model analysis tools that might use OCL can be used to verify that any specific model conforms to its metamodel. This concept is similar to the idea that a BNF grammar is used as the basis for specifying the syntax of a language (e.g., C++), and a compiler checks to see that the source code is syntactically consistent with respect to the BNF metamodel. Similarly checkers can provide assurance that a source model (e.g., PIM) conforms to that metamodel. However, model analysis capabilities usually cover other types of semantic analysis and provide model validation evidence as an output. A non-MDA model analysis example is provided in Session 4.

6. Verification evidence ensures that the target implementation satisfies the models. This too often is done manually, but model-based test automation is evolving that can provide comprehensive verification evidence demonstrating model-based test coverage of the associated implementation. Automated support for testing can provide significant ROI after the initial release of a system, as model evolution and maintenance involves significantly more testing than development. Models for testing may include information from a PIM and PSM, as well as other test related details such as input ranges, test coverage criteria, and test sequencing criteria. An example is provided in Session 4.

7. Configuration management, sometimes called model management, may not be thought of as an output of the modeling process, but it is a key element important to modeling tools. Configuration management of models can be significantly different than configuration management of code or documentation files, and therefore when planning

to select a tool for project use determine the tools that provide the best support for your organization.

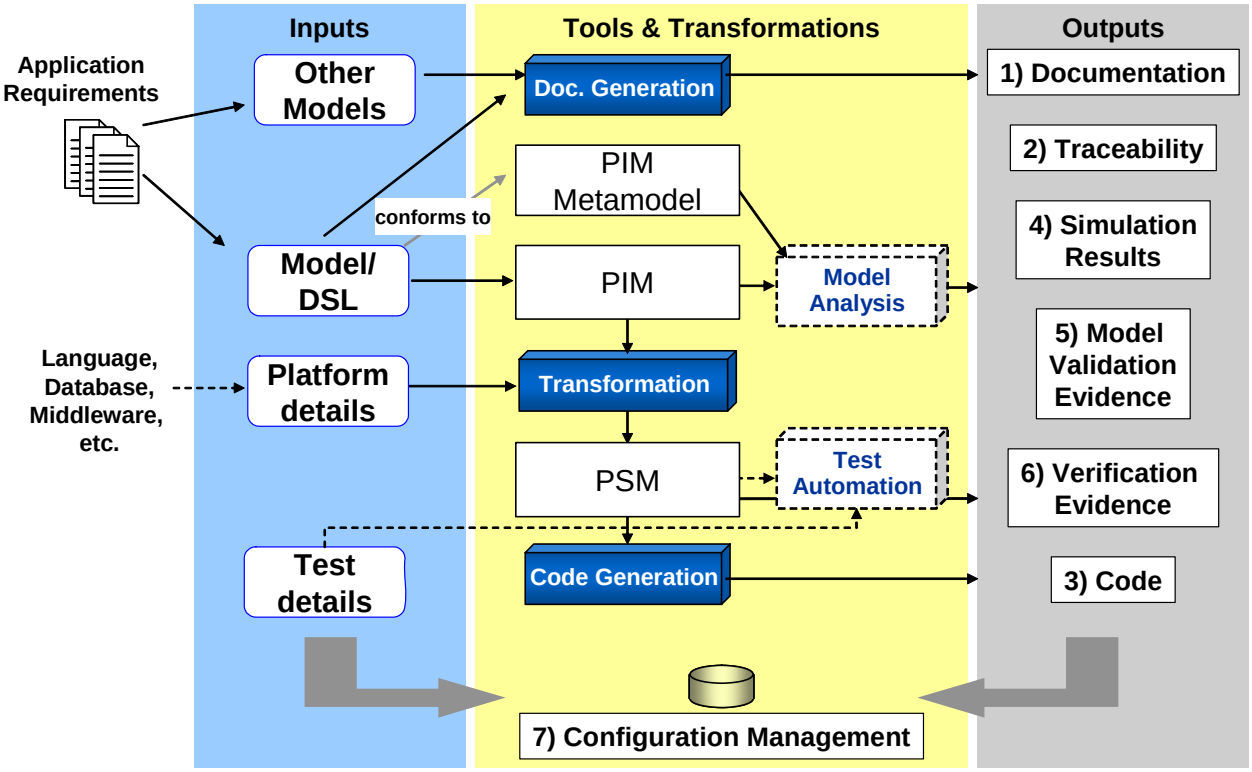


Figure 14. Artifact Perspective Related to Generalization of MDA Concept

Eclipse Open Source Platforms

Some of the more established tools were developed before the Eclipse platform was created or emerged into a significant basis for tools. Currently, there are approximately 70 modeling projects, many which include integration from commercial packages into Eclipse. Just because a product is integrated into Eclipse, the tool is not necessarily free. There are different license strategies and users should make sure that if they plan to use or extend the products that they can comply with the licensing terms. However, for pilot project analysis, and to better understand tool technologies, Eclipse-based tools provide a substantial basis for getting started.

The Eclipse Modeling Project, as reflected in Figure 15 integrates a number of different capabilities that are leveraged by open source and commercial tools. The framework evolved from OMG standards efforts and contributions by IBM, Borland and others, and it addresses needs of both the embedded system and enterprise communities. For example, see the next section on Structural/Elaborative Code Generation

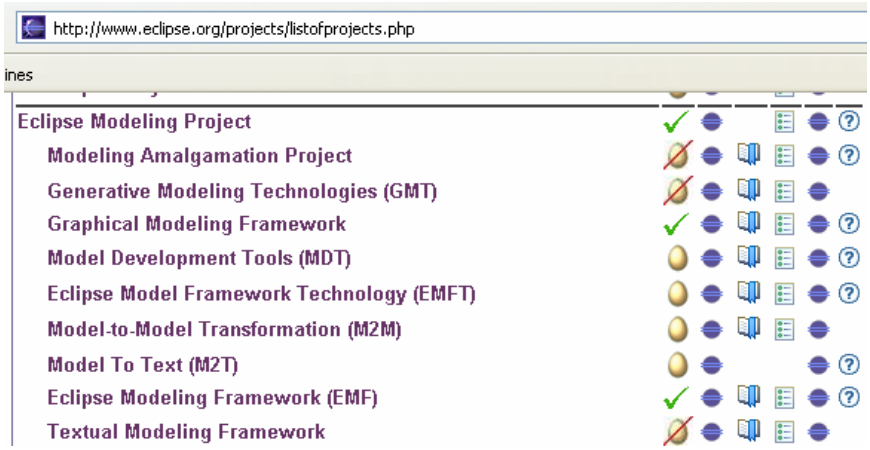


Figure 15. Eclipse Modeling Project

Structural/Elaborative Code Generation

For users unfamiliar with the concept of structural or elaborative code generation there are free open source plugins such as Omondo that can be used to quickly understand the concepts of structural/elaborative code generation. By constructing a simple class diagram, the tools produce from an UML diagram and class associations the structural elements of Java code shown in Figure 16. The Address class in the UML diagram is directly associated with the Java code Address.java. Attributes and operations added to the UML class are reflected dynamically in the Java code.

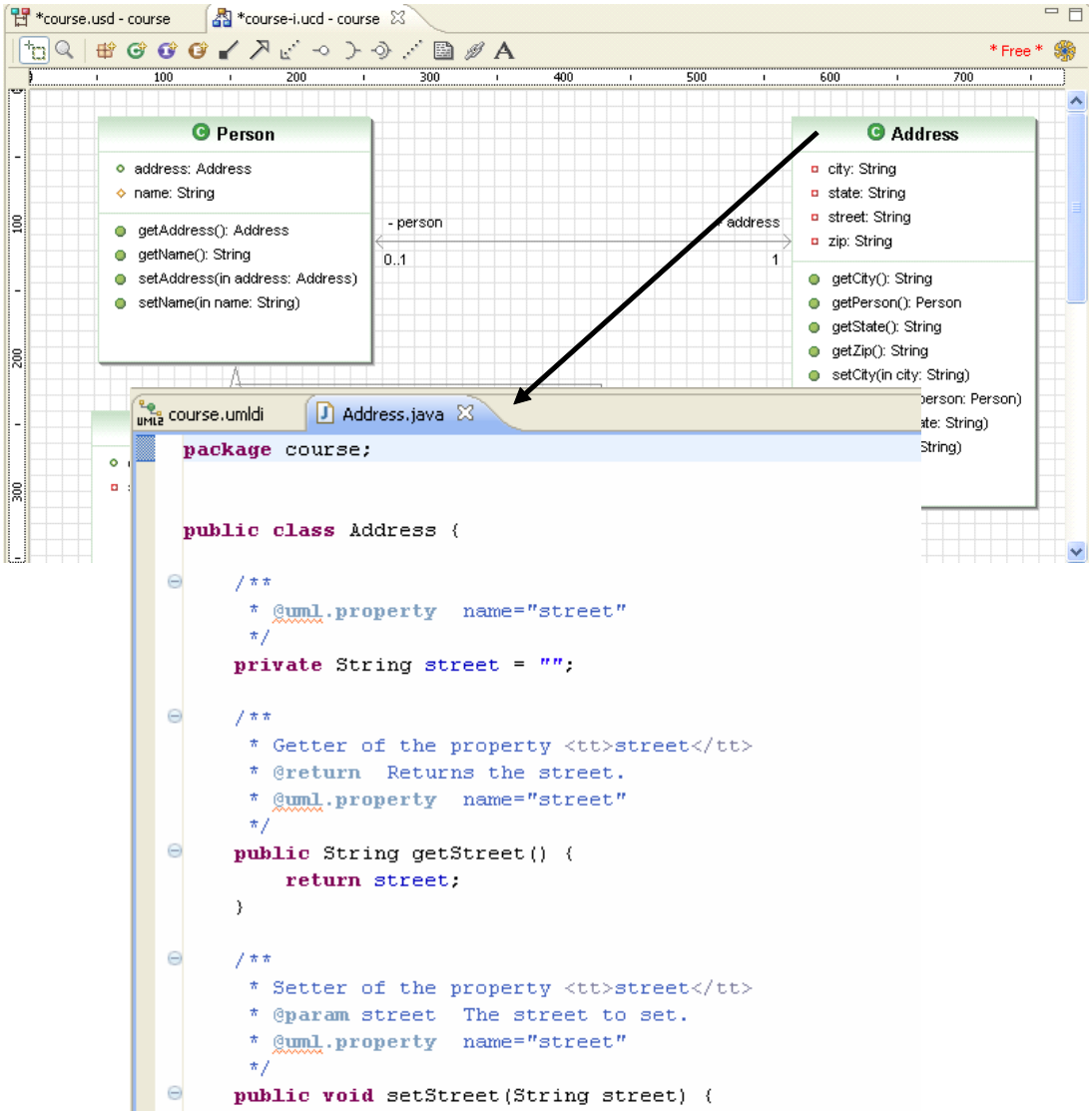


Figure 16. Class Diagram and Associated Java Code

Eclipse Modeling Framework (EMF)

The EMF project is a modeling framework and code generation facility for building tools. It represents information mostly related to class diagrams in UML, and was derived from OMG MOF. It is leveraged by many projects and modeling tools, for example it is the underlying foundation of the OpenEmbeDD platform as shown in Figure 17, which is an Eclipse-based "Model Driven Engineering" platform dedicated to Embedded and Real-Time systems (E/RT). The OpenEmbeDD project aims to offer engineers who design and develop E/RT software the means to express, simulate, validate and test the targeted system before any component implemented. For example, the Topcased tool kit supports:

- Graphical editors for generic modeling tools (e.g., UML)
- Graphical editors for modeling tools, dedicated to real-time and to embedded systems :
  - Structured Analysis Model (SAM)

  - Architecture Analysis & Design Language (AADL)
  - SysML
- Generation of graphical editor for models
- Model transformations
- Anomaly Management, Version Control, Requirements Traceability
- Documentation generation
- MARTE: OMG UML profile for Real Time systems; another domain specific language (DSL) – see Session 4 for more information.

