Model-Based System Development for Managing the Evolution of a Common Submarine Combat System

Steven W. Mitchell
Lockheed Martin MS2
9500 Godwin Dr.
Manassas, VA 20110 USA

Abstract- **Managing the evolution of a complex product family that is deployed and maintained in multiple variants on various platforms using traditional systems engineering tools and processes is a significant challenge. An example is managing the evolution of a common combat system across a fleet of submarines. Due to the realities of budgets and operational scheduling, multiple versions of the product must be managed for each ship: the currently deployed version, the upcoming tech refresh version, and future versions in planning and development. Adding to the complexity, the product family has variations for each class and flight of submarine, individual ships may vary in capabilities and equipment from the flight baseline, ships within a flight are upgraded at different times driven by maintenance availabilities, and each periodic upgrade may introduce new functional capabilities as well as updated software and hardware as the combat system evolves.**

To streamline this task, an integrated model based systems engineering process is being developed around a collection of tightly coupled models built in SysML and UML. This paper provides a preliminary description of the structure of those models, and places it in the context of related research.

General Terms: Systems Engineering, System Evolution

Additional Key Words and Phrases: Product Family, Model Based Systems Development, SysML, Maintenance

I. INTRODUCTION

In order to maximize capability and interoperability while minimizing development and maintenance costs, a common submarine combat system (CS2) is being deployed across the entire US and Australian submarine fleets. However, those fleets are composed of multiple classes of submarines, each of which typically has variations in physical implementations of key equipment such as sensors and weapons, along with different physical arrangements in spaces such as combat control centers, equipment spaces, etc. Beyond this, ship classes are typically composed of sub-classes known as "flights" which are sequential groups of ships that are built to the same design baseline. That baseline may vary to greater or lesser degrees from other flights within the class. An example of this inter-flight variation is the introduction of vertical weapons launchers in Flight 2 of the LOS ANGELES class. In addition, operational needs can cause variations ranging from small to quite substantial even within a flight. A prime example of this is SSN 23[1], which differs from the other two submarines in the single-flight SEAWOLF class by an additional one hundred feet in length and a thirty percent increase in displacement to accommodate special operational requirements.

This host platform variability means that the CS2 is in reality a product family with different variants for each class, flight, and, sometimes, individual submarine. While the core components that make up the product for a given release of the CS2 are common across all host platforms, there are host-specific variants that have to be developed, integrated, installed, supported, and then replaced with the next version on a regular upgrade cycle.

Although it started its evolution as a loosely connected collection of legacy military systems hosted on traditional military-unique computational platforms, the federated CS2 is now based largely on a Commercial Off-The-Shelf (COTS) computational and networking platform. The software of the CS2 also contains massive amounts of COTS software[2] in the form of operating systems, database systems, graphics libraries, etc. While utilizing COTS in this way brought the cost of this submarine combat system down about an order of magnitude over the previous generation[3,4], it brought with it the challenge of managing the continuous, unsynchronized evolution of those COTS products. The CS2 acquisition program handles the hardware side of the COTS management

---

1 United States Navy Fact File
http://www.navy.mil/navydata/fact_display.asp?cid=4100&tid=100&ct=4

2 In this usage, COTS includes Free or Open Source Software (FOSS) as well.

3 The Advanced Rapid Capability Insertion (ARCI) program reduced the cost of the AN/BQQ-10 sonar system by a factor of six in Development and Production costs, and a factor of eight in Operating and Support costs according to Reference [1] p. 317. The CS2 applies the same process and approach to the overall submarine combat system as ARCI does to the submarine sonar system.

4 RADM H. I. Fages, the sponsor for the development of the AN/BQQ-10 sonar, which is part of the CS2, stated, "ARCI development costs were one-tenth (1/10) the cost of BSY-2 and ship set cost was less than one-thirtieth (1/ 30)" [2]

problem with biannual baseline updates that allow each ship-set of equipment to be state-of-the-market when it is installed aboard ship. This process is referred to as Technology Insertion (TI). These TI baseline changes occur in even years, leading to the nomenclature TI10, TI12, etc.

To avoid COTS software obsolescence issues and to provide incremental improvements in system capability, the application software running on this COTS hardware platform is also upgraded biannually. For historical reasons this process is called Advanced Processor Builds (APB) [3]. The APB baseline updates occur in odd years, leading to the nomenclature APB09, APB11, etc. This TI/APB cycle is in reality a double-helix collaborative spiral development life cycle [4] where the hardware and software spirals are one year out of phase with each other.

Given that there is an unavoidable coupling of the APBnn and TImm updates, fleet-wide baselines installed aboard ship are referred to as TImm/APBnn. The overlap of the TI and APB update cycles leads to an annual change in baseline installations. Thus submarines that are upgraded in 2011 will receive TI10/APB09, while those upgraded in 2012 will receive TI10/APB11, etc. These annual baselines must go through the full systems integration process [5] to ensure that the system installed aboard ship is operationally suitable, effective, and interoperable.

Compounding the complexity of managing the evolution of the COTS components of the CS2, the primary purpose of the APB process is to integrate new algorithms and software technologies to enhance the operational effectiveness of the submarines. For the most part the enhanced capabilities ride on the steady TI technology refreshes, but occasionally they require significant new hardware such as new or improved sensors. As such, new capabilities are typically evaluated on one submarine before being incrementally introduced to some subset of the fleet. This approach adds additional variants to the complexity of managing the evolution and deployment of the CS2 product family.

The engineering problem of managing the evolution of the CS2 goes beyond simply tracking bills of materials for the various baselines. Since this system is installed on a submarine, power, cooling, mass properties, and physical layout are tightly constrained and must be tailored to the idiosyncrasies of the various host platforms. Other materials characteristics that have safety implications for the crew under casualty conditions, such as the presence of various hazardous materials, must be tracked. These systems process classified information, and so must be documented, tested, and certified to the appropriate standards for information assurance (IA). The CS2 has direct impacts on safety of ship, and is a core part of a weapons system: both of those things require additional documentation, verification, and certifications. The CS2 interfaces with larger military command, control, communications, computers, and intelligence (C4I) networks, which entails interoperability testing and certification.

Beyond these engineering requirements, as military platforms the submarines hosting the CS2 are subject to the various national acquisition management processes. The CS2 development program is required to provide informational inputs to those processes, and significant portions of that information have to be extracted periodically from the CS2 product family integration process.

