Experiences in Applying Architecture-Centric Model Based System Engineering to Large-Scale, Distributed, Real-Time Systems

THOMAS M. WHEELER

The MITRE Corporation,M/S 1630B, 202 Burlington RD., Bedford, MA 01730
twheeler@mitre.org
(781) 377-7010

MICHAEL D. BROOKS

Northrop Grumman Corporation, 2000 West NASA BLVD., Melbourne, FL 32902
md.brooks@ngc.com
(321) 726-7708

Experiences in applying Model Based Systems Engineering (MBSE) techniques on a large-scale, distributed, real-time system are presented. Challenges and applied approaches with associated lessons learned are described for both technical and socialogical issues. Examples of technical issues addressed include: defining timing requirements and analyzing the ramifications of design choices on these requirements; early execution of the design for behavior analysis to include the use of realistic publish and subscribe mechanisms; managing the rapidly evolving design model so that a large number of engineers can make modifications to it (including the need to incorporate practices of managing software builds); and incorporating legacy software into an MBSE approach. Socialogical issues addressed include: using MBSE to work as a team when members span multiple organizations and technical/business domains; the need for a variety of views of the model based on team members backgrounds (i.e., one type of view such as UML will not meet all needs); information hiding so that there is one logically consistent model, yet parts of it can be hidden from some members of the team (for example, when individuals may not have a security clearence that allows them to see certain aspects of the design); and aspects of MBSE that need to be accounted for when creating project schedules (for example, there may need to be model integration periods scheduled similar to software integration periods). A major source of material for this paper was the authors collaboration on the requirements analysis and preliminary design of the U.S. Air Force's E-10A Multi-Sensor Command and Control system.

Key Words: UML; Model Driven Development; UML Profile for Schedulability, Performance, and Time; Rhapsody, RapidRMA, OPNET

1. Introduction

The U.S. Air Force is developing the next generation airborne Command and Control Intelligence Surveillance and Reconassance (C2ISR) capability called the E-10A Multi-Sensor Command and Control (MC2A) aircraft. The Government and industry have employed model based system engineering (MBSE) principles as an integral part of the program system and software engineering approach. Work to date covers the E-10A requirements analysis and preliminary design phases that includes the development of an executable Unified Modeling Language (UML) system model. The objective of this paper is to document our experiences using MBSE on a large-scale, software intensive project. Included in this paper are the original issues we are attempting to address, the approach used, and lessons learned. We believe the technical and organizational aspects of this program make for a realistic test of MBSE for large-scale development efforts; and that the E-10A challenges, approaches, and lessons learned are applicable to other software intensive development efforts. Unfortunately, the U.S. Air Force recently decided to terminate the E-10A program due to pressing Service budget constraints. It is undecided at this point when or where the emerging E-10A capabilities will transition.

1.1 Project Overview

E-10A technical requirements include providing air and ground surveillance, tracking, and tactical battle management. As shown in Figure 1, the E-10A has three subsystems. The Multi-Program Radar Insertion Program (MP-RTIP) sensor, which was largely designed prior to work beginning on the other subsystems, is an active electronically scanned array radar that concurrently supports both air and ground based missions. The aircraft platform is the second subsystem. This subsystem is based on modifying a Boeing 767-400 extended range aircraft to carry the radar subsystem as well as the Battle Management and Command and Control (BMC2) subsystem. The BMC2 subsystem includes the radios, networking, computers, and software used by operators (both on and off-board the aircraft) to perform missions such as Cruise Missile Defense and Time Sensitive Targeting. Example application level functionality includes track fusion, image exploitation, sensor planning and control, weapon planning, and engagement workflow control. Most of the activities described in this paper are related to work done at the system level across these subsystems, and in particular, the BMC2 subsystem.