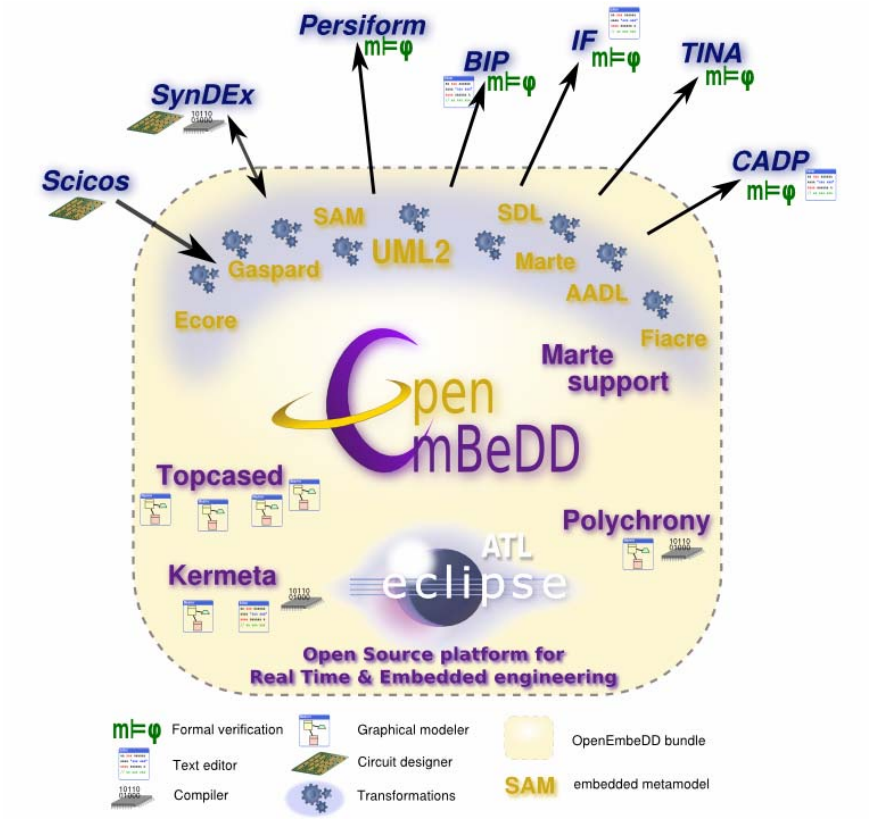


Figure 17. Open EmbeDD Platform

Requirement Models

Requirement models provide a different perspective on modeling that can provide significant ROI for the effort involved in producing the model. A requirement model describes the behavior in terms of the interfaces to a component or system. A requirement model describes “what the box should do” as opposed to a design model that describes “what is in the box.” People sometimes confuse requirement management such as that supported by tools like DOORS with requirement modeling. To clarify, tools such as DOORS, as shown in Figure 18, manage requirements using an outline form, much like a requirement specification document. It has capabilities to link and report on additional artifacts, but the information is not formalized like models, although DOORS could link to a formal model. Use cases based on a template (e.g., precondition, postcondition, main scenario, alternative course) add structure to requirements, but are not formalized as models. SysML supports a diagram for requirements, and it provides

greater requirement structure with visible traceability, as shown in Figure 18 but the information is still mostly text.

DOORS® SysML

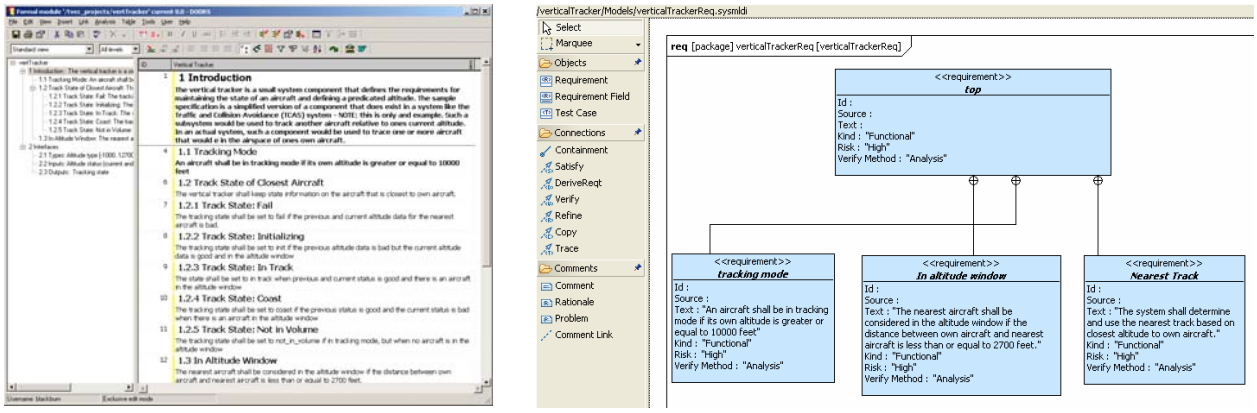


Figure 18. Requirement Management versus Modeling

The Software Cost Reduction (SCR) Method[6] created by Naval Research Laboratory formalized concepts for requirement modeling. Simply stated, behavior is formally defined in terms of:

- System inputs, outputs, terms, and mode classes
- Condition Tables - define values in terms of computations when associated conditions are met
- Event Tables - define values in terms of computations upon occurrence of discrete events
- Mode machines (classes) are simple state machines with:
- Modes (states) and transitions

There are a number of tools for SCR that support modeling, simulation, model analysis, model checking, proof, and verification (testing). There are significant resources that discuss SCR-related tool and case studies conducted by SSCI members[7].

One SSCI member documented that significant benefits achieved through the use of Requirement Modeling[8]. Figure 19 provides a conceptual overview of the roles and flow of the artifacts that ultimately result in the target software. Figure 19 represents how the use of models was integrated into the existing project process. The adapted process included both the traditional process steps and roles (top of figure), and modeling extensions (bottom of figure)

[6] Heninger, K., Specifying Software Requirements for Complex Systems: New Techniques and Their Application, IEEE Transactions on Software Engineering, Vol. SE6, No. 1, Jan, 1980.

Heitmeyer, C., A. Bull, C. Gasarch and B. Labaw, SCR*: A Toolset for Specifying and Analyzing Requirements, Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95), June 1995.

[7] The Engineering of Model-Based Testing: Guidelines and Case Studies, SSCI-2005005-MC, Version 01.00.25, July 2005.

[8] Part accepted to CrossTalk – date for publication not known at release of this whitepaper.

used to develop the application software. The system engineer develops textual requirements as well as any other type of analytical model that are captured in a Software Requirement Specification (SRS). The lead software architect identifies the components of the software architecture and works with the software requirements modelers to formalize the requirements and associated interfaces into models. The Software Requirements Modeler develops requirement models from the SRS and interface control document (ICD) using a modeling tool that supports the SCR method. Models capture behavioral requirements and interface information (e.g., inputs, outputs, types, ranges) extracted from an ICD.

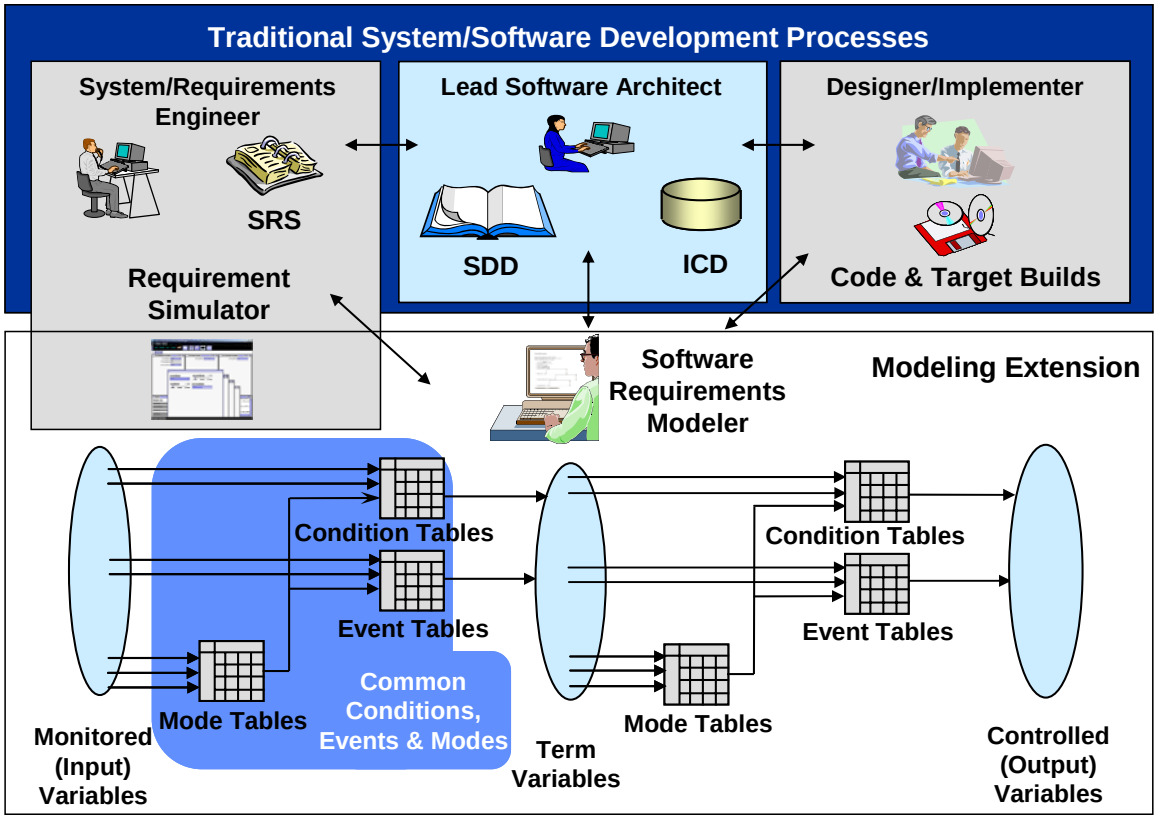


Figure 19. Process Roles and Flow

The modeling process often identifies requirement or interface problems that must be resolved through interaction between the system engineer or software architect. For example, interface specifications were captured in a database that is shared by the project team, including subcontractors. The requirement modeling process and associated tools force the interface information to be complete and consistent. Additional problems or anomalies are identified by the system engineers through requirement simulation of the models. Validated requirement models are linked to the Software Design Document (SDD). The designers and implementers work directly from the SDD, requirement models, and interfaces to implement the code. These modeling-related extensions to the process help to improve the overall performance of the team. Better requirements and interface documentation allow software designers to focus on the detailed design and implementation of the code rather than chasing requirement issues or making assumptions that can result in costly rework.