All of these requirements together imply that a product family model of the CS2 encompasses at least twenty dimensions of information including Operational Capabilities; Requirements; the Conceptual Data Model, the Logical Data Model; the Physical Data Model; the software bill of materials; the hardware bill of materials; network topology; network addressing and routing; software allocation; hardware placement; hardware shock and acceleration characteristics; hazardous materials characteristics; weight; power; cooling; host platform; IA parameters; Integration and Test; and time.

II. A Conceptual Model

Before proceeding with the description of the submarine product family and the proposed design of a submarine data model, a conceptual data model is introduced. There are three processes that change the data describing the common submarine combat system: Technology Insertion, Systems Engineering, and Integration and Testing. If, conceptually, there existed a single data base for all submarine CS2 data, defined here as the System Data base, then any one of these three processes could cause change in one or more submarines and thus change the data in the data base. This is shown graphically in Fig. 1. The three processes can be represented as three services that cause change to the actual systems that, in turn, must be reflected in the System Data Base.

The design challenge is twofold: to design a service for making changes to the system data base to reflect the changes in the systems themselves and to create a data extraction and report generation service that is capable of producing standard reports and also respond to new queries. This is shown in Fig. 2.

In section 3, related research in complex product family modeling is discussed. In section 4, the product family that is described as “Physical Systems” in Fig. 2 is discussed, and the submarine data model is presented (the System Data Base in Fig. 2). Finally, in Section 5 the remaining steps in the design of this system are outlined.

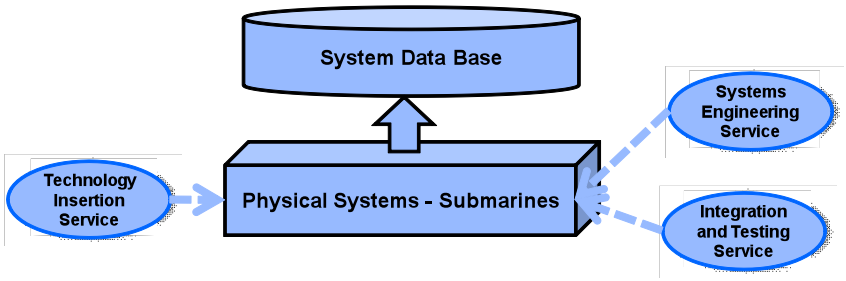

Fig. 1. Conceptual Model of System Database

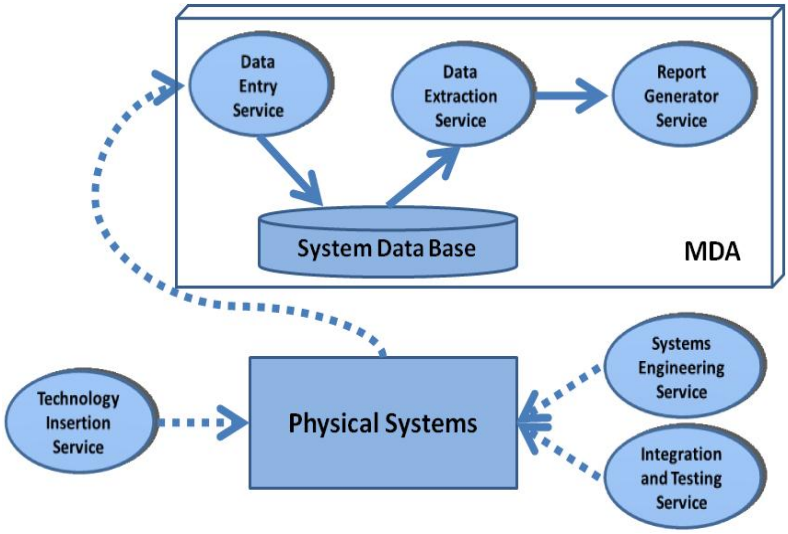

Fig. 2. The Conceptual Model

III. Product Family and Legacy System Evolution Modeling

Over the past two decades considerable research has been conducted in the area of modeling product families. Reference [6] provides a concise overview of this research. Much of the early work focused on deliberate design of a product family [7], although some authors have studied the problem of creating a product family from a collection of closely related products. One approach taken was the Generic Bill of Materials, introduced [8] to address customized manufactured-to-order products. This technique was generalized from the manufacturing domain to the functional and technological domains [9], and further generalized [10] to capture business processes associated with mass-customized production.

Several authors have tackled the complexity of representing product families by inventing new modeling languages [11], or other representations [12], [13]. Callahan [14], [15] takes a particularly interesting approach to this, making a clear distinction between product family architecture and product variation. Both [16] and [12] are careful to minimize the amount of redundant information required to describe each product variation through efficient representations. Part of their concern is to manage the overall size of the data sets associated with large product families, which if the model were duplicated for each variant would grow exponentially for a product platform such as an automobile or an airplane that has several variants, each of which has many options that can be ordered in many combinations. This concern for efficient reuse [17] of the data sets involved is common to mass customization problems [18].

A larger concern addressed in this research is avoiding the problem of authority when dealing with multiple copies of a complex data set such as that defining the product family core platform: whenever multiple copies of data are kept, errors can occur. When instances

of the same data disagree, how does one determine which is authoritative? The best solution is to avoid potential inconsistencies due to duplication of information: if a representation requires duplicated data, authoritative data must be uniquely identified, and all copies clearly identified as such. This is non-trivial when complete product data sets are duplicated and then modified for each member of a large, rambling product family.

An adjacent problem to modeling product families is managing the evolution of a complex legacy system. Based on decades of attempts to evolve software-intensive systems with a disappointingly low success rate, [19] proposes that system understanding through systems architecture within the larger enterprise context is a key factor in economically evolving legacy systems into product families to achieve greater functionality and maintainability. This emphasis on architecture for system understanding is a common thread across the field, also reflected in [20], among others.

Ulrich [21] provides an overview of standards work being done in this area by the Object Management Group (OMG) Architecture Driven Modernization (ADM) task force. Ulrich also sees the first step in modernizing legacy systems to be the understanding of the architectural and implementation structures of those systems through modeling.