Figure 1. E-10A Subsystems

From a business perspective, the work was accomplished across a number of Government and contractor organizations, contracts, and locations. The principle Government acquisition organization was Electronic Systems Center, supported by MITRE, located at Hanscom Air Force Base in Bedford, Massachusetts. The prime contractor was Northrop Grumman located in Melbourne, FL. Other Northrop Grumman sectors were involved as well as other companies such as General Dynamics, BAE, Boeing, and Raytheon. Hundreds of engineers were involved in the total program, although less than 100 were involved in the MBSE aspects.

1.2 Paper Outline

The paper is organized into four sections beyond the introduction. Section II discusses the challenges we expected when we began E-10A development. This covers technical issues as well as those faced when a large number of multi-disciplinary engineers need to collaborate to develop a complex system. Section III discusses key aspects of the MBSE approach used by E-10A. We follow-up with a Section on Lessons Learned, and end with Conclusions – including areas we think need further examiniation.

2. Challenges

Having previously worked large-scale systems development, we knew there were certain technical and sociological (that is people working with people) challenges we were likely to encounter on this program. These challenges were significant contributors toward the decision to use an MBSE approach for both systems and software engineering. This Section discusses the challenges and attempts to leave discussion of solutions to the following Section.

2.1 Sociological Challenges

For programs with the complexity, size, and intended long system life-cycle that was envisioned for E-10A, the sociological issues should not be underestimated. Our experience is that these issues are not typically given the attention they need. Too often the immediate focus is on what technologies should be part of the solution, rather than on how a large number of geographically separated engineers can collaborate to get a job done.

The first conclusion we reached was that there was a significant need for common mechanisms to allow a variety of people to reason about the system. Associated with this is the need to achieve common understanding of key technical decisions by the project team. While this sounds simple and obvious, developing an approach that supports meaningful reasoning can be a significant challenge. Clearly, there is a critical need for quality data at higher levels of abstraction than software source code. Tools and data need to be used that support:

- ☐ Multi-disciplinary collaboration
- ☐ Multi-organizational collaboration
- ☐ Information hiding
- ☐ Multiple levels of abstraction
- ☐ Data currency
- ☐ Data consistency
- ☐ Data correctness

Multi-disciplinary and organizational collaboration is a must for effective systems engineering. Individual groups using their own tools and data put at risk the usefulness of their results. Reasons include the difficulty of people understanding each others results when they use different tools, languages, and data; and the likelihood that these differences result in signficant manual

effort to keep the individual results synchronized. Our experience has been, given the fast-paced nature of system development, that manual synchronization is doomed to fail.

For example, computer/network performance analysts have typically performed a manual translation of the software design into formats that performance analysts/tools can use. Since software engineers are paid to show up to work and build software, the performance engineers quickly end up with a representation of the software that is outdated and decreases the value of their analysis. In addition, if technical management (such as chief engineers and chief architects) are to take action based on the analysis, they end up having to invest signficant time understanding both the software and performance engineers representations of overlapping data. The combination of stale data and different “look and feel“ decreases usefulness which can drive program cost.

Information hiding is an important technique. A commonly discussed reason is the need for abstraction to help people deal with complexity. A second reason for a project like E-10A is the need to truely hide data from certain people, yet expose it to other people. For example, since E-10A is a military project, parts of it are classified at different security levels. While classified data must be protected, we also need to ensure that data is available to as many people to the maximum extent possible, and that seams are not unnecessarily introduced into the engineering data that introduce design mistakes.

The last three bullets shown above are not exciting, but are bread and butter principles that can be diffiult to fully support. Data currency is about knowing whether we are looking at up-to-date information (or at least, being able to map the data to specific builds or milestones). Large programs can take several years to develop, and system sustainment might be done over a number of decades. Left unmanaged, there will be a number of mistaken decisions made simply because the person did not realize they were viewing obsolete data.

Consistency aims to achieve harmony among data both horizontally (e.g., between software components) and vertically (e.g., between system and software abstraction levels). To often, requirements and design artifacts are loosely structured textual statements or graphics (e.g., briefing charts) where there is minimal computer-based enforcement of consistency available. The cost of manual enforcement (not to mention that this sort of work is not something people

tend to like to do – and therefore don't do particularly well) grows as the amount of requirements and design data increase.