Figure 20 shows measurement data that provides a basis for discussing the benefits of using requirement modeling. Figure 20 compares data from the new (i.e., process with requirement modeling) against old (i.e., traditional text-based requirements process) using measurement

data that was captured in 100-day increments from the start of each respective program. The software system developed by both projects was very similar. The base measure that was common to both program, shown in Figure 20 is the total number of accumulated integration problem reports (IPRs). The number of IPRs for the new program was slightly higher than the old program through the first 400 days of the program. The number of IPRs for the old project increases significantly at about the 500th day of the program, and by the 800th day of the program the number of IPRs on the old program is about double the number of IPRs on the new program. The modeling activities helped find defects early and helped to minimize the defects later in the program even though both programs had ongoing releases associated with updated requirements over the multi-year program.

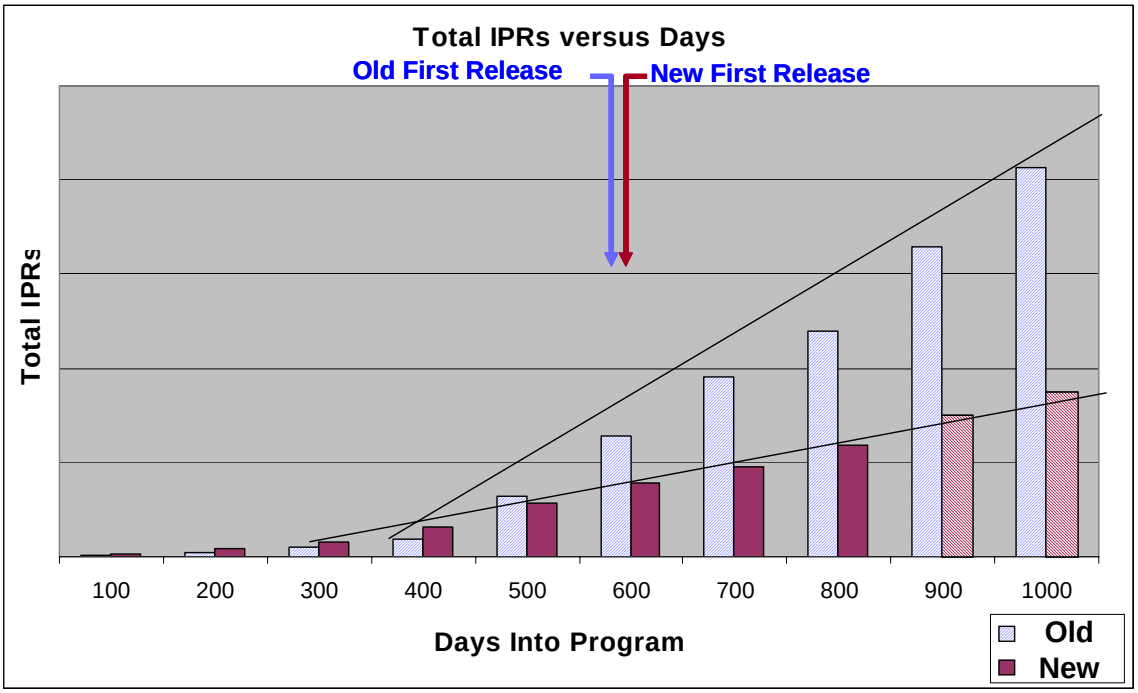


Figure 20. IPRs versus Day Into Program

Another benefit of the models is that approximately 90% of the detailed software design descriptions rely on the requirement models. The requirement model is linked to the SDD rather than having the design specified in text. Models represent both high-level and low-level requirements (i.e., derived requirements). Unusual or complex designs are documented in the SDD using text, flow diagrams, or other engineering drawings as needed. This is another process efficiency gained through leveraging the requirement modeling process. The model provided a formal, precise statement of the requirements that could be referenced directly in the SDD.

Metamodels

The final perspective on models is the metamodel. A metamodel describes how modeling constructs can be used together. Metamodel tool support has resulted in more rapid creation of modeling tools. Some tools use UML as a metamodeling language and through a metamodel a DSL can be defined. As an example, Figure 21 reflects on the Generic Model Editor (GME) and associated tool suite developed by Vanderbilt. Metamodels allow rapid development of DSLs and associated modeling editors and tools. Microsoft provides a free DSL toolset that integrates

with the Visual Studio, and although the details of the Oslo Project[9] have not been fully disclosed, it appears that Oslo will be a DSL or set of DSLs that allow models to be used to create various types of applications on Microsoft specific platforms. See Session 4 for additional details about how DSLs are used for many different types of languages.

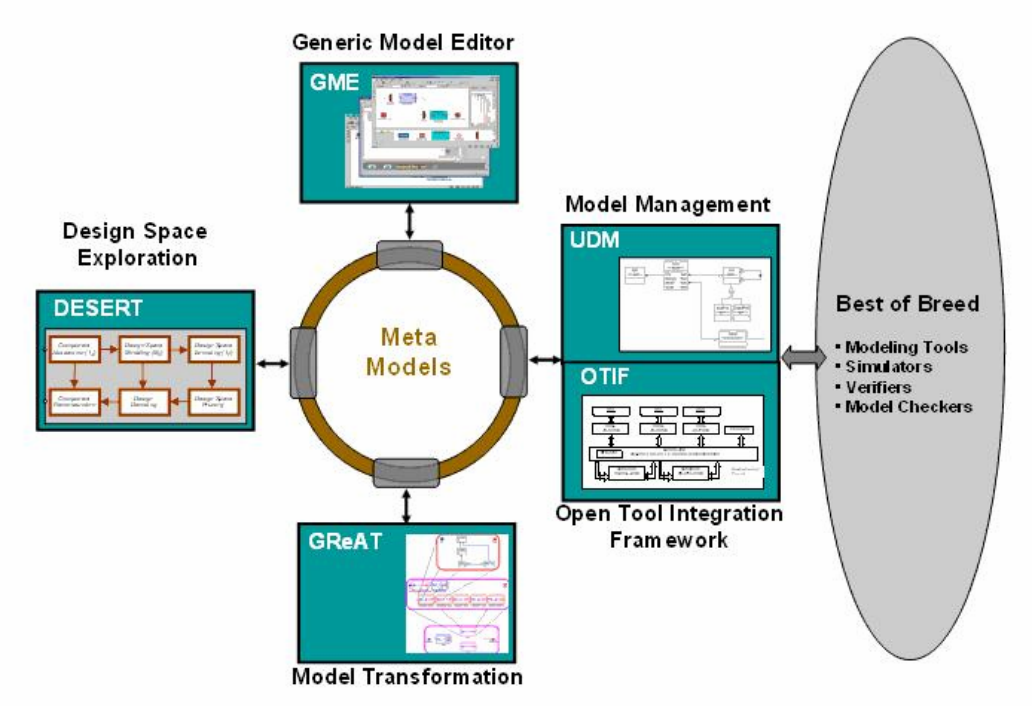


Figure 21. Generic Model Editor

Session 4: What's Next to Come with MDE?

Session 3 identified types of tools to cover design, requirement and metamodeling. The focus in the past has been on building models to support some type of hardware or software synthesis (e.g., code generation). It is difficult to predict the future, but there is a need for greater lifecycle coverage, and better model integration for systems, software and hardware. This session discusses model integration, which is needed to better support the complex interchange of information across multiple engineering disciplines. Domain-specific modeling approaches have and will promote the use of modeling in various domains, but model integration, and addressing system engineering integration concerns will be challenging. Tool automation is improving for models that represent structural and behavioral system aspects. Modeling approaches to support the system interactions such as timing, scheduling, and resource allocation is still needed, and this session briefly introduces some of the efforts to support concurrency and resource models that are being integrated with MDE approaches and tools. Model synthesis is only useful from models that are free of defects or can satisfy certain types of properties (e.g., timing, safety). Therefore, the session concludes by describing a few model analysis examples where MDE processes cover the entire lifecycle such that verification and validation (V&V) tools complete the development loop, supporting requirements management, and resulting in the

[9] http://www.microsoft.com/soa/products/oslo.aspx

development cycle becoming a living dynamic process with stability and correctness properties of its own.

This section takes a top-down view from high-level models to modeling tool technologies and briefly summarizes:

- Model integration and challenges
- Evolution of domain specific modeling language and associated standards
- Specialized model notations and tools
- Support for modeling concurrency
- Model transformation
- Model analysis, model checking and proof of properties
- Model-based testing
- Customer perspective on use of models, and the need for measures that can provide assurance from release-to-release that a system is working as modeled

Model Integration Challenges

SSCI members build complex systems that are often part of other complex systems of system such as the Future Combat System reflected by Figure 22. Many of the system elements are advanced and complex, but they must integrate with other systems that are equally advanced and evolving to address continually changing environmental situations.

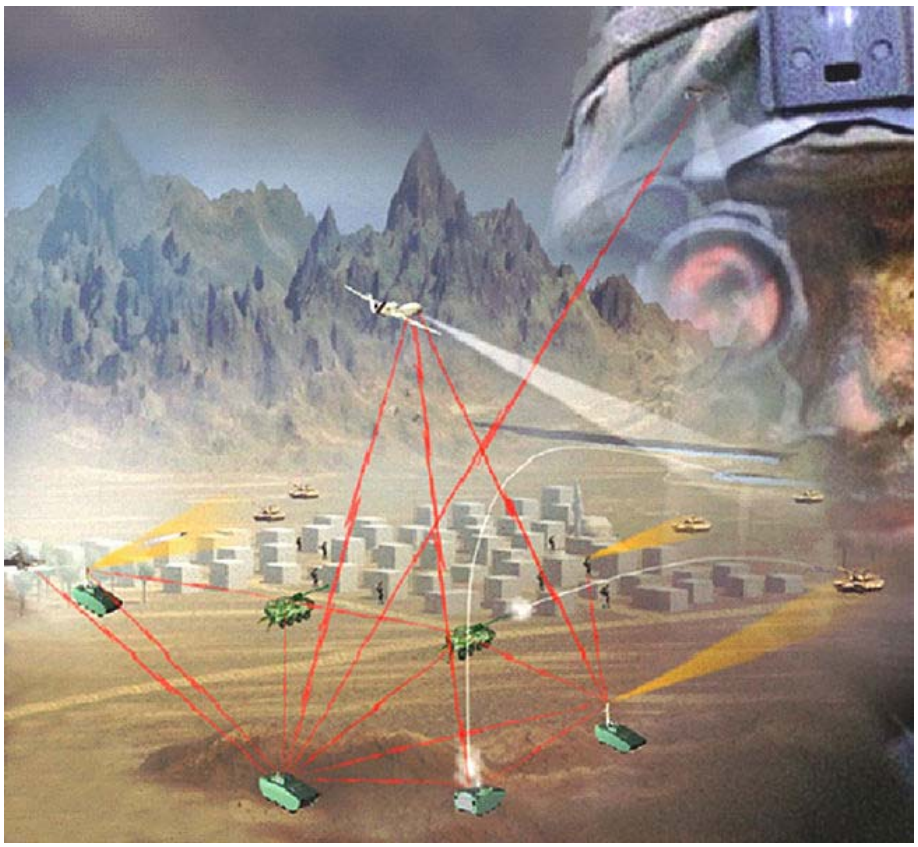


Figure 22. Future Combat System

Electrical and mechanical methods and tools for Computer-Aided Design (CAD) have been around for many years. Software modeling based on standards such as UML have advanced significantly in the past few years. There are standards supporting enterprise systems-of-systems such as DoDAF and MoDAF, and other standards associated with system engineering such as SysML and MARTE. Integration standards related to work by the OMG, INCOSE, and AP233, as well as Eclipse for open source development have resulted in many tools to cover various aspects related to modeling, simulation, code and document generation, and analysis. As reflected in Figure 23, any SSCI member organization that faces the challenges of developing these complex systems needs to improve the way they integrate the various disciplines, models, and tools. The ability to formalize information between groups would provide better assurance that the system will come together at integration time. In addition, model integration would support better dependency analysis related to new and changing requirements. This would allow SSCI member projects more information for cost and schedule impacts and estimation. There are significant integration challenges, and some related topics are described in this section.