Veering away from standards-based architectural modeling, [22] approaches the problem of extracting architectural information through design structure matrices and design rule theory. Their objective is not simply to increase understanding of legacy systems by revealing their structure to enable engineers to make informed design decisions, but to provide tools for evaluating proposed design changes as well. They analyze two web application servers to demonstrate the accuracy of their approach. However, the primary utility of this approach seems to lie in retrospective analysis of system evolution, rather than forward evolution support.

Reference [23] takes a slightly different perspective. They argue that conventional configuration management systems are not sufficiently powerful to manage architectural evolution for complex problems such as product families. They propose instead a new architecture modeling and evolution paradigm that integrates fine-grained configuration management into the architectural modeling tool itself. Acknowledging that this approach violates the fundamental assumption of conventional configuration management that the management process should be strictly separated from its object, they substantiate their argument by implementing an integrated architecture modeling and configuration management environment called Mae and demonstrating its effectiveness on three sample problems. Interestingly, some of their internal representational constructs are similar to the constructs of [15], although they are addressing superficially different problems.

Another, rather more ambitious approach is taken by [24]. They propose to do away with manual modification of legacy system software components in favor of automated code generation in a Model-Driven Program Transformation approach. While recognizing that constructing the underlying tools is a serious technological challenge, they argue that domain-specific model interpreters are the key breakthrough to allow an integrated toolset to reverse-engineer the modules of a legacy application to construct a system architecture model. Domain experts would then modify that model to meet new requirements. Their integrated toolset would finally generate the transformed application.

IV. The Submarine C5I Engineering Data Model

Historically, management of the CS2 baselines and upgrade process has been done using a variety of documents and databases controlled by various Integrated Product Teams (IPTs) that manually coordinate their efforts. These IPTs – staffed from the CS2 system integrator, numerous independent developer organizations in industry, government, and academia, Navy procurement and engineering agencies, and the shipyards – collaborate in a complex systems integration process. In order to streamline this process and to reduce the amount of engineering labor dedicated to coordinating the various documents and other products used to support the CS2 product family and its multitudinous variants, an integrated Submarine C5I Engineering Data Model (SCEDM) is being constructed.

Fig. 3 suggests some of the complexity of this SCEDM. It is important to understand that this diagram merely hints at the full dimensionality of the problem. As noted in the Introduction, every submarine in the fleet has some TImm/APBnn baseline installed, with class and/or flight and/or hull variations from that baseline. While that baseline of the CS2 is being supported, the next baseline for each submarine is being developed, as is the baseline beyond that. All of this data must be tracked for two submarine fleets comprising six classes, eleven flights, and almost ninety boats.

As indicated around the edges of Fig. 3, this SCEDM does not exist as an end in and of itself. Rather, it exists to support the engineering process of developing, manufacturing, installing, and supporting the CS2 tailored for each boat. Many more products than the figure shows will be derived from the model including hardware and software baselines for manufacturing to generate new installation kits, cabling data packages to support the shipyards in installing new hardware (collectively represented by the Top Level Interface Functional

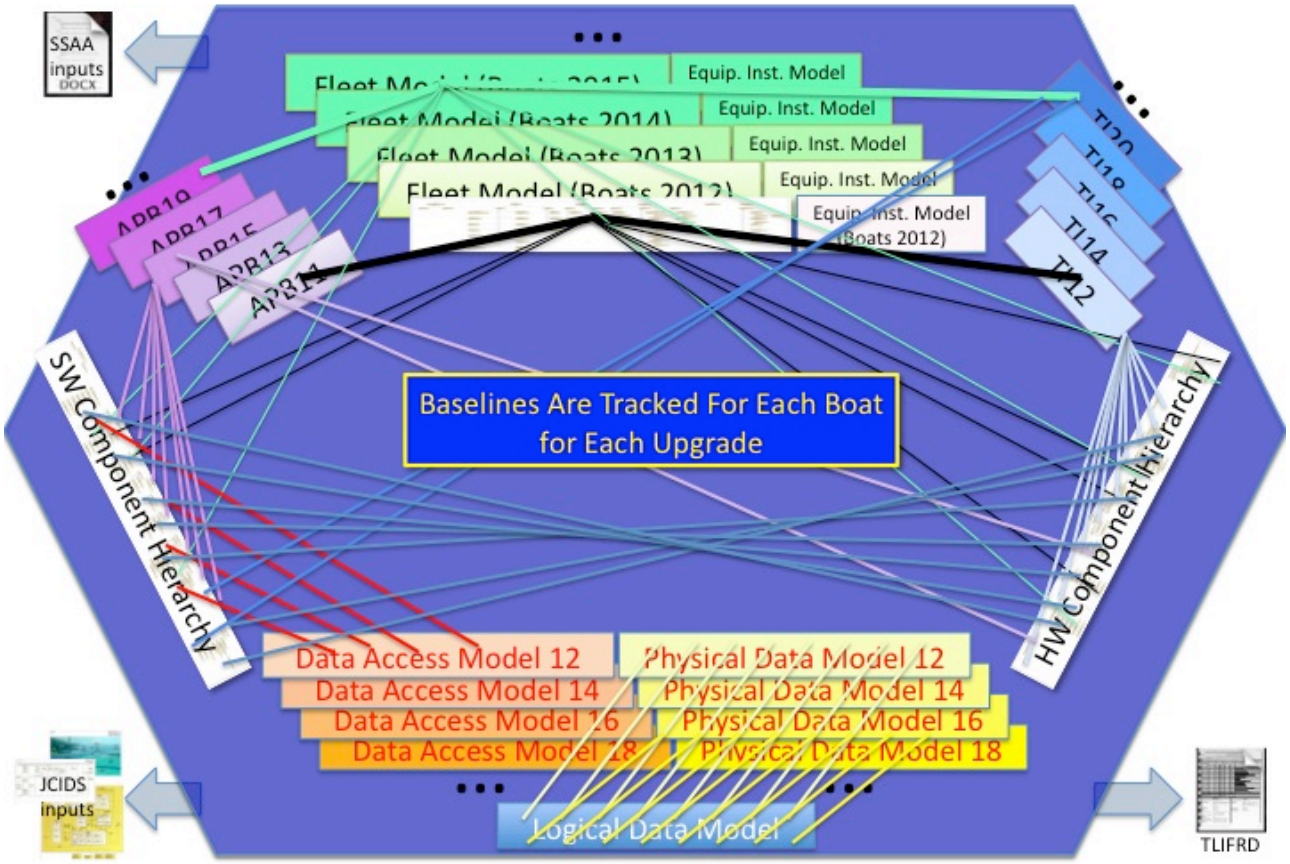

Fig. 3. A Simplified View of the Data Dimensionality of the SCEDM.