Finally data correctness refers to being able to assess the impact of a design choice on the system's ability to meet requirements. Our experience is that software engineering has a long way to go before it approaches the rigor of more established fields such as radar engineering w.r.t analysis. It is striking to attend a software design review and then a radar or radio design review. Analysis presented at a software design review tends to be more subjective and limited than the other disciplines mentioned. Some of this may be the nature of the beast that is software development, but we believe more can and needs to be done to have meaningful understanding of the ramifications of a particular software design choice.

2.2 Technical Challenges

There are a number of technical challenges identified early in E-10A planning that we expected to face based on working other similar type programs. Challenges identified (some of which are not further discussed in this paper) included:

- Location transparency (internal and external to the aircraft)
- Shared resources (E-10A supports numerous users in a dynamic environment)
- Range of hard and soft timing and performance requirements
- Portability of crew (within E-10A and with other Air Force Platforms)
- Reuse of legacy components
- Security
- Openness and extensibility
- System dynamics (dealing with changing environment and faults)

The ones we focused on most in arriving at our need for an MBSE approach were the „moving parts" related challenges. Our experience has been that it is often not until the integration and testing phases that issues with things like system dynamics, performance, and sub-system/component interfaces are discovered. The cost of rework at that point can be prohibitive (not to mention by that time you are often running short on both schedule and funding).

We believe a contributor to inadequately detecting system dynamic (moving parts) type problems in the requirements and design phases are the tools and products used during these phases. Requirements and design products are often static in nature. For example, we typically write

requirements as textual sentences with the key word “shall“ in it; e.g., the System shall track objects that are moving on the ground. Likewise, design artifacts tend to be static pictures of components and their relationships. Given these fixed products, it should not be a surprise that an engineer has trouble forseeing dynamic problems.

3. Approach

The approach we used for the system and software initial design included:

- Architecture-centric focus
- Executable UML-based design models integrated across security levels
- Publish and subscribe communication in the executable design model
- Integrated design and analysis, including use of the UML Profile for Schedulability, Performance, and Time
- Computer enforced linkage of requirements and design
- Early involvement of testers in definition of system threads

3.1 Architecture-Centric Approach

An architecture-centric approach is one where there is an emphasis on the patterns of connections among system components with defined constraints. The identification of critical system views and the accompanying approach for addressing these views can be a powerful mechanism for making people aware of key technical decisions.

E-10A used a variation of the architecture views proposed by Bruce Douglass[1] which are subsystem and component view, distribution view, concurrency and resource view, safety and reliability view, and deployment view. We extended by adding views for functionality and security. The set used by E-10A is shown in Figure 2. While we were committed to producing a robust model of the design, we also thought that emphasis on these views was in itself a useful abstraction and has the benefit of not risking the reader getting lost inside a large UML model. The E-10A/BMC2 software architecture description document we produced ended up having sections for these views where the view diagrams were extracts from the UML system model.

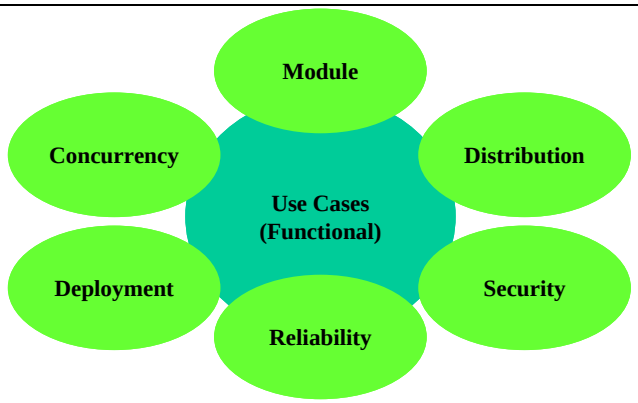

Figure 2. E-10A Architecture Views

3.2 Executable UML Model

The creation of an executable model of the system/software design was a key focus area. We believed such a product would directly address the system dynamic issues discussed in Section 2 to include functional behavior, performance analysis, and interface definition. We developed executable models of the initial design using a UML-based tool from Telelogic called Rhapsody. This model was first populated with the initial static structure, i.e., UML classes representing computer software configuration items (CSCI) and hardware configuration items (HWCI). It was then systemically built-up by the addition of functional behavior to handle a variety of mission oriented system threads.

