2012-10-10

Formal methods for architecture model assessment in systems engineering

Giammarco, Kristin.

8th Conference on Systems Engineering Research
March 17-19, 2010, Hoboken, NJ

Paper #1569270961

Formal Methods for Architecture Model Assessment in Systems Engineering

Kristin Giammarco
Naval Postgraduate School
Department of Systems Engineering
Monterey, CA

Abstract

In both the public and private sectors, systems engineering analysis studies are used to inform a range of decisions, from selecting among low level design alternatives to conducting high level portfolio management. These studies are conducted to help various stakeholders make and justify decisions that have a lasting effect on a system throughout its lifecycle. The quality of the data used in these analyses can have a strong influence on the outcome of these decisions. The research described in this paper investigates the application of formal methods to architecture model quality assessment. These methods can be used to create sets of measures and criteria relevant in the systems engineering problem domain that can be tailored for different events, analytics, and decision points. Using formal methods, stakeholders can decompose and express architecture data quality expectations unambiguously, and in a way that is abstract and independent of tool.

Introduction

This paper summarizes the research conducted by NPS and CERDEC S&TCD in an operationally relevant venue provided by Command, Control, Communications, Computers, Intelligence, Surveillance, Reconnaissance (C4ISR) On The Move (OTM) to mathematically express architectural elements, relationships, and some rules that can be used to develop customizable sets of criteria for some hard-to-quantify terms often used to characterize architecture models, such as “integrated” and “complete”. In the field of software engineering, formal methods are extensively used to quantify such terms as they apply in the software domain (e.g. Darnton 2007). The use of formal methods is less ubiquitously found in systems engineering practices, especially in lifecycle phases that precede system specification and design, where interpretations of the terms can especially vary widely among stakeholders. The scope of this paper focuses on the application of formal methods for defining and assessing architecture model quality characteristics using example terms found in the systems engineering domain[1]. A formal model-based systems engineering method is applied using the Vitech CORE systems engineering tool schema (Vitech 2004) as an experimental reference and tool. The paper concludes with a summary of the concepts and next steps to be taken in future work.

Problem and Need Statement

Needs for short term systems engineering analysis studies often emerge quickly with little time to gather and integrate the needed

[1] Note that there is a distinction between assessing the quality of a system design, and assessing the quality of the data used to model that system. This research focuses on the latter.

data to produce recommendations for senior leadership that have an empirical foundation. Analysis results and corresponding recommendations are only as credible as the source data that was used to generate them. Knowing this, expert analysts often spend more time on the chore of gathering and organizing source data than conducting the analysis itself. The problem is compounded by the short turnaround times required for recommendations, often resulting in the collection and tailoring of data in a way that is optimized for answering the burning analysis questions of the day, and the sacrifice of the data's suitability for reuse in future studies with differing scopes. In many cases, the quality of the data is not known until problems emerge after the effort is complete. Such cases pose the risk of erroneous recommendations due to faulty underlying data. Acquisition Programs of Record (PoR), Advanced Technology Objectives (ATOs), and other longer term programs have more time and resources to deal with data quality issues, but these larger organizations still continue to lack sufficient formalism needed to empirically assess their vast quantities of data for quality and suitability for use in various types of analyses. Whether a project is a 90-day study or a multi-year PoR, reams of data are often produced, used, and discarded after it has served its immediate purpose or when they have otherwise become obsolete.

A capability is needed to enable the assessment of architectural models for quality and suitability for use (and reuse) in various types of analyses.

Definition of Architecture

An architecture is defined by the DoD Architecture Framework v1.5 as "the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time" (DoDAF v1.5 2007). An architecture model is typically developed to ensure that the concerns of stakeholders are addressed by the system(s) under development. For systems that are deployed without going through a formal acquisition process, the architecture is often retrospectively documented to enable a better understanding of the system and its relationships to other systems, and how to co-evolve them. The research herein applies to both types of architecture models.