Requirements Document (TLIFRD) in the lower right corner of the figure), information assurance documents to support operational certification of individual submarines after installation of a new CS2 baseline (collectively represented by the System Security Authorization Agreement (SSAA) inputs in the upper left corner of the figure), systems requirements verification matrices (SRVM), and DoDAF architecture reports to support both the acquisition and the evaluation of new operational capabilities (collectively represented by the Joint Capabilities Integration and Development System (JCIDS) inputs in the lower left corner of the figure).

The information dimensional complexity of this model along with the size of the configuration space (roughly fifty interface groups defined between up to fifty subsystems distributed across approximately three hundred computers grouped into nine different network enclaves) of the CS2 dictates that industrial-strength tools be used to develop this model.

Where many earlier researchers have defined their own idiosyncratic representations for product family modeling (as discussed in Section 2), the objective of this project is to support the Fleets. That means the SCEDM must be developed using current industry best practices for Model Based System Development, which in turn means using current industry standard languages such as UML[5] [25] and SysML[6] [26]. Given that the model must capture a broad range of information ranging from CORBA IDL and WSDL used to define inter-subsystem

[5] The current specification of UML is at http://www.omg.org/spec/UML/2.2/

[6] The current specification of SysML is at http://www.omg.org/spec/SysML/

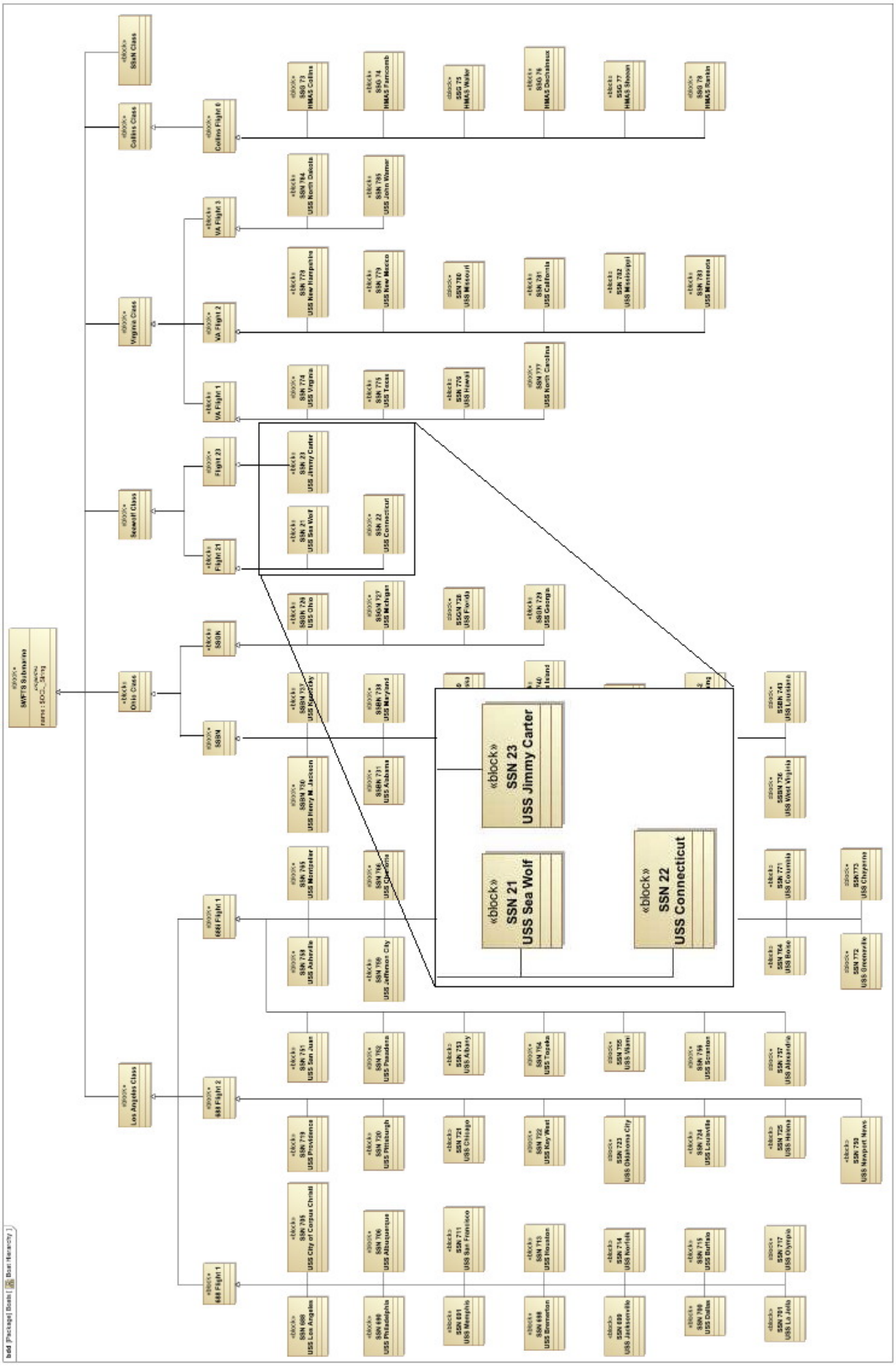

Fig. 4. Inheritance Hierarchy of Submarines Employing CS2.