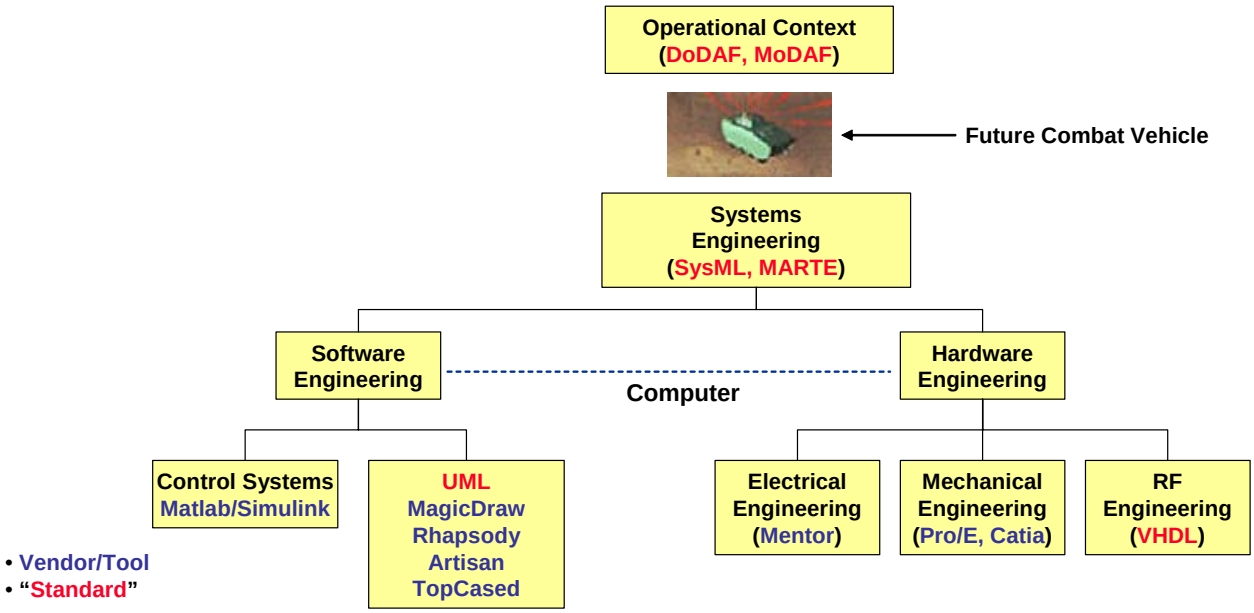


Figure 23. Models, Standards, and Tool Example

Enterprise and System Modeling Languages

A brief introduction of some of the enterprise level and systems modeling languages and frameworks is provided with pointers to more information. DoDAF[10] provides a set of views for organizing enterprise or systems architecture as reflected in Figure 24. MoDAF[11] defines a way of representing an enterprise architecture which enables stakeholders to focus on specific areas of interests in the enterprise. MoDAF has been developed from DoDAF and keeps compatibility with core DoDAF viewpoints in order to facilitate exchange of architectural information as shown in Figure 25. Tools, for example MagicDraw provides a UML profile for DoDAF and SysML,

[10] DoDAF http://www.defenselink.mil/dbt/Training/ACART/DoD_Architecture_Framework.htm

[11] MoDAF http://www.modaf.org.uk/

allowing the model development to be integrated. The key challenge is integrating the modeling concept into the SSCI member organizations.

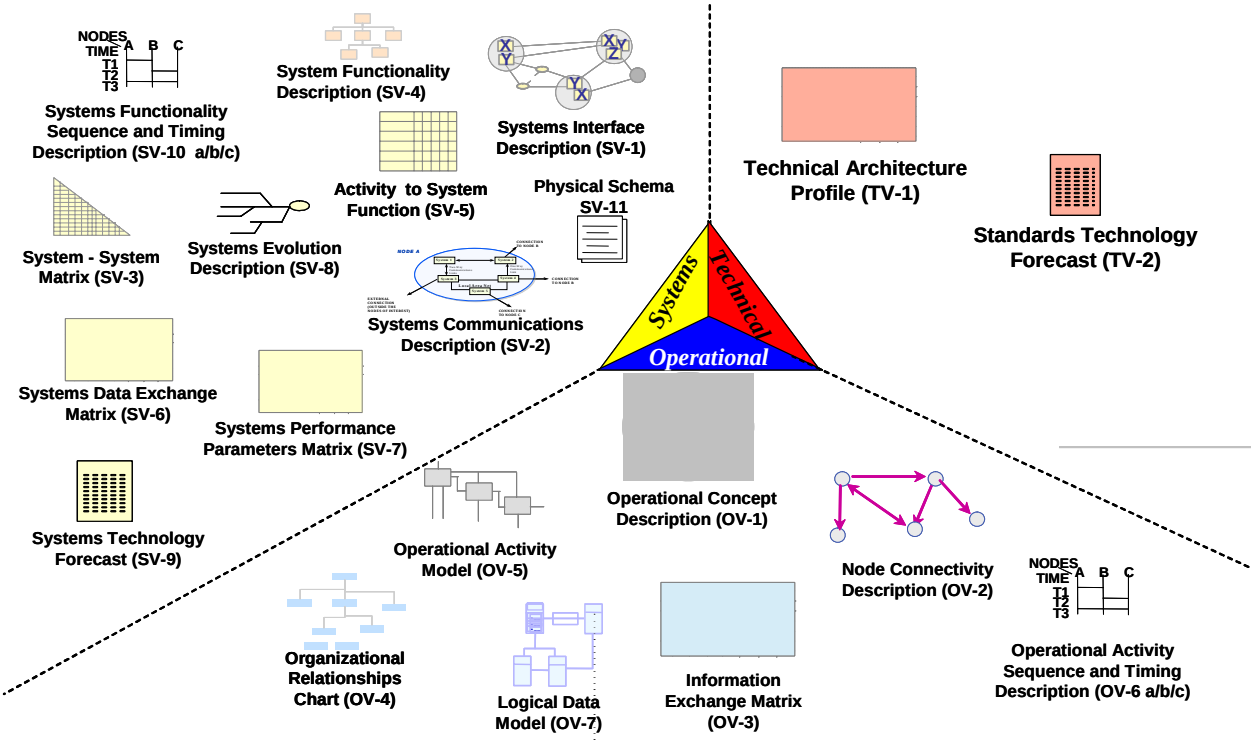
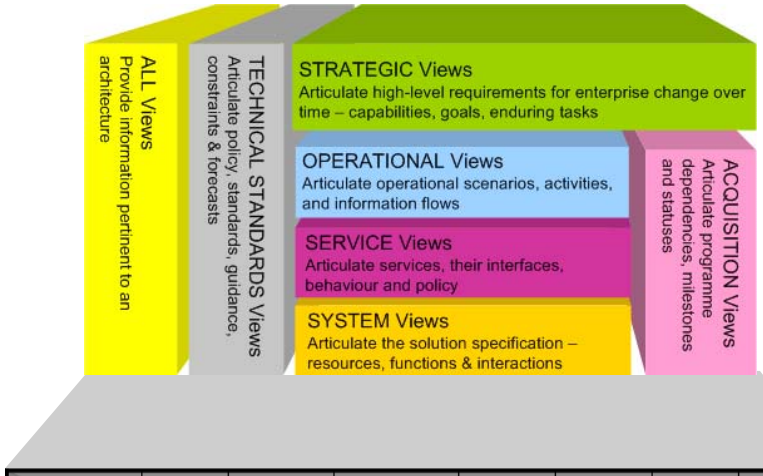


Figure 24. Department of Defense Architecture Framework Views Coverage



Category / VP	Tabular	Structural	Behavioural	Mapping	Ontology	Pictorial	Timeline
All Views	AV-1				AV-2		
Str VP	StV-1	StV-4		StV-3 StV-5 StV-6	StV-2		
Op Vp	OV-1b OV-1c OV-3	OV-2 OV-4 OV-7	OV-5 OV-6a OV-6b OV-6c			OV-1a	
Sys VP	SV-6 SV-7 SV-9	SV-1 SV-2a SV-2b SV-2c SV-11	SV-4 SV-10a SV-10b SV-10c	SV-3 SV-5 SV-12			SV-8
Tech VP	TV-1 TV-2						
Acq VP		AcV-1					AcV-2
Service VP	SOV-2		SOV-4a SOV-4b SOV-4c SOV-5	SOV-3	SOV-1		

Figure 25. UK Ministry of Defence Architectural Framework

SysML[12] is a graphical modeling language in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233. SysML extends UML using diagrams that support structural and behavioral views, and has requirement and parametric diagrams as shown in Figure 26. SysML supports specification, analysis, design, verification and validation of systems that include hardware, software, data, personnel, procedures, and facilities. It supports model and data interchange via XML Metadata Interchange (XMI) and the evolving AP233 standard. AP233[13] is an ISO standard specifying communications pipeline between systems engineering tools and databases. The details of AP233 are important to model integration, but beyond the scope of the Webinar series and this paper. The OMG provides the latest information on SysML[14].