E-10A used three classes of system threads: mission Verification and Validation (V&V), engineering analysis, and performance analysis each of which had a specific role in the MDSE process. Mission V&V threads were meant to be operationally oriented and typically flowed through many system components in order to verify system behavior satisfied operational requirements. Example E-10A mission V&V threads are “Establish ground track and ID“ and “Support air to air engagement“.

Engineering analysis threads were used by design engineers to examine hard technical problems. These threads typically were limited to interactions among a relatively small number of components. Example of these include “Target Engagement“ and “Develop Combat ID“. Performance analysis threads were intended to evaluate whether the proposed system design could meet timing and performance requirements. These performance threads were typically

subsets of the other two types of threads with allocated performance parameters assigned to each operation.

E-10A ended up developing 9 mission V&V threads, 19 engineering analysis threads, and 18 performance analysis threads for the initial design phase. Note that these threads shared the same functional model components/operations. They just exercise different functional paths through the model.

The E-10A executable model currently has two major levels of abstraction with respect to software. The highest level has a UML class for each of the 43 Computer Software Configuration Items (CSCI). Along with these classes are numerous classes that capture the interfaces provided and used by the CSCI classes as well as classes representing external entities, other E-10A subsystems, and E-10A hardware. The number of CSCIs was driven primarily by the extensive amount of software reuse planned. Each reuse component was assigned a separate CSCI to maximize the control of black box interfaces and to minimize future life-cycle impact with regard to component upgrades.

The second major abstraction level is a drill down of the 43 CSCIs into 132 Computer Software Components (CSC). The current model has these as being a separate „projects“ from that containing the CSCI version although the two projects share common interface classes. This was done in part due to tool maturity issues with code generated ports. We are now in the process of bringing the two models into a single project using UML composite classes and ports. This will significantly improve the computer-enforced consistency between these levels of abstraction.

Rhapsody supports auto-generation of sequence diagrams that serve as traces that capture interactions between system objects as well as internal object method invocations for a specific session of executing the system model. These sequence diagrams can be powerful aids in understanding the system dynamics. The size of the sequence diagrams generated for the E-10A mission threads varied dramatically. Engineering analysis sequence diagrams might be a few pages while some of the mission V&V sequence diagrams were roughly 70 pages long. In addition to the auto-generated sequence diagrams, we also manually produced diagrams for each thread that would highlight only key object interactions. These were typically one to three pages long. The brevity of the manually produced sequence diagrams along with their focus on key interactions made them popular artifacts for engineering reviews.

To give some context to the above numbers, the equivalent source lines of code (ESLOC) expected to be developed for this phase of the E-10A program is approximately 600 thousand SLOC. The current E-10A System Model contains approximately 12 thousand SLOC that supports the functional behavior and interfaces necessary for thread execution.

In order to support the E-10A design tenet for a modular/loosly coupled architecture, we incorporated a publish and subscribe capability into our system models. Topics were created, and objects subscribed and published to these topics. This had the consequence of giving application designers early insight into the loose coupling and performance characteristics associated with this technique.

The model exists across two different Department of Defense security classification levels. It is not uncommon, either due to proprietary or classification issues, that development projects have to confront how to share only limited amount of data among project staff. This information hiding, while necessary, can increase the risk that there are flaws in the resulting design. Two common techniques for handling this at the design stage is to (1) have separate design expressions at the different classification levels, where consistency must be accomplished manually, or (2) keep most of the design information at the highest classification level – thus, keeping people cleared at lower levels unaware of a large amount of data they otherwise should be able to have access to.