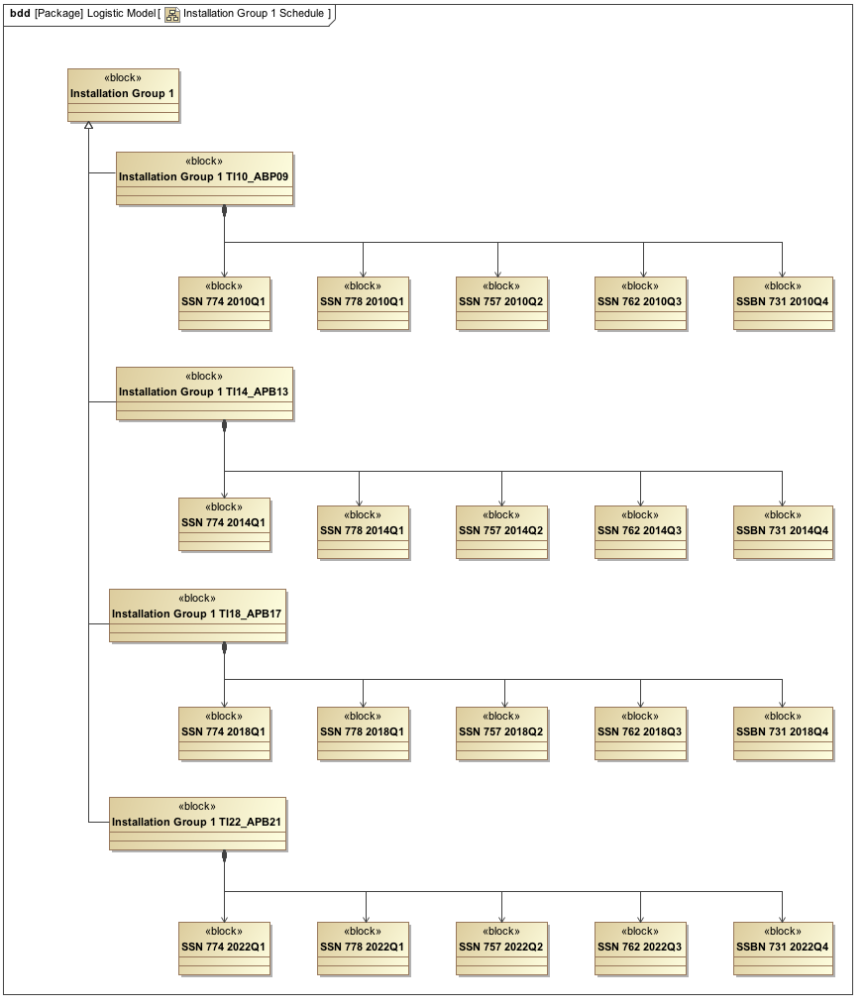

Fig. 5. Notional Upgrade-Group Structure

software interfaces to physical hardware specifications including mass, power, cable runs, etc, a mixture of UML to represent the software domain and SysML to capture the hardware and systems engineering domains has been chosen.

Following [15], the SCEDM leverages inheritance to minimize information redundancy. As shown in Fig. 4, an inheritance hierarchy is defined for all of the submarines supported by the CS2. The second tier of the hierarchy is the ship Class, with the Flight below that. Individual ships chain down from the Flight. CS2 baselines are assigned as high up in the hierarchy as possible, with individual class / flight / ship variations overriding the inherited configuration baseline where necessary.

Due to both operational and fiscal constraints, not all submarines in the fleet are upgraded every year. Instead, groups of ships are scheduled for upgrade each year on a rolling basis. This process is represented by *Upgrade Groups*, as shown in Fig. 5. The timings of these TImm/APBnn installations are driven by operational deployments and new construction, and are coordinated with shipyard or pier side ship availabilities.

Since the CS2 evolved after many of the submarines in Fig. 4 were already at sea, back-fitting this common submarine combat system to multiple classes of ships has driven some configuration compromises. For example, some ship classes have network-enabled navigation radars, while others use a serial interface between the radar and the navigation subsystem. This means that on some ship classes in some baselines the radar belongs to

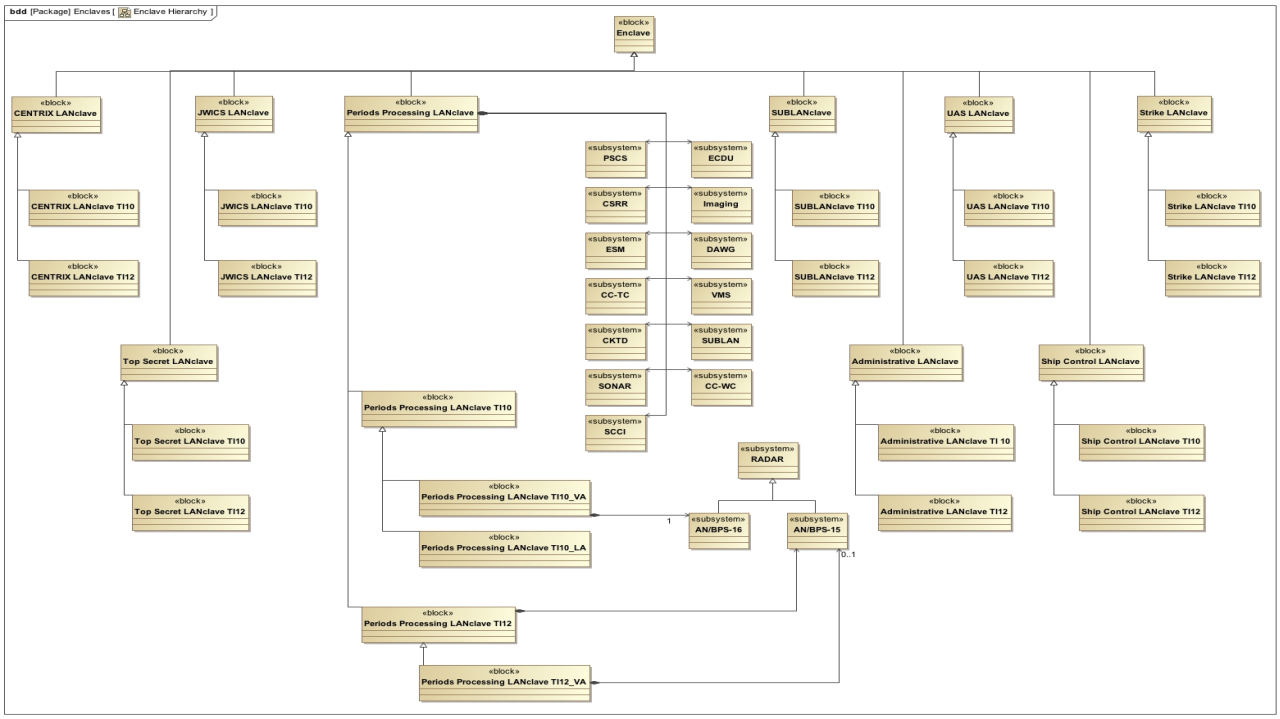

Fig. 6. CS2 Network Enclaves with Navigation Radar Assignments Migrating Across Network Enclaves with Evolving Baselines