[12] SysML Overview and Tutorial – see http://www.omgsysml.org/INCOSE-2008-OMGSysML-Tutorial-Final-reva.pdf

[13] AP233 - http://www.ap233.org/

[14] SysML standard and documentation - http://www.omgsysml.org/

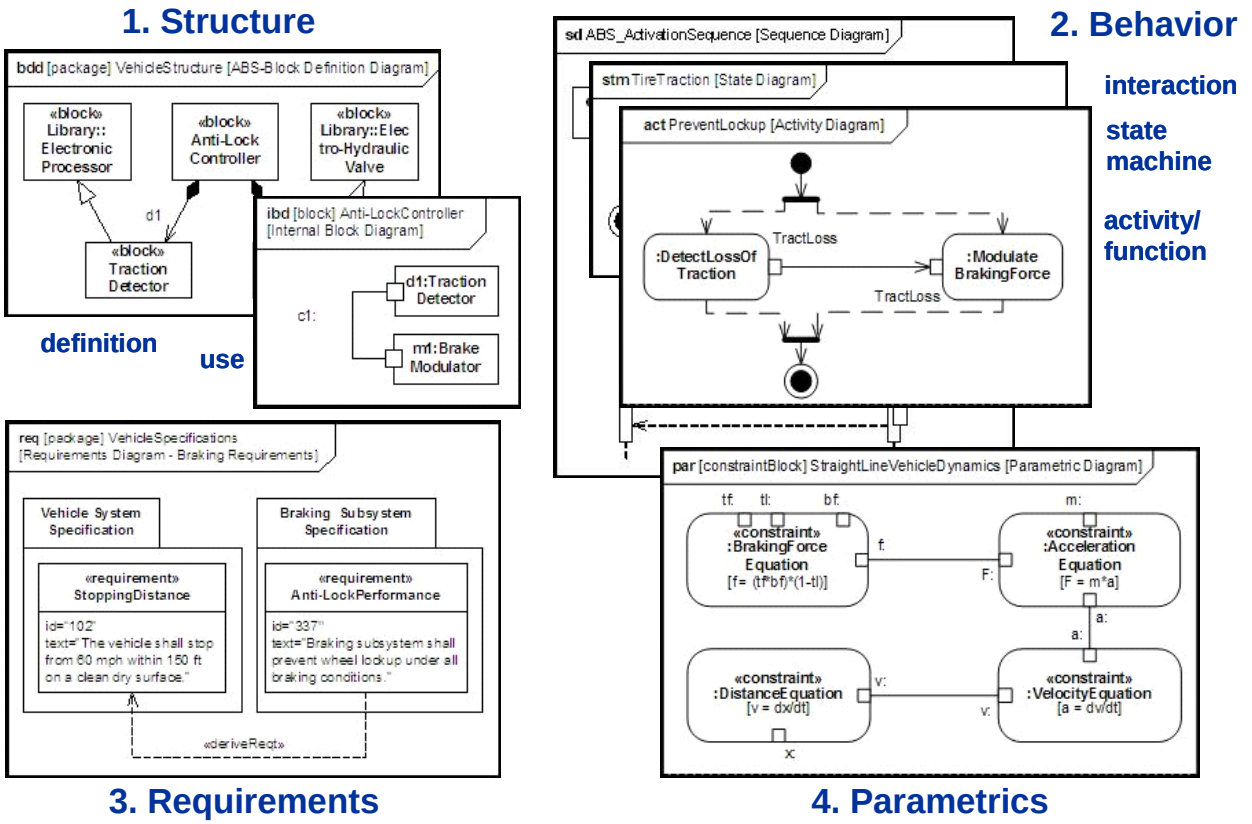


Figure 26. Pillars of SysML

Model Integration Steps

Like any system, for example a house, it's difficult to add one aspect of a home's functionality such as plumbing, without considering the location of the electric wiring, switches and walls, and this is true in the integration of models. As summarized in Figure 27, Joseph Sifakis, a co-winner of the 2007 Turning award stated that we need a holistic approach to integrate the essential domains, not simply extend hardware and software. This particular statement is based on embedded systems where there are significant integration challenges related to both software and hardware. Although the emphasis is on correctness-by-construction, in a world where hardware can fail simply by breaking, the demands on software that can support diagnosability, adaptivity in order to reconfigure the system to support some type of survivability demands more information about the system itself. These are models of the system ***resources*** and ***concurrency*** to address formal models of distributed processes that can support a reconfiguration effort.

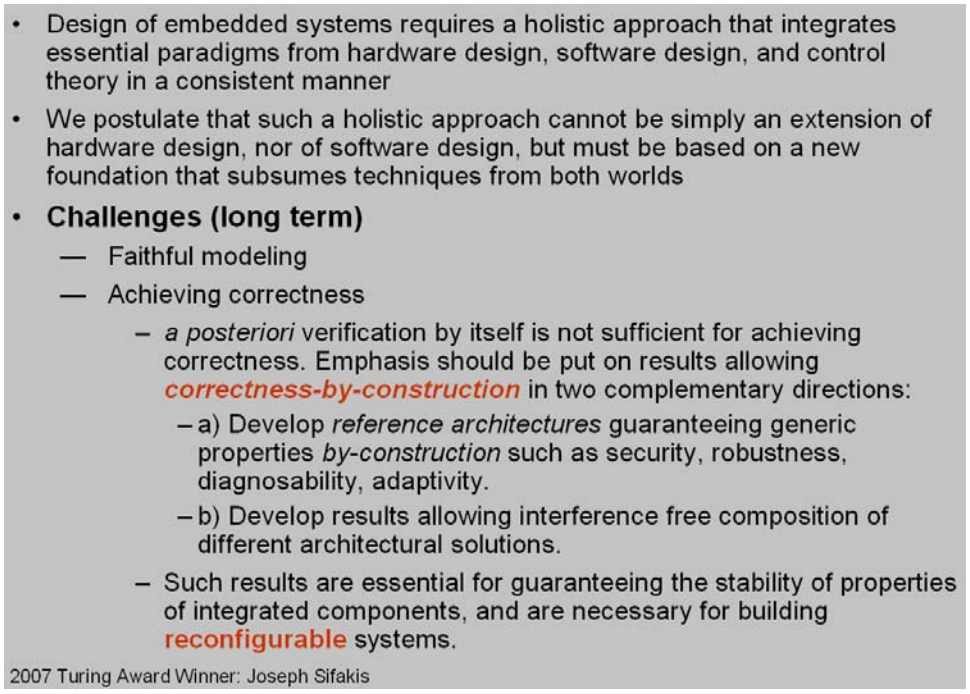


Figure 27. Embedded System Design Challenge

Representation and Tailoring of Models for Domain-Specific Uses

Modeling and Analysis of Real Time and Embedded System[15] (MARTE) is the first attempt at UML for real-time engineers like SysML was the first attempt at UML for system engineers. It addresses key elements to support real-time system details such as time, resources and scheduling. MARTE is attempting to support models for time that consider the concept that system time is not equivalent to physical time (e.g., two systems could have clocks that are not synchronized). MARTE also defines concepts for software and hardware resources as reflected in the example shown in Figure 28. There is significant work to do to provide better tool integration support and there is participation by most of the major commercial tool and service companies. In addition there are open source tools that are available through OpenEmbeDD platform discussed in Session 3 and represented in Figure 17.

[15] See http://www.omgmarte.org/Specification.htm for more details and presentation at this site identify tool vendor involvement.

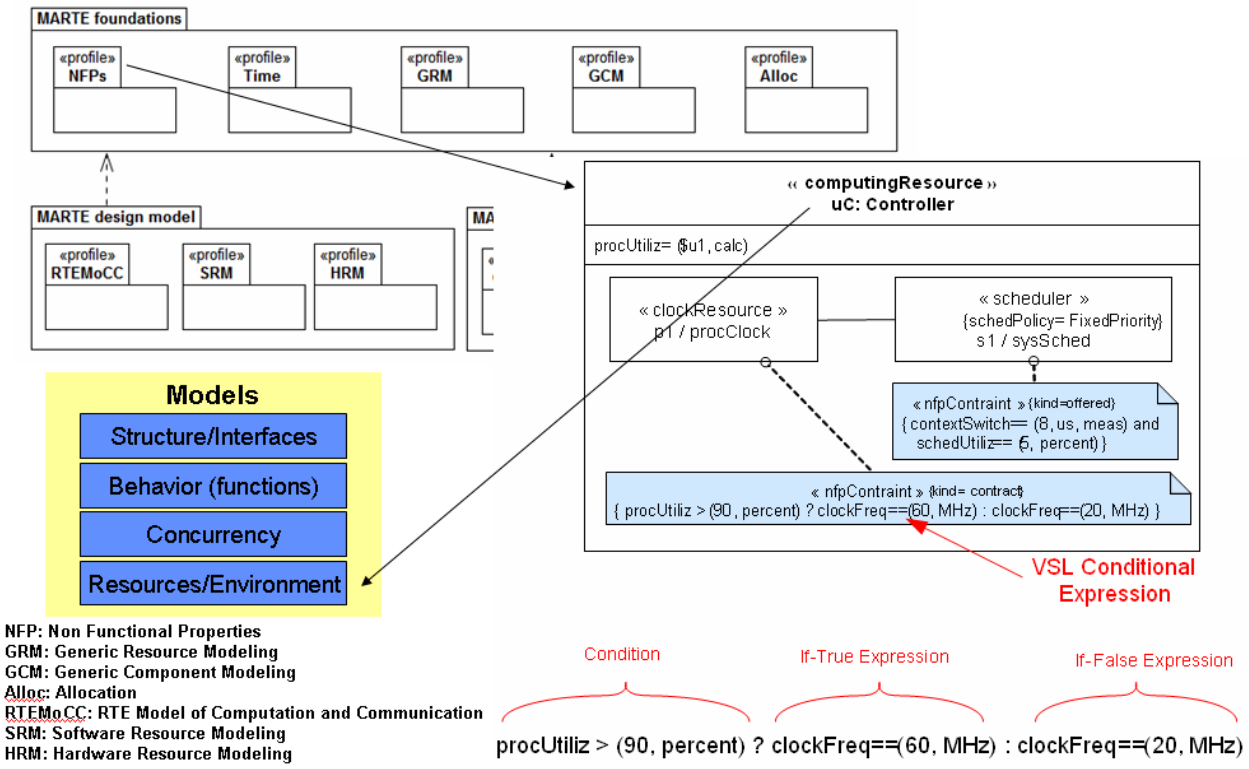


Figure 28. MARTE Resource Specification of Non Functional Property

Model Representations for Concurrency

The concept of concurrency is required to support distributed and parallel systems which are much more prevalent today. Developing the control for distributed systems has been the topic of research for many years. To align with the needs reflected by Sifakis' challenge shown in Figure 27 the only way to obtain highly dependable[16] systems is to develop it to be complete and correct or to be adaptable in the face of failure. Considering that hardware can break, especially when exposed to hostile environments such as the Future Combat System, it is essential to develop approaches for adaptability and reconfigurability. However, adaptation involves concurrency in that one or more processes must be able to monitor the system for failures and be able to modify the system processes to reconfigure and continue with some aspects of the overall system functionality.

Professor Edward Lee (UC Berkeley) has been working on models of concurrency for many years, and his work on the Ptolemy II project studies modeling, simulation, and design of concurrent, real-time and embedded systems. Lee points out that there are some potential flaws in the concept of threads[17], which are often used to support concurrency. Not all threads are problematic, but because threads can share memory there can be undesirable consequences such as deadlock. The Ptolemy II project focuses on assembly of concurrent components. The

[16] Dependability is a collective term subsuming the notions of reliability, availability, safety, confidentiality, integrity, maintainability, and security [Laprie 1984].