E-10A's goals include (1) maximize the amount of data available at the lowest classification level, and (2) maximize the degree of computer-enforced consistency checking of the models across the security levels. We were able to achieve this by using a combination of the UML generalization (a.k.a. inheritance) capability as well as a Rhapsody capability that allows insertion of parts of one Rhapsody project into another. This approach has been a success.

3.3 Integrated Engineering

As discussed in Section 2.1, keeping the data used and produced by designers and analysists consistent is a challenge. For E-10A, this is an important issue as there are a number of size, weight, power, and timing requirements that have to be satisfied in addition to functional requirements.

We decided to address this challenge in part by using the UML Profile for Schedulability, Performance, and Time (SPT)[2]. This was developed by the Object Management Group with this kind of goal directly in mind. In a nutshell, the design is captured using standard UML and then timing and performance characteristics are captured in the model using standardized tag-value descriptors. An analysis tool then extracts this information, performs the analysis, and the results are stored back in the original model. Note, the OMG is currently working on a follow-up to SPT called the Modeling and Analysis of Real-Time and Embedded Systems (MARTE) Profile.

For real-time analysis, we used the RapidRMA tool from Tri-Pacific Software. This has a direct interface to Rhapsody, so we did not have to do manual translation of data from one tool format to another. In addition to the real-time analysis, we also used the OPNET tool from OPNET Technologies for more traditional performance analysis. There is not yet a direct interface between Rhapsody and OPNET, so design was transformed from the UML System Model into an OPNET model and kept consistent manually between the two tools.

The above is an example of a direction that we are trying to achieve where data that is useful to more than one type of project person can be made available to them in a largely computer-enforced consistent manner. Figure 3 shows a big picture view of our goal. E-10A made positive steps in this direction.

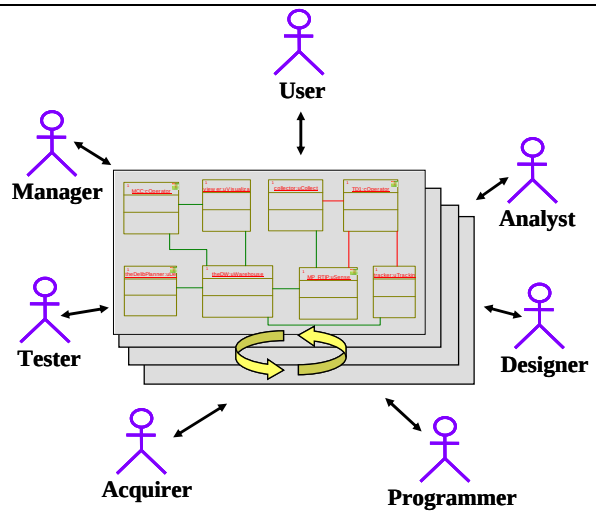

Figure 3. Integrated Engineering using UML Based System Model

For example, our user was able to quickly understand UML diagrams and provide valuable feedback on the emerging design. Analysts and designers were able to, in part, share computer-

enforced consistent data. The Government, as acquirer, gained significant insight into the design by reviewing, and in fact, executing the design models. Furthermore, the testing staff quickly embraced the use of system threads and UML diagrams and were a major presence in reviewing the design.

Finally, it is a key point that the MDSE process did not "replace" the contractor's CMMI System Engineering Process, but rather enhanced the process with robust model driven practices and tools as shown in Figure 4. The associated architecture modeling concept is an enabler to perform comprehensive analysis of the system trade space early in the product life cycle for both the E-10A development and its integration with other C2ISR platforms