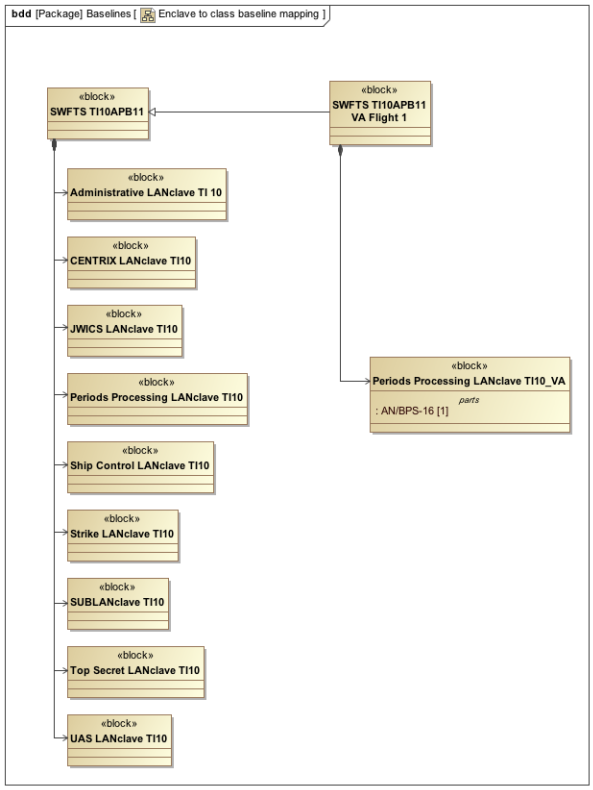

Fig. 7. Organization of CS2 Baseline by Network Enclave, with Class Variants

one network enclave, while for other classes of ships the radar is part of a separate enclave.

Further complicating the problem, future baselines include network-enabling the older navigation radars, which means that at that point in time the radar will move from one network enclave to another. This kind of system evolution is captured in Fig. 6. In the center of the diagram is the *RADAR* subsystem class with two variants, the AN/BPS-15 and the AN/BPS-16, which get assigned to different enclave baselines in different upgrade cycles.

Another example of how the SCEDM leverages inheritance to minimize data redundancy is show in Fig. 7. The network enclaves and their component subsystems are collected together in the *TI10APB11* baseline class, but the basic *Periods Processing LANclave TI10* is specialized in the *TI10APB11 VA Flight 1* subclass to include a sub-classed *Periods Processsing LANclave TI10_VA* that includes the AN/BPS-16 navigation radar. This structure enables the SCEDM to achieve the same kind of representational power and efficiency seen in [15] while using an industry-standard modeling language.

An important benefit of the SCEDM is automating requirements traceability throughout the product family. Currently CS2 requirements are maintained in a database, but the allocation of requirements to design elements is a matter of manually entering that

information into the database, then integrating it into a variety of other products such as software requirements specifications, hardware requirement specifications, interface specifications, and other engineering documents. This is an inefficient process even for developing a new product, but when managing an evolving product family it becomes a serious impediment. Typically, it is requirements changes that drive the introduction of variants, including the TImm/APBnn baselines of the CS2 product family. Those requirements changes ripple through the product design in ways that are difficult if not impossible to predict. Without automation, each change must be manually tracked through the various documents of the design. That is a labor-intensive process, particularly since any errors in the requirements can have very expensive impacts on the implementation of a new member of the product family, dictating careful, redundant inspection.

As shown in Fig. 8, in the SCEDM the requirements are directly linked to the appropriate elements of the design. Thus when a requirement changes, the tool hosting the SCEDM can generate a report identifying everything in the system that is touched by the ripples of that change, sidestepping a great deal of error-prone manual labor. When new requirements are introduced into the model or old design elements are eliminated, similar reports can flag the lack of requirements linkages, clearly identifying the gaps that need to be addressed by the systems engineering team.

As the hardware model changes due to TI refreshes or the software model changes with APB revisions, similar tool-assisted requirements traceability analysis can quickly identify all of the impacts that must be analyzed. The linkages are not limited to requirements: elements of the software model may have dependency linkages to elements of the hardware model, which in turn may have linkages to budget allocations, etc. All of these linkages can be traced by the tool hosting the SCEDM, and reported to facilitate change impact analysis.