An integrated architecture is defined as "An architecture consisting of multiple views or perspectives (operational view, systems view, and technical standards view) that facilitates integration and promotes interoperability across capabilities and among related integrated architectures" (CJCSI 3170.01G 2009 and DoDI 4630.8 2004). DoDAF 2.0 elaborates on this definition, defining "integrated" to mean that "data required in more than one instance in architectural views is commonly understood across those views" (DoDAF v2.0 2008). One intended application for this research is the formal specification of terms like "integrated" to remove ambiguity from a natural language definition such as that given above, so that architecture assessment activities have a scientific, objective, and tool-agnostic specification to use.

An architecture framework "establishes terms and concepts pertaining to the content and use of architectural descriptions" [IEEE 1471, 2007]. Architecture frameworks are employed to create, communicate consistent architecture descriptions. Some examples of frameworks are the Zachman Framework, US Department of Defense (DoD) Architecture Framework (DODAF), Ministry of Defence Architecture Framework (MODAF), and the Software Engineering Institute (SEI) Views and Beyond Framework. All frameworks embrace the concepts of having views, viewpoints, and stakeholders. All frameworks also have the notion of striving for consistency between the views. The research described in this report is independent of framework.

Quality Attributes of Architecture Models

Independent of tool, framework or format, architecture models have certain quality aspects associated with them. Table 1 illustrates some characteristics of data in general that can be applied to data present in architecture models. These characteristics are important factors in determining the suitability of data for use in analysis and decision support. One might assess architecture data against such suitability characteristics, for example, to determine the suitability of the data's ability to support an analysis of a given scope, or to provide an indication of the confidence level that stakeholders should have in a data set upon which recommendations and decisions will be based.

The DoDAF includes "completeness", "consistency", and other qualitative characteristics as desirable attributes of an architecture model, presenting them as principles but providing no specific criteria that can be used as a checklist of sorts to determine whether in fact an architecture model meets the expectations for these attributes (DoDAF v1.5 2007). Qualities such as those in Table 1 are often expressed as desired system characteristics, and techniques are available for associating them with quantitative parameters when designing a system. For example, Quality Function Deployment (QFD) (Maier 1995, Verma et al. 1995 and Verma et al. 1998) is a technique that maps qualitative stakeholder requirements (e.g. flexibility) to quantitative design dependent parameters (DDPs) (e.g. number of different I/O port types). The system design is assessed for the qualitative requirements by measuring the values of the associated quantitative parameters. As DDPs are used in QFD to guide system design, measurable expressions can likewise be used to quantify ambiguous stakeholder requirements for architecture model elements, attributes, and relationships. More specifically, a set of quantitative logical assertions can be associated with each qualitative characteristic, to formally define (perhaps as a computable metric) what exactly is meant by each of these natural language terms. Formal methods are available to systems engineers for use in creating and tailoring unambiguous specifications for quality aspects of an architecture model.

Table 1: Example Data Suitability Characteristics[2]

Qualitative Characteristic	Natural Language Definition
	The degree to which...
Completeness	…the data requirement is met.
Clarity	…the data is clear and unambiguous to all (not just the authors).
Stability	…the data is stable in the face of changing data requirements.
Reusability	…the data can be used again by others and in future events.
Consistency	…the data is consistent with other models covering the same scope.
Data Portability	…the data can be integrated with data from different data models.

Formal Modeling of Architecture

An architecture can be modeled informally using such tools as viewgraphs, word processing documents, drawing tool diagrams, and unlinked spreadsheet tables. Because there is no programmed logic linking the data

[2] Definitions derived from (West & Fowler, 1999) and (Maier & Rechtin, 2002)

in and among these tools, opportunities to develop inconsistencies in such informally modeled architectures exist. Users of the architecture data are continuously engaged in the manually intensive effort of carefully coordinating the inevitable changes to the data. Capability to perform analyses (especially quick ones) is extremely restricted because it takes time to describe the data for different scenarios and keep the data in multiple views synchronized. To address this problem, many commercially available tools (INCOSE 2009) have been designed for use for architecture development and analysis efforts. These tools address what Fred Brooks describes as "accidents" (Brooks 1987), advancing the technology available to architects for the development of their work products. This research pertains to the underlying "essence" of architecting, which is tool-agnostic: how can systems engineers formally specify architecture model suitability characteristics, and then assess any architecture for possession of those characteristics?