[17] The Problem with Threads, http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-1.html. See http://www.researchchannel.org/prog/displayevent.aspx?rID=9488&fID=2501.

underlying principle is to use well-defined models of computation that govern the interactions between components, and attempting to deal with the use of heterogeneous mixtures of models of computation. Ptolemy II includes modeling tools that focus on the concurrency aspects of a system. These tools are available for download, and there are efforts underway to integrate these tools with other tools to continue to extend tool chains to address concurrency issues.

As standards evolve, such as MARTE, tools continue to evolve too. For example, there are other tools that are part of OpenEmbeDD that provide time-oriented model analysis. Two examples include:

- CCSL is Clock Constraint Specification Language provides visual tree of parsed model clocks
- Polychrony environment based on synchronous multi-clocked model of computation with model checker

Model Checking

Model checkers can also support time-based analysis. Model checking is the process of checking whether a given structure is a model of a given logical formula. Model checkers can check to determine if a model satisfies certain properties (e.g., timing constraints). For example, finding errors such as data-races, deadlocks, livelocks in multithreaded software by exploration of a thread. The concept is general and applies to all kinds of logic, although it has been used more for hardware than software. SPIN is a well-known general tool for verifying the correctness of distributed software, but there are many other examples too. As discussed in the next section, model transformation is often required to pull different model views together to leverage tool chains.

Model Transformation

This section discusses other tools that may need or perform model transformations. The challenge with tool chains is that often one tool does not use or produce all the needed information for upstream, or downstream tools. The concept of transformation is not new, although other words may have been used to describe a tool's function in the past such as compilation or translation. Traditionally, the tools provided a text-to-text transformation. For example machine code from assembly to higher language forms such as Fortran, C, Java, including domain specific language SQL, and Prolog have been ongoing for many years. More general forms such as Extensible Stylesheet Language Family (XSL) Transformation (XSLT) transform XML documents into other XML documents.

Tools now support model-to-model, model-to-text, or text-to-model transformations. Model transformations are needed because information is not necessarily in one model. Representations that are suitable for simulation or code generation are often not directly usable to support analysis or verification. For example, consider the UML example shown in Figure 29. If the goal is to produce code, then there is a need for structural information and behavior. UML class diagrams provide structural information that maps directly to code as shown in the example in Figure 16, but this is not sufficient for code generation. UML behavior can be provided in state machines, sequence diagrams, activity diagrams or action specification such as that provide in xUML. Each of these individual aspects of behavior might not be sufficient for code generation. For example, state machines be associated with a class diagram to specify the lifecycle of an object, but that does not necessarily describe the application which requires a sequence diagram to specify the interactions between objects.

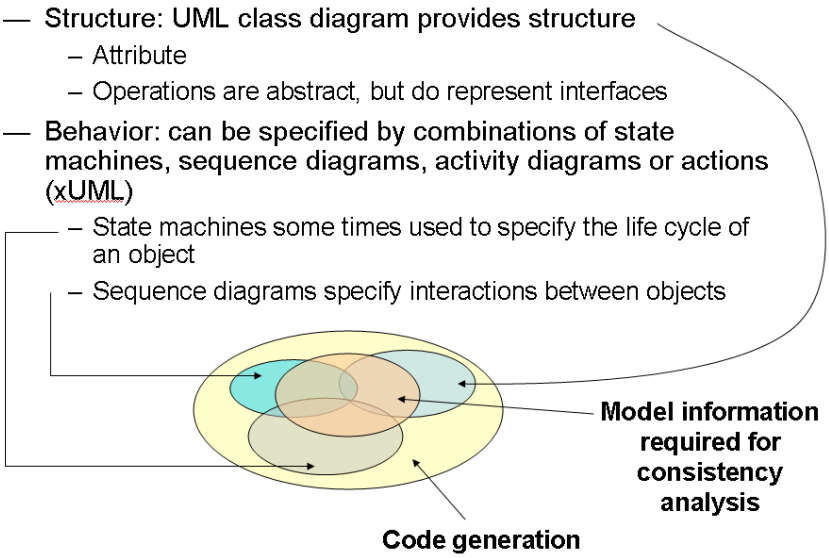


Figure 29. Models Transformation Required for Transforming Different Representations

The OMG standard QVT (Query/View/Transformation) is another MDA-related standard. From a simplistic point of view the concept of model transformation involves converting a model A conforming to metamodel A into a model B conforming to metamodel B as shown in Figure 30.

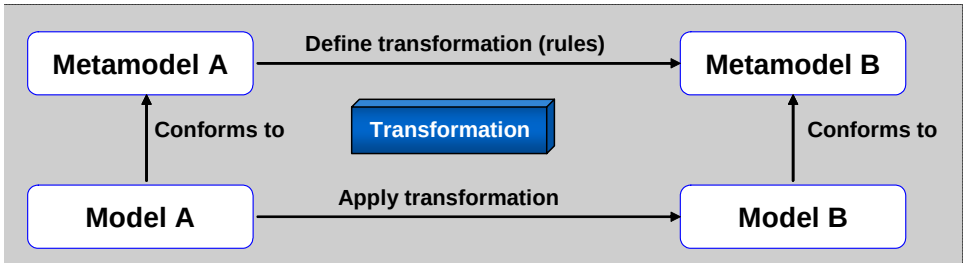


Figure 30. Simply View of Model Transformation Concept

Currently tools exist that are compliant with the OMG standard including the following non-exhaustive list:

- Borland Together is a component in the M2M Eclipse project
- SmartQVT an Eclipse open source implementation of the QVT-Operational language
- Eclipse M2M open source implementation of QVT operational

There are other transformation approaches that are not compliant with the OMG standard, for example the ATLAS Transformation Language (ATL)[18] was inspired by the OMG QVT standard and builds upon the OCL formalism. It is a hybrid language providing a mix of declarative and imperative constructs, and available as part of the OpenEmbeDD toolset making it easily available for experimentation and evaluation. openArchitectureWare provides yet another open source for transformation tools, and there are video examples at www.openarchitectureware.org that demonstrate the use of model transformation.

[18] http://www.eclipse.org/m2m/atl/

Proof of Properties

Significant focus has been placed on code generation, but model analysis is important to ensure that models meet certain criteria before being used for code generation. If the model has defects, then the correctness of the generated code is likely to be incorrect too. Model analysis can identify defects in a model such as inconsistent constraints or behavioral conditions, and violation of timing or safety properties.

There are different approaches used to support model analysis. Model checking was briefly mentioned above, but theorem provers provide another approach to support model analysis. One of the most basic types of proof involves proof by contradiction. The programming language Prolog uses this basic mechanism to support computation. For model analysis the idea is simple:

> Assert something is NOT true, then if a solution is found that violates the proof, it identifies a problem in model

This mechanism can be used for checking that a model satisfies safety properties too, for example, consider the safety property:

> The aircraft radar should not be enabled when there is weight on wheels

If this particular situation is permitted within the model, then this violates the safety property, because the model analysis has detected certain input conditions that would potentially allow a person on the ground near the aircraft to be “radiated” while the aircraft has weight on wheels (i.e., is on the ground). Using a proof mechanism, an assertion can be made stating that *radar is enabled* and *weight on wheels is true*. A theorem prover would determine if there are any paths through the model that permit this situation, and if so, it would show those paths to the user performing the analysis, leading hopefully to a correction of the model.

A similar approach can be used to identify unreachable paths through the system. The basic process applies the general question:

> Can a function or path within a model be reached? Or, are there paths to functions that cannot be reached?

This type of check can be automated, although model transformations are generally required to transform the model into a form that can be processed by theorem provers. Consider the simple example shown in Figure 31. Assume that there are three variable:

```
x: Integer with domain from 0 to 10
y: Integer with domain from 0 to 10
z: Integer with domain from 0 to 10
```

If there is a requirement that specifies:

```
z = 0 when
      x < 3 AND
      y < 4 AND
      x + y > 7

then
   maximum value for x is 2
   maximum value for y is 3
```

```
minimum value for x + y is 8
```

The region represented by the intersection of x & y does not overlap the constrained region defined by x + y > 7. The constraint expression is contradictory and cannot be satisfied, because the variable z will never be assigned a value of 0 through this requirement. Thus, the model has a defect. Real-world problems typically include complex constraints that span many modules or components of an application. In these situations it can be difficult to isolate these types of errors through manual processes. Automated model analysis provides a tool for locating these errors. The Session 4 Webinar slides provide a detailed set of slides that summarize a similar situation that is briefly explained below.

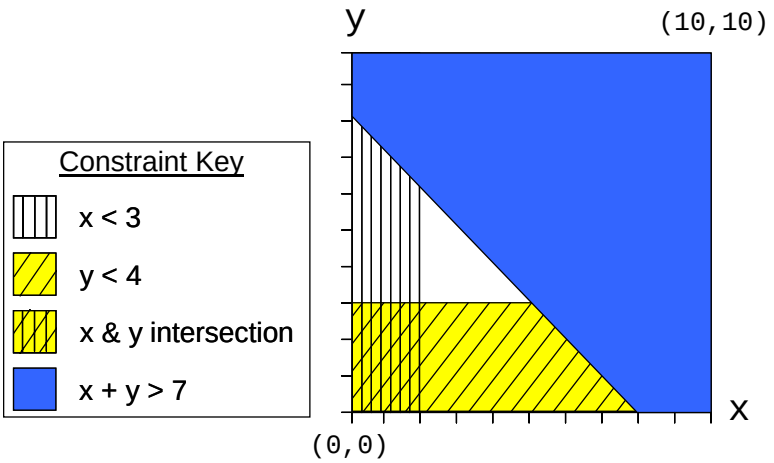


Figure 31. Inconsistent Constraints

The Simulink model shown in Figure 32 has a seeded defect to illustrate the model defect identification and tool chain traceability links from a model report to the model. The example includes four related Simulink subsystems. The highest-level subsystem, `hierarchical_root` references `child_yz`, and `parent_xy`, each with two threads. `Parent_xy` references `child_xy`, which also has two threads. As shown in Figure 32, the defect exists because there is no combination of threads through the lower-level subsystems that permit both `x` and `z` to be greater than zero when the output (i.e., assignment) of `hierarchical_root` must be TRUE. The model `child_2_xy` requires y <= 0 when x > 0, but `child_2_yz` requires y > 0 when z > 0. Thus, a contradiction exists between the logic of `hierarchical_root` and logic across two dependent subsystems.