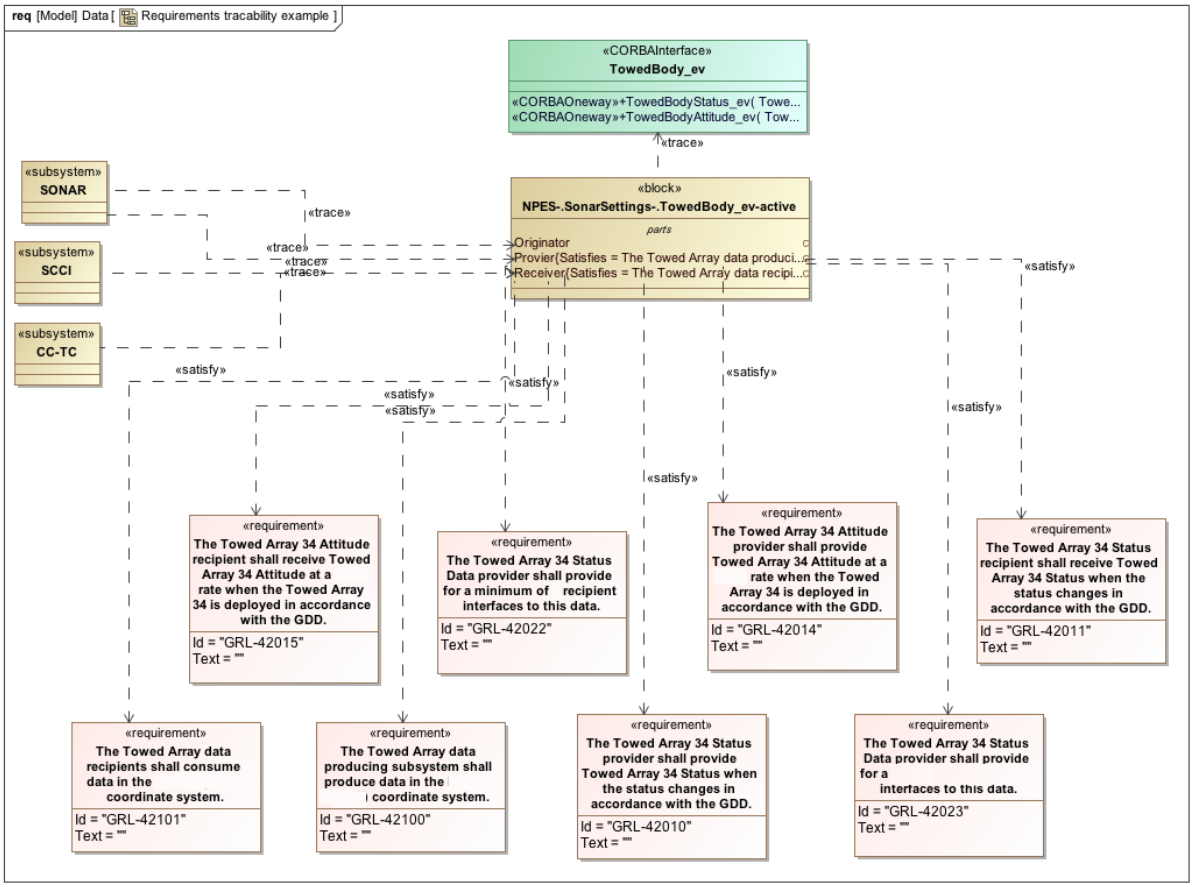

Fig. 8. SCEDM Requirements Traceability

V. Current Work and Conclusions

The structure of the SCEDM described in this paper is preliminary and addresses only the System Data Base portion of the overall problem defined in section 2. As more details are worked out and additional dimensions of the data are filled in, the detailed structure of the SCEDM will evolve. It is likely that filling in some of the details of this evolutionary product family model will stretch the bounds of the current UML and SysML language standards, and potentially will require inventing extensions to those languages. This will be done in collaboration with the developers of the respective standards so that those extensions are defined in the spirit of the languages, and can be incorporated in future versions of those evolving standards.

Only a few information dimensions have been even partially addressed in the SCEDM prototyping that has been accomplished to date. The next dimension that will be explored more fully is that of requirements. Several hundred inter-subsystem interface requirements have been imported from the current DOORS repository, but while some preliminary investigation has been conducted (as seen in Fig. 8, the best level at which to link the requirements into the model has not yet been determined. One problem that has to be addressed is that the requirements are currently all at one level, and include significant amounts of implementation direction as well as performance requirements. The performance requirements and implementation directions need to be de-interleaved, and the requirements need to be structured into a hierarchy to better facilitate aligning them with the other model dimensions. This de-interleaving will also isolate the requirements from the implementation, significantly reducing requirements churn and resulting cost. In addition, that hierarchy needs to be extended from the current set of inter-subsystem interface requirements up the tree to the top level system requirements to provide full traceability and to support the national acquisition and evaluation processes.

Similarly, all of the CORBA IDL interfaces that are part of the Physical Data Model for current CS2 baseline have been imported into the prototype SCEDM. Efforts are currently underway to determine how and where best to link those interface implementations into the model to support the evolving TImm/APBnn baselines.

In parallel with the work on requirements, the Conceptual and Logical Data Models of the CS2 must be constructed. These data models will provide a framework to anchor the Physical Data Model created from the imported CORBA IDL and evolved with the TImm/APBnn product variants. The Conceptual and Logical Data Models will also help link the product implementation model with the higher-level operational modeling that supports the acquisition and evaluation processes.

Another way that the SCEDM might increase the efficiency of the CS2 product family life cycle is by more tightly coupling integration, test, and verification with the requirements generation process. Today use cases, scenarios, and various dynamic modeling artifacts (activity diagrams, sequence diagrams, etc) are used to generate lower level design requirements from the high-level capabilities requirements levied on the program. The derived requirements are entered into the requirements database and then manually allocated to various design elements, but the analysis that supported the requirements generation process is often lost in the process. In the future that analysis and the corresponding products may be captured in the SCEDM and associated with the corresponding requirements. When integration and test comes around on the spiral those sequence diagrams, etc. could be used support development of test and verification plans, reducing the communications gap between requirements engineers and test engineers and closing the requirements-to-verification cycle.

Another positive impact the SCEDM could have would be to leverage the change impact analysis discussed at the end of section 3 to focus integration and test efforts. Tracing the ripples of change through the model identifies those parts of a new baseline implementation that are different from previous variants. Since these areas were not tested in the integration of any previous system variant, they are most likely to hide new bugs. Where these changes do not represent new or improved capabilities (such as the replacement of a COTS hardware component with a newer model with the same specifications, or the upgrade of device drivers for a previously used hardware component), they are excellent candidates for targeted regression testing. Where they represent new or improved capabilities, they are a natural focus for intensive integration and test. Either way, the overall cost effectiveness of integration and test should be improved.

In addition to building the SCEDM, an ecology of tools must be defined and developed around the model to make it efficient for the many IPTs that collaboratively evolve, implement, and support the CS2 to access those portions of the model that they need without requiring all of those engineers to become proficient in the tools, languages, and schemas used to build the model. This is the problem of efficiently providing the Data Entry, Data Extraction, and Report Generation services described in section 2. Most of those engineers are currently using various Microsoft Office products to manage their specialized nexi of the overall information space that will be subsumed into the federated SCEDM. It is anticipated that various web services will be built around the

UML/SysML model to provide familiar interfaces to those nexi and to eliminate the cost of climbing the learning curve that would be necessary if every engineer supporting the CS2 were required to become expert in a UML/SysML modeling tool.

Promising research in this direction [18] defines a dialect of XML called platform product eXtensible Markup Language (ppXML) specifically to support lifecycle modeling of platform-based product families to support mass customization. While this XML dialect does not map precisely to the problem at hand, there are some useful ideas in that work that may be adapted to the SCEDM tool ecology.

Reference [23] also has potentially interesting implications for the future SCEDM tool ecology from the perspective of configuration management of the SCEDM model. However, the choice of the industry-standard representational languages UML and SysML for the CS2 problem precludes the freedom to invent a new representation such as used in Mae, which builds upon xADL 2.0 [27], an academic XML-based architecture description language. Furthermore, major UML/SysML modeling tools are evolving rapidly towards using relational database management systems for heir underlying data stores. It is not yet clear if or how these concepts might work together. Still, there are some interesting ideas in this work that warrant further consideration.