Features		*Traditional System Engineering Techniques*		*E-10A Model Driven System Engineering Techniques*
Requirements-driven	☑	Text based tools (DOORS/SLATE) used as Requirements Management repository	☑	Integrated Text/UML tools provide central repository for complete traceability
Integrated visual model repository	☐	Heterogeneous mix of visual modeling artifacts using various languages and tool formats; must be manually linked	☑	Architecture Model serves as common repository for system architecture and related SE artifacts – "visual bandwidth"
Shared knowledge base	☐	Ad hoc domain-specific and architectural rules scattered in various documents	☑	Architecture Model is augmented with domain-specific and architectural integrity rules
Common SE/SWE language	☐	Heterogeneous mix of proprietary and standard languages	☑	UML is common language for specifying SE and SWE work artifacts
V&V	☐	Mostly manual V&V at during formal integration	☑	Early Test/Customer participation starting with preliminary design

Figure 4. MDSE Features

4. Lessons Learned

This Section captures experiences we gained in applying MBSE during the requirements analysis and initial design phases of E-10A development. Section 4.1 discusses areas that we believe are strengths while Section 4.2 identies areas that remain challenges in applying MBSE during these phases. In some cases, a topic is discussed as both a strength and challenge.

Some members of the E-10A had experience in various aspects of the approach discussed in Section 3. But in general, this was the first time many of us had worked together on a project and

used the set of techniques in one cohesive activity. It is fair to characterize the E-10A effort as including a substantial amount of discovery and learning. That makes the fact that E-10A leadership is pleased with the results all the more positive as it is reasonable to expect even more efficiency as we repeat these techniques in the future.

We realize the lessons learned in this paper are subjective in nature. That is, these statements are opinions. The E-10A program is not yet mature enough, having just completed initial design, that we believe we can perform a more quantitative assessment. That will more naturally occur once a capability is developed and tested.

4.1 MBSE Experiences - Strengths

Strenths of the E-10A MBSE approach include:

- ☐ Models were understandable by a broad range of stakeholders
- ☐ Improved early insight into system dynamics
- ☐ Improved insight in performance/timing via integrated design and analysis
- ☐ Early realism of design choices
- ☐ Improved design consistency
- ☐ Tool enforced consistency between unclassified and classified data
- ☐ Integrated tool set allowed for controlled, managed efforts across a large geographically separated team
- ☐ Improved design compliance due to tight coupling of requirements and system model elements
- ☐ Early involvement of test community via definition of system threads

We were pleasantly surprised at the range of stakeholders who found it valuable to review the UML-based executable system model. Diagrams such as class, state, activity, and sequence diagrams were able to be understood by non-software engineers after just a brief explanation.

For example, our user (typically U.S. Air Force officers) reviewed our use cases and mission V&V threads. They were very supportive of the information being presented in that manner. In particular, they found that sequence diagrams helped them understand how the engineers were thinking about user – system interactions. Review of these diagrams led to productive discussions on these interactions. While we are pleased with the involvement of the user, a future enhancement we want to make is to integrate our executable system model with GUI displays that

will present information in formats that are more reflective of the operational environment. The user could then focus on the GUI windows - including stimulating the executable UML design model and observing the effects.

The acquirer (U.S. Air Force Electronic Systems Center assisted by MITRE) also gained an unusual amount of insight into the design. This was due in part to having ready (updates often weekly) access to the emerging models. In many cases, acquirer engineers performed independent assessments by executing the model themselves and even extending the model to look at alternative design concepts. During design reviews, the Government was able to have executable models on their individual laptops and were able to independently look at various aspects of the design model while the audience, as a whole, could follow along in a more traditional projection of parts of the model on a large screen. Because MBSE tends to make information more consistent and readily available (e.g., the model is the design, not hundreds of seperate textual documents or briefing charts), we believe it was a major factor in an overall high and productive level of collaboration across the team.

We were pleased with the early insight we achieved in understanding system dynamics and also the ability to assess the design's ability to meet performance and timing requirements. Key to this improvement in understanding was having an executable model and also having tight coupling between the system model and performance and schedulability analysis tools.