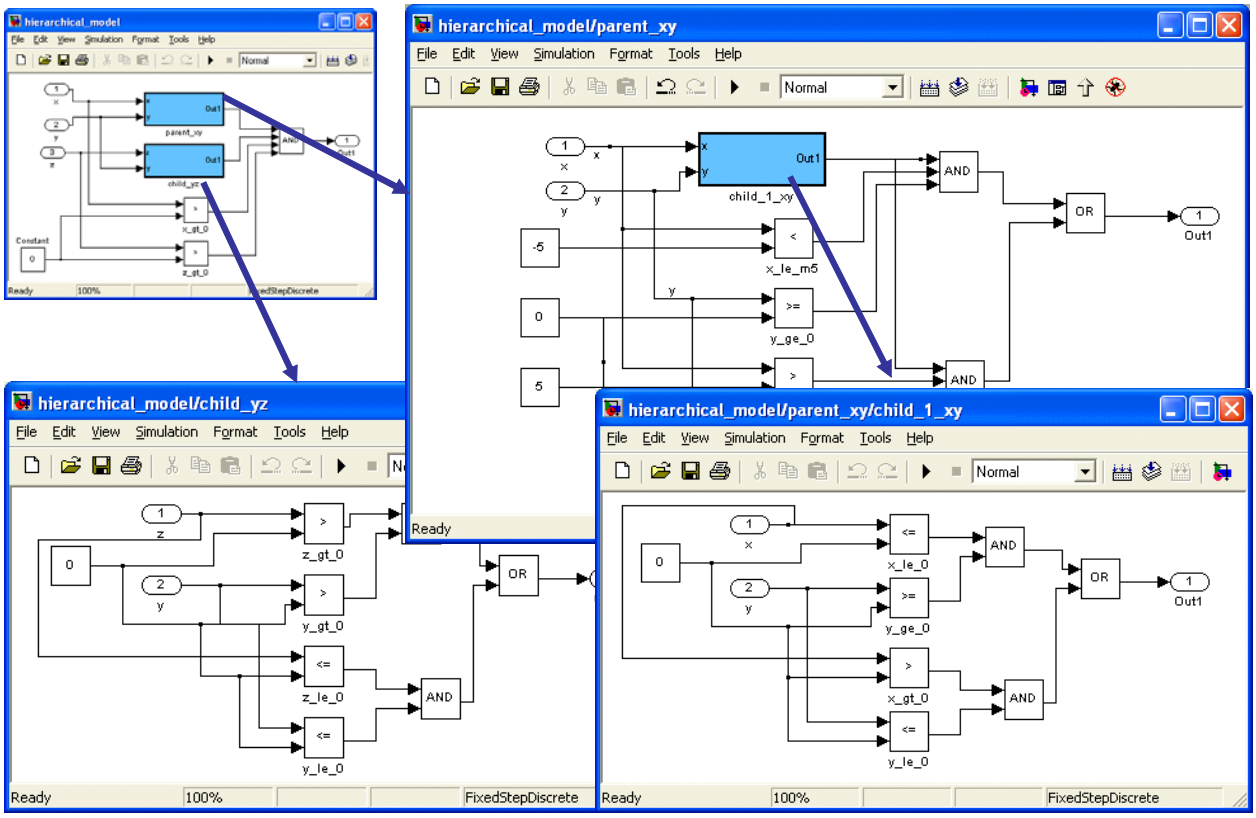


Figure 32. Model Defect Simulink Example

A model transformation from Simulink into T-VEC Vector Generation System (VGS) is performed to expand the model threads shown graphically in Simulink into analyzable paths as shown in Figure 33. VGS is general purpose tool supporting model analysis and test generation. There are three different, but related model transformation mechanisms that can convert graphical or textual models characterizing requirement, design and application properties (e.g., safety), based on representations such as decision tables, state machines, control system, and code, into a hierarchical form that mirrors the representation of the Simulink subsystems. The underlying modeling language provides support for an extensive set of mathematical operators (e.g., trigonometric, intrinsic, integrators, quantization, matrix) that extend standard arithmetic operators to specify functional behavior supporting various applications domains. Other VGS components include the test vector generator that integrates with a test driver generator to produce test drivers that automate test execution for most any language and test environment with automated test results analysis.

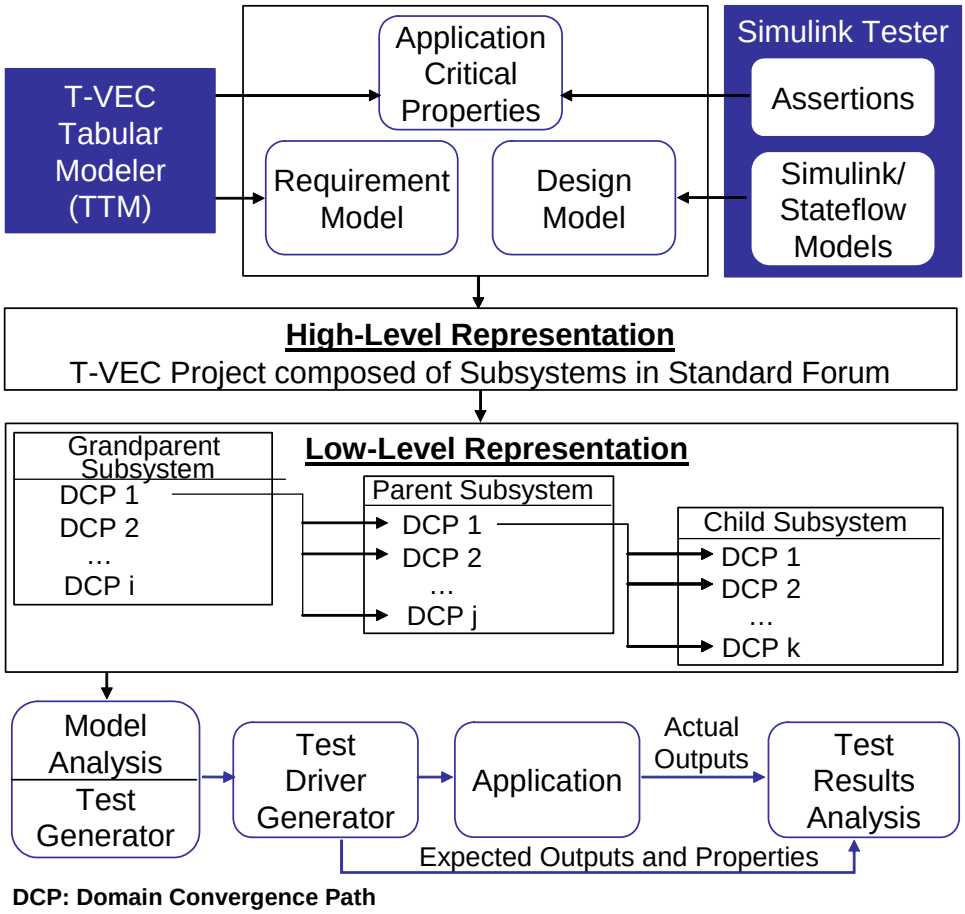


Figure 33. Simulink to T-VEC VGS Model Transformation of Structure and Behavior

The traceability links from the VGS status and error reports link to the likely source of the model error as shown in Figure 32. The status report provides a summary for each subsystem, including the number of Domain Convergence Paths (DCSs) derived during the model transformation process. The summary report provides the number of test vectors, and the number of model coverage errors. Hyperlinks from the project status report link to other reports including the model defect error report that is produced for each DCP that has a defect. A hyperlink from the model error reports traces back to the Simulink model construct that is the likely source of the problem.

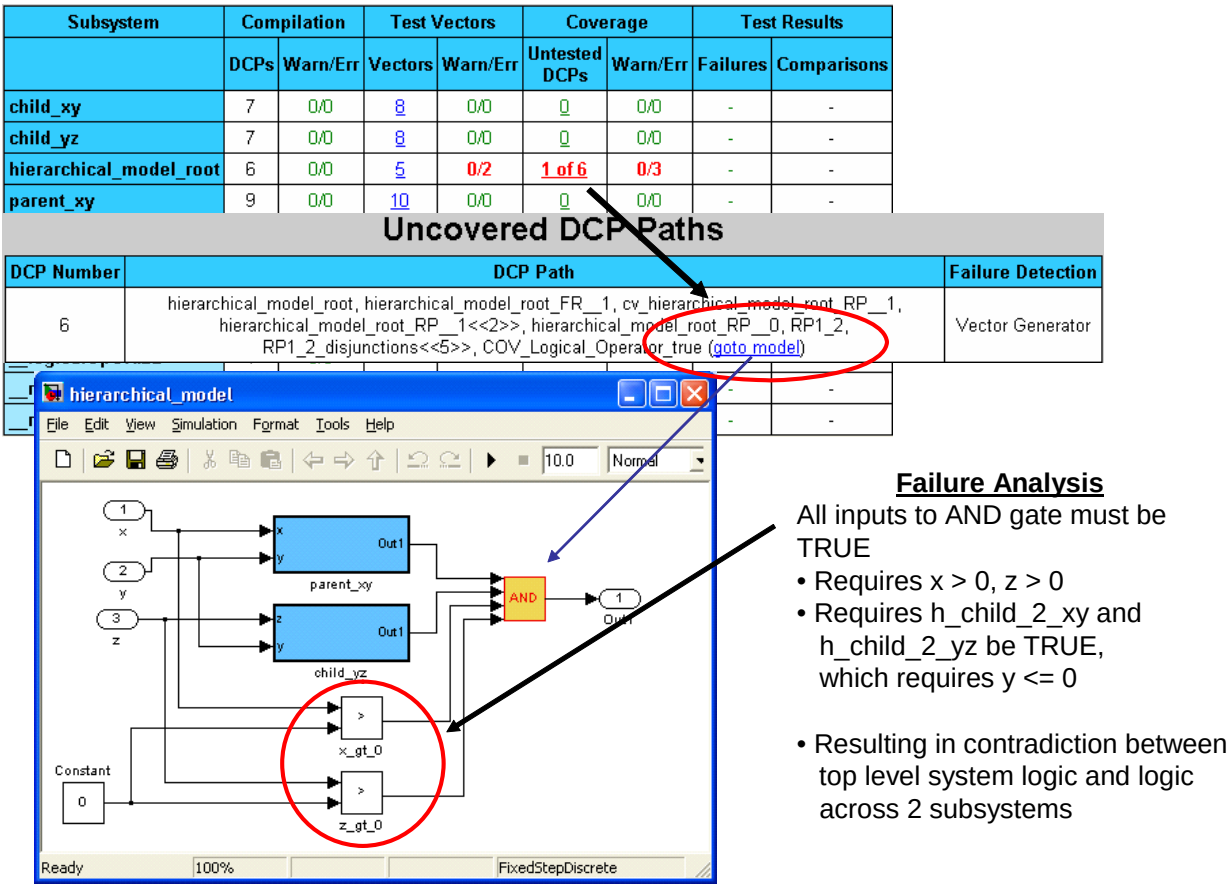


Figure 34. Model Coverage Results and Traceability Links

The model analysis capability also supports proof of properties (e.g., safety). Model assertions representing safety properties can be specified external to the model, and during the test generation process, if test vectors are generated from a safety property assertion that is associated with a model, the test vector identifies a DCP thread through the model, where the safety property is violated.

Other checks such as mathematical errors or potential errors (e.g. division by a domain that spans zero) are flagged as being a potential divide-by-zero hazard, or range overflow or underflow, where variables of the model have values outside the specified bounds of the type of that variable. The error reports generated for these errors link back to the model source.