ACKNOWLEDGMENTS

This research was supported by NAVSEA contract N00024-06-C-6272. The model design described in this paper was the result of vigorous discussions with Professor Alex Levis of George Mason University, with Mr. Greg Bussiere, Mr. Mark Hassel, and Mr. Rob Pollack of the Naval Undersea Warfare Center, and with Mr. Sanford Friedenthal, Mr. Brandon Gibson, and Mrs. Danielle Robinson of Lockheed Martin. The useful results are the fruits of that collaboration, while the errors remain my own.

REFERENCES

1. Ford, David N., and John T. Dillard. “Modeling open architecture and evolutionary acquisition: implementation lessons from the ARCI program for the Rapid Capability Insertion process.” *Proceedings of the Sixth Acquisition Research Symposium: Defense Acquisition in Transition* 2 (Apr 2009): 207-235.
2. Fages, H I. “Submarine programs: A resource sponsor’s perspective.” *The Submarine Review*, (July 1998): 53–59.
3. Jacobus, P., P. Yan, and J. Barrett. “Information management: the Advanced Processor Build (Tactical).” *JOHNS HOPKINS APL TECHNICAL DIGEST* 23, no. 4 (Jan 2002): 366-372.
4. Boehm, B., and P. Bose. “A collaborative spiral software process model based on theory W.” *Proceedings, ICSP* (Citeseer) 3 (1994).
5. Sage, Andrew P., and Charles L. Lynch. “Systems integration and architecting: An overview of principles, practices, and perspectives.” *SYSTEMS INTEGRATION AND ARCHITECTING* 1 (Jan 1998): 176–-227.
6. Timothy W. Simpson. “Product platform design and customization: Status and promise.” *Artif. Intell. Eng. Des. Anal. Manuf.* (Cambridge University Press) 18, no. 1 (Jan 2004): 3-20.
7. Erens, F., and K. Verhulst. “Architectures for product families.” *Computers in Industry* (Elsevier) 33, no. 2-3 (1997): 165-178.
8. Hegge, H. M. H. and Wortmann, J. C. “Generic bill-of-material: A new product model.” *Intl. J. of Production Economics* 23 (1991): 117-128.
9. Erens, Frederik-Jan. “The Synthesis of Variety-Developing Product Families.” PhD Dissertation (Jun 1996)
10. Jiao, J., MM Tseng, Q. Ma, and Y. Zou. “Generic bill-of-materials-and-operations for high-variety production management.” *Concurrent Engineering* 8, no. 4 (Nov 2000): 297-322.
11. Medvidovic, N., D. S. Rosenblum, and R. N. Taylor. “A language and environment for architecture-based software development and evolution.” *Proceedings of the 21st International Conference on Software Engineering (ICSE’99)*, (Nov 1999): 44–53.
12. Jiao, J., and M. M. Tseng. “An information modeling framework for product families to support mass customization manufacturing.” *CIRP Annals-Manufacturing Technology* (Elsevier) 48, no. 1 (Jul 1999): 93-98.
13. Xuehong Du, Jianxin Jiao, and Mitchell M. Tseng. “Product family modeling and design support: An approach based on graph rewriting systems.” *Artif. Intell. Eng. Des. Anal. Manuf.* (Cambridge University Press) 16, no. 2 (2002): 103-120.
14. Callahan, Sean M. “Relating functional schematics to hierarchical mechanical assemblies.” *Proceedings of the fourth ACM symposium on Solid modeling and applications*, (1997): 229-239.
15. Callahan, Sean M. “Extended generic product structure: an information model for representing product families.” *Journal of Computing and Information Science in Engineering* (ASME) 6 (Nov 2006): 263-275.
16. Callahan, Sean M. “Extended Generic Product Structure: An Information Model for Representing Product Families.” (Nov 2005): 1-68.

17. Briere-Cote, Antoine, Louis Rivest, and Alain Desrochers. “Adaptive generic product structure modelling for design reuse in engineer-to-order products.” *Comput. Ind.* (Elsevier Science Publishers B. V.) 61, no. 1 (Jul 2010): 53-65.
18. Huang, George Q., Li Li, and Xin Chen. “ppXML: A generic and extensible language for lifecycle modelling of platform products.” *Computers in Industry* (Elsevier) 59 (Aug 2008): 219-230.
19. Weiderman, Nelson H., John K. Bergey, Dennis B. Smith, and Scott R. Tilley. “Approaches to Legacy System Evolution.” (Dec 1997): 1-42.
20. Gerber, A., E. Glynn, A. MacDonald, M. Lawley, and K. Raymond. “Modelling for Knowledge Discovery.” *Proceedings of the 2004 EDOC Workshop on Model-Driven Evolution of Legacy Systems (MELS)*, (Aug 2004): 1-5.
21. Ulrich, William. “A Status on OMG Architecture-Driven Modernization Task Force.” *Proceedings of the 2004 EDOC Workshop on Model-Driven Evolution of Legacy Systems (MELS)*, (Sep 2004): 1-4.
22. LaMantia, Matthew J., Yuanfang Cai, Alan D. MacCormac, and John Rusnak. “Evolution Analysis of Large-Scale Software Systems Using Design Structure Matrices & Design Rule Theory.” (Apr 2007): 1-11.
23. Roshandel, R., A. V. D. Hoek, M. Mikic-Rakic, and N. Medvidovic. “Mae---a system model and environment for managing architectural evolution.” *ACM Transactions on Software Engineering and Methodology (TOSEM)* (ACM) 13, no. 2 (Apr 2004): 240-276.
24. Zhang, Jing, and Jeff Gray. “Legacy System Evolution through Model-Driven Program Transformation.” *Proceedings of the 2004 EDOC Workshop on Model-Driven Evolution of Legacy Systems (MELS)*, (Aug 2004): 1-5.
25. Gomaa, Hassan. Designing Concurrent, Distributed, and Real-Time Applications with UML. Addison Wesley, (2000).
26. Friedenthal, Sanford et al. *A Practical Guide to SysML: The Systems Modeling Language*. Morgan Kaufmann, (2008)
27. Dashofy, E. M., Van Der Hoek, A., and Taylor, R. N. “An infrastructure for the rapid development of XML-based architecture description languages.” *Proceedings of the 24th International Conference on Software Engineering (ICSE2002)*, ACM, New York, NY, USA. (2002): 266-276.