The use of auto-generated sequence diagrams that act as trace recordings of a specific model execution instance was helpful for engineers to understand the concurrent processing that occurs in large distributed systems. This was hindered somewhat by our implementation of a publish and subscribe mechanism in the system model which caused some artificial sequential execution actions. Still, seeing the flow reveal itself as the model was stimulated with different events is a signficant improvement over what is traditionally done during requirements analysis and initial design.

The number of elements and the level of design captured in the E-10A model identified certain scalability challenges to using executable sequence diagrams. As previously mentioned, some of our sequence diagrams reached 70 pages in length. Likewise, some of our sequence diagrams had tens of lifeline objects shown. Rhapsody has two nice features to manage the number of

lifelines shown. First, there is an environment lifeline that acts as the surrogate for all objects that are involved in the execution, but are not represented as a specific lifeline.

Another useful feature for managing size of sequence diagrams is to use composite classes and their parts. There is a simple property in Rhapsody that allows the designer to have the parts either shown on the diagram or have the composite classs contain the parts' behavior in a black box manner. This can help control the number of lifelines needed on a diagram. A potential enhancement we've raised with Telelogic is to have a property set that would allow the model execution engineer to select whether internal method invocations are shown or not. This would be a powerful black box tool for cutting down on the number of pages of a sequence diagram and would still have the model execution be correct.

We were an early proponent of applying the UML Profile for Schedulability, Performance, and Time on E-10A given that the goals of the OMG effort were similar to goals we had[3]. We have found the SPT Profile, as implemented in the RapidRMA and Rhapsody tools, to be helpful. Sequence diagrams are a natural for defining specific timing scenarios, and the language developed for the SPT Profile was intuitive to learn and appropriate for E-10A conditions. A nice feature of RapidRMA is that you can go from analyzing simple to complex timing scenarios by adding additional sequence diagrams to the situation being studied. A challenge we faced was that RapidRMA would perform a rigorous error checking of the entire system model prior to doing a timing analysis. While this is helpful in one sense, in practice there were parts of our system model that were in development that were not yet ready for that level of scrutiny. It would be helpful if RapidRMA (and analysis tools in general) allowed a narrowing of scope to be identified for performing error checks.

We also used the OPNET tool for performance analysis. This tool has a rich set of libraries for various architectures which makes it attractive to use. Unfortunately, it does not yet support the SPT Profile, nor does it have a direct bridge to Rhapsody. This caused us to have to manually keep the design model and performance model consistent which is something we'd like to avoid in the future. Even given the need for manual consistency, we still found that the OPNET work raised important questions with our design and resulted in modification to that design.

4.2 MBSE Experiences - Challenges

Challenges of the E-10A MBSE approach include:

- ☐ Need to modify the team's existing processes
- ☐ Getting buy-in from system engineers who have not used UML before
- ☐ Executable models need care from hands-on software developers
- ☐ Integrating the work of many engineers within the model
- ☐ Model compilation time as the model grows in size
- ☐ Infrastructure choices impact on early application design
- ☐ Consistency between models at different security levels
- ☐ Consistency between model abstraction levels
- ☐ Purchase cost of tools and training

Companies doing business with the U.S. Department of Defense are highly encouraged to have defined, repeatable processes. On one hand, we found during this activity that it is not necessary to completely reinvent an organization's process. For example, the interface information one normally wants by the time of an initial design review doesn't necessarily change – just how that information is captured and perhaps presented. On the other hand, we found the need to define a detailed process for creating and updating the system model. The more precise we made the exit criteria, the better the results we got from the large number of people working the project.

In addition to needing to carefully document expectations of the engineers, we also found the need to perform specific, detailed prototyping of the approach before unleashing it on the entire team. One quick way to turn people off is to tell them to do something and then not have the tools and processes allow for that to occur. When planning what prototyping should be accomplished, areas that we found needed attention included: number of people updating a product, number of sites needing access to the data, the number of security levels, the number of model elements, and the number of abstraction levels. At first our focus was on tools, their functional capabilities, and interoperability. Over time, we realized that scaleability is a critical aspect to focus prototyping on.