Model-Based Testing

The process efficiencies derived from fully automated model-based testing can provide significant cost reductions especially as programs move from initial development and deployment to evolution and maintenance, however there are significant challenges due primarily to the completeness of the models and the ability to support model transformations that can leverage tool chains to support the test automation. Some of the key limitations result from:

- Lack of formalized behavior, which was pointed out as an issue for code generation too, for example – even if a state machine is provided for each class,

the constraint and actions associated with each state transition must be formally defined

- Lack of formalized mapping between the model variables and implementation variables to support automated test execution and results analysis
- Determining test inputs sets that can cover the threads of a model – often the complexity of the constraints can make the determination of inputs value difficult to determine
- Determination of the expected output

As shown in Figure 33, VGS provides automated test vector generation (i.e., inputs and expected outputs) and test driver generation to automate test execution and results analysis. It does this as a side-effect of performing the model analysis. Once VGS proves that a set of constraints from the model is satisfiable, test values for the inputs are selected from the boundary values, which are often most effective at finding potential faults in manually produced or automatically generated code. The expected outputs are computed internally to VGS based on the computation derived from the model transformation using the input values selected for test cases.

Model and Modeling Tool Evolution

The pace of advancement of modeling tools has increased over the past few years. Standards have helped establish a basis for modeling products, and open source efforts such as the Eclipse Modeling Project have provided a quickly evolving infrastructure for research and development of tool chains, while providing greater availability for evaluation, experimentation, and extension. However, users of the tools must know how different tool versions can impact evolving systems that SSCI members produce. Members must know that the system is operating correctly when new modeling types and tools are used. Modeling tools and the associated tool chains are more complex than the compilers that are used to produce code. SSCI customers are interested in the same question and will be demanding more information as they perform oversight of projects.

For example, consider Figure 35 that was taken from an SSCI training course that is being given to a NASA organization that is overseeing model-based tool usage on their programs. The NASA organizations do not necessarily develop models, but they want to have the assurance that the resulting system operates correctly. This involves several different points of view that are shown using the model analysis results shown in Figure 34. The oversight process, which could be used by a quality assurance organization too, is reflected in the flow chart in Figure 35:

1. Models should be free of defects

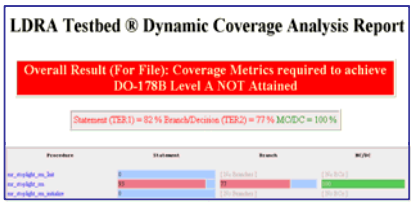
2. Generated tests should be executed against code that is instrumented to ensure that all paths through the code have been tested. Tools such as LDRA Testbed are used by SSCI members to support this function.

3. All test cases should pass (i.e., actual outputs should match expected outputs within numerical tolerances)

4. All test cases should be executed against un-instrumented code to ensure that instrumentation had no impact on the test execution results. This is the code that will be deployed in the target system.

5. Code defects should be analyzed to determine if the code or model was incorrect

Subsystem	Compilation		Test Vectors		Coverage		Test Results	
	DCPs	Warn/Err	Vectors	Warn/Err	Untested DCPs	Warn/Err	Failures	Comparisons
child_xy	7	0/0	8	0/0	0	0/0	0	8
child_yz	7	0/0	8	0/0	0	0/0	0	8
hierarchical_model_root	6	0/0	5	0/2	1 of 6	0/3	0	5
parent_xy	9	0/0	10	0/0	0	0/0	0	10



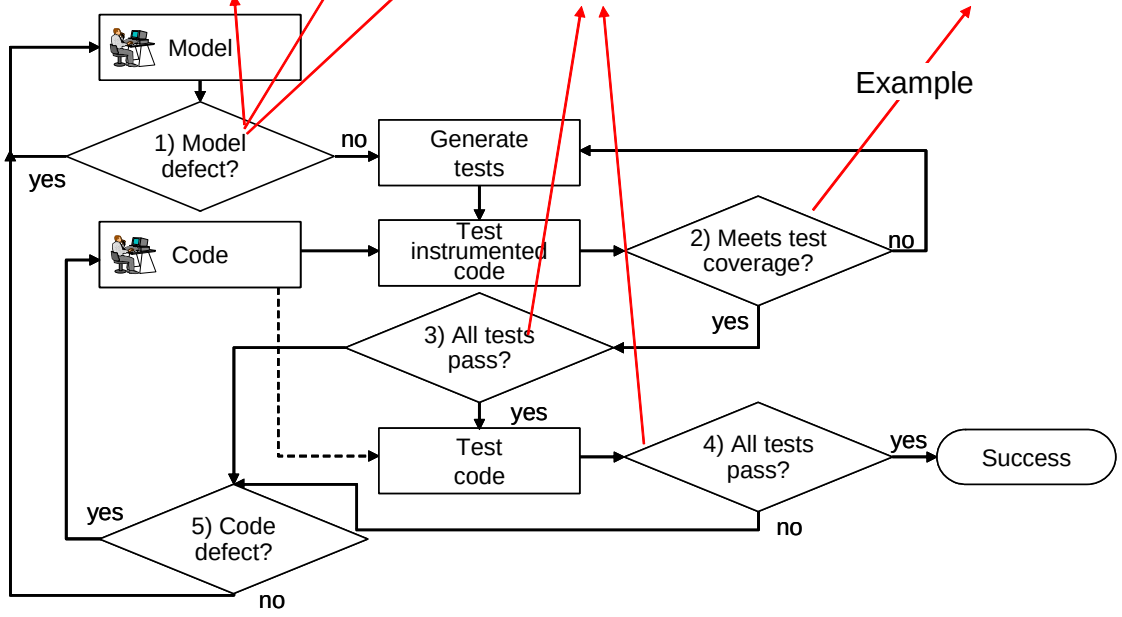


Figure 35. Model-Based Measures

Part of the model-based adoption process requires additional effort to follow a few additional guidelines related to the use of modeling tools that are evolving much more quickly than traditional compilers. An approach similar to the oversight role described above could leverage model-based test automation in making sure that model evolution and new tool usage is predictable and dependable. Minimally projects should put processes in place to:

- Create baseline models for re-validating tool functionality from release-to-release
  - Testing could be used to identify unexpected changes in the model
  - Performing automated differencing of the generated artifacts is also a possibility
- Update and maintain the baseline models with each new tool version
- Version control tools to ensure deliverables are reproducible from source artifacts maintained under configuration control

Conclusion

This paper and associated Webinar series has identified the types of models that can be used to support lifecycle activities so that members can better understand what they should invest in to achieve the immediate cost saving or long-term benefits. The focus attempts to identify and provide realistic expectation on the information that can be derived from models and associated modeling tools that can contribute to the lifecycle activities including development, verification,

evolution, maintenance and management of the software systems that are critical to SSCI members.

SSCI member organization should determine model-based technology objectives by using pilot project to assess technology and method alignment for their organization and programs. They should use a modeling maturity model concept to consider both ROI and key practices changes while making technology adoption decisions. As shown in Table 2, ROI is not necessarily the only factor for considering MDE adoption, because structuring of developmental practices provides longer-term benefits that can lead to formalizing organizational knowledge.

Table 2. Model Adoption Benefits and Tradeoffs

Levels	Goals/Approach	Benefits	Comments
1	Models not used.		The risk is too high.
2	Opportunistic use of models.	Increased awareness of modeling practices and terminology, potentially beneficial in future programs.	ROI should not be expected, but there is potential for increasing maturity of organization. Need to understand phase-in to legacy program, or adopt only on new programs.
3	Models used for guiding implementation and production of documentation. Code frameworks may be generated, but detailed logic created by hand.	Potential reduction in cost of documentation; training of people and acquisition and mangement of tools, and understanding modeling.	Expected ROI should be minimal, but process viewed as stepping stone for organization. Focus on modeling standards, model and project organization, model integration, and CM.
4	Separation of business, domain-specific models from platform-specific models, and models used to produce the target systems.	Organization domain knowledge in models is established, with standarized ways of managing and measuring projects. Separation of domain models from platform details, and significant software development done automatically through models.	The spectrum for level four could be large based on the type of system and potential targets.
5	Domain-specific and business models separated from platform; all models transformed to executable systems.	All intellectual knowledge and properties of the organization captured, reused and extended.	

Pilot project are the time when organizations should establish process guidelines, methods and standards, some of which are summarized in Session 2 of this paper, as well as take into consideration proposal impacts, cost, schedule, and training.

Many modeling tools are evolving to address evolving modeling standards, but there are significant model integration challenges. Organization should understand how different tools can cover the lifecycle and ensure that the specific tools align with the processes and capabilities of the organization.

Terms and Acronyms

This section provides a list of some of the terms used throughout the paper.

AADL	Architecture Analysis & Design Language
AP233	Application Protocol 233
ATL	ATLAS Transformation Language
BNF	Backus Naur Form

BPML	Business Process Modeling Language
CAD	Computer-Aided Design
CASE	Computer-Aided Software Engineering
CDR	Critical Design Review
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
CORBA	Common Object Requesting Broker Architecture
CWM	Common Warehouse Metamodel
DBMS	Database Management System
DCP	Domain Convergence Path
DoDAF	Depart of Defense Architectural Framework
DSL	Domain Specific Languages
EJB	Enterprise JavaBeans
Erwin	Data modeling tool produced by Computer Associates
IBM	International Business Machines
ICD	Interface Control Document
INCOSE	International Council on Systems Engineering
IPR	Integration Problem Report
IT	Information Technology
Linux	An operating system created by Linus Torvalds
MARTE	Modeling and Analysis of Real Time Embedded systems
MATRIXx	Product family for model-based control system design produced by National Instruments
MBT	Model Based Testing
MDA®	Model Driven Architecture®
MDD™	Model Driven Development
MDE	Model Driven Engineering
MDSD	Model Driven Software Development
MDSE	Model Driven Software Engineering
MIC	Model Integrated Computing
MMM	Modeling Maturity Model
MoDAF	United Kingdom Ministry of Defence Architectural Framework
MOF	Meta Object Facility
MVS	Multiple Virtual Storage
NASA	National Aeronautics and Space Administration

OCL	Object Constraint Language
OMG	Object Management Group
OO	Object oriented
PDR	Preliminary Design Review
PIM	Platform Independent Model
PSM	Platform Specific Model
RFP	Request for Proposal
ROI	Return On Investment
SSCI	Systems and Software Consortium
Simulink/Stateflow	Product family for model-based control system produced by The Mathworks
SCR	Software Cost Reduction
SDD	Software Design Document
SOAP	A protocol for exchanging XML-based messages – originally stood for Simple Object Access Protocol
Software Factory	Term used by Microsoft
SRS	Software Requirement Specification
SysML	System Modeling Language
UML	Unified Modeling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language family (XSL) Transformation
xUML	Executable UML
Unix	An operating system with trademark held by the Open Group
VHDL	Verilog Hardware Description Language
VGS	T-VEC Vector Generation System
VxWorks	Operating system designed for embedded systems and owned by WindRiver

About the Systems and Software Consortium, Inc.

The Systems and Software Consortium, Inc. (SSCI) is a nonprofit partnership of market leaders, government agencies, and academic affiliates. As a consortium, SSCI enables industry and government to co-invest in the development of systems and software processes and capabilities that improve business performance. Members also have access to a team of technical experts whose collective knowledge of best practices and lessons learned gives SSCI the unique opportunity to offer practical advice and proven solutions.

The Consortium is interested in your comments and suggestions. Please send your thoughts and insights to ask-ssci@systemsandsoftware.org.

For more information about the Systems and Software Consortium, please visit www.systemsandsoftware.org