Towards this end, formal methods (Wing 1990), (Luqi & Goguen 1997), (Berry 1998) are used in this research to mathematically represent some example rules and constraints associated with an example architecture model quality aspect. First order logic is used to demonstrate how some architectural element relationships may be modeled and used in a notional data suitability assessment. The larger questions being asked in this research are as follows: Can natural language architectural principles such as those advised in the DoDAF be described more formally, in a way that can be used and reused again and again? How can the suitability of data in various architecture tools used by systems engineers be assessed to support a confidence level in an analysis, and in the resulting recommendations to decision makers? How can models of systems and system of systems (SoS) be assessed as the pace of their growing complexity increases? These are questions about the "essence" of systems architecting.

The formal methods used in this research are demonstrated using the Vitech CORE systems engineering tool schema, which was used to enable convenient experimentation. The CORE schema is based on a data model that has elements in common with many other architecting tools, although these elements may be called by different names in different tools. A *data model* provides a standard for use by different organizations and systems so that the same data has the same meaning to everyone involved. (Maier & Rechtin 2002) describes a data model as that which "specifies data that a system retains, and the relationships among the data".

One can describe a data model in structured natural language as a series of statements, as follows:

"Architectures are composed of Components."
"Components are built from other Components."

Of course, the terms used in the data model must be explicitly defined, since broad terms like "component" mean different things to different people.

The same information that is stated in natural language above can also be graphically conveyed, as in Figure 1. (More detailed graphical notations also exist for representing data models (e.g. UML, SySML)).

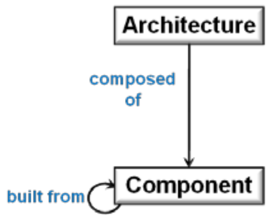

Figure 1. A simple data model. The boxes are elements, and the labeled arrows describe relationships among the elements.

The same information can also be conveyed using expressions in first order predicate calculus:

X = {x |x is an architecture}
Y = {y |y is a component}
Composedof (x, y): x is composed of y

The predicates can be further elaborated on with assertions about the quantities of elements in the relationship:

$$\forall x \in X \ \exists y \in Y \ \ Composedof\ (x,y)$$

(All architectures are composed of some components.)

Assertions such as the example above can be specified as suitability criteria required by architectures for use in a given purpose. For example, an analyst can use the above expression to unambiguously convey "I can't use this architecture in my particular analysis unless it has some components in it." This condition might be the case if the analyst is doing systems level analysis and requires more than an operational architecture. Together with other expressions, the formal expression for containing some components may constitute one analyst's definition of "complete". This is a simple example – one may wish to be more specific about the exact components or types of components required – but it communicates the idea of using formal logic to be very clear about what is needed. Following such a method, one could choose appropriate criteria based on the intended use of the architecture model, and collect them together into a formal specification. For instance, a selected set of quantified expressions could be used to specify exact levels of architecture "maturity" expected before a milestone decision point, to specify exactly what information is needed to answer a given analysis problem, or to establish clear and unambiguous criteria for certification of an architecture. Since such formal specifications are independent of architecture development methodology, programming language, and tool, they enable a fundamental approach that can be applied in any architecture development methodology, programming language, or tool capable of implementing them. It is a vehicle for stakeholders to be more specific about what they fundamentally need, without specifying how it should be delivered.

"The essence of mathematics is its freedom."
– George Cantor (1845-1918)

Example Application

The formal notation used in this research can be applied to very practical quick turn analysis of actual deployed or deploying architectures. A relevant example is the integrated architecture of the C4ISR On The Move (OTM) Event '09 (E09), executed over the summer of 2009. C4ISR OTM E09 provided a unique venue, actual architecture data sets, and SoS-level problems that inspired much of this research. One such SoS-level problem concerned the tracking and updating of architectural data about the E09 systems, personnel, schedule, and experiment logistics. Informal tools such as viewgraphs, word processing documents, drawing tool diagrams, and several independent spreadsheet tables were used to model the E09 Integrated Architecture. Although the manual coordination methods used met the basic functional needs of its users, the activity of tracking, assigning, and deconflicting platforms, equipment, drivers, operators, spare parts, and so on, occupied a substantial amount of their time. The level of effort needed to keep conflicts from occurring in the rapidly changing data was