One of the more interesting parts of this effort was the involvement of talented system engineers in the MBSE activity. Many of these engineers have been involved in the susccessful development of some incredible systems. Yet they generally had little experience with UML or

executable software models. Since we had the philosophy that the model is the design, this could have been a problem. In general, we found the system engineers to at first remain a bit distant from the development of the executable UML-based system model. Over time, they ended up engaging in the work and directly reviewing and improving the model. A project would benefit by reducing the amount of time it takes to get some of the system engineers more involved in the MBSE work.

There are some technical issues with using MBSE to the extent we did. We found that to produce the executable model of our initial design, we needed a number of hard core, hands-on software developers involved to care and feed the model. We would not expect to reduce the number of programmer to zero for the requirements analysis and initial design phases, but MBSE tool vendors need to think of providing simpler interfaces that are aimed at system engineers if they want to play in the systems engineering market.

Another technical issue is how to structure the model and manage the model so that a large number of engineers can update it. One possibility we have thought about employing is scheduling model integration time similar to the integration periods that are scheduled for integrating various software components. This probably will help, but since much of initial design work is to define boundaries and interfaces, we have not easily resolved the challenge of having a number of people updating the model simulataneously.

A strength of our work was the use of a publish and subscribe mechanism in the executable initial design model. This caused the application designers to early-on think about their interactions with other applications via this pub/sub pattern. Overall, this was worthwhile, but we do wonder whether, during the early stages of system development, application level designers (and for that matter systems engineers) should be freed from knowing and being impacted by such an infrastructure design choice. It is a mixed blessing that they had to sort through the various pub/sub messages that early in design.

Abstraction is a key tool for allowing engineers to think about the system being developed. Keeping the data consistent across abstraction levels can be a challenge. As mentioned earlier, E-10A is now updating their system model to use composite classes and ports with the goal of the tool providing significant support for keeping the abstraction levels synchronized.

5. Conclusions and Way Ahead

The application of MBSE with robust, UML-based executable models for the E-10A requirements and initial design work was a success. This approach mitigated risks associated with large hetergeneous organizations attempting to collaborate to build a leading edge large-scale distributed real-time system.

Techniques such as having a strong architectural vision, robust executable design models, integrated design and analysis, and computer enforced horizontal and vertical (as well as across classification or proprietary boundaries) consistency are significant performance enablers.

Challenges associated with this approach exist. It takes training and committment to a more robust engineering approach to be able to reach a state where the model is the design. Achieving consistency requires a high degree of coordination across the team.

There are several areas we would like to see improved in the future. Better vendor support for integrating tools that support a range of development activities (not just code generation) is needed. Lowering the expertise required to construct executable models of system design would be valuable. While full Model Driven Architecture (MDA) might always require talented software developers, executable models aimed at capturing system high level design should be able to be created by non-programmers. Improved tools and techniques that allow a large number of people to simulanteously update the design model – yet still have a high degree of consistency checking – would be very beneficial.

Acknowledgements

The most satisfying part of applying MBSE on E-10A was having a large number of people participating in our MBSE approach. It was a true test of the technology and the various experiences and opinions only served to strengthen the approach. It is impracticable to list each by name here given the large number of people involved. Regardless, we deeply appreciate the creative and hard work these people performed to make the activity successful. We want to espcially highlight the role management played in this; without their support and patience, the MBSE approach would have died a quick death.

References

[1] B. Douglass, „Real-Time Design Patterns“, ISBN 0-201-69956-7, Addison-Wesley, 2003, p. 68.
[2] UML Profile for Schedulability, Performance, and Time Specification, OMG, Jan 2005.

[3] T. Wheeler, „An Early Look at the UML Profile for Schedulability, Performance, and Time for Engineering Large Scale Airborne C2ISR Platforms“, SIVOES-SPT Workshop on the usage of the UML Profile for Scheduling, Performance, and Time, Toronto, Canada, May 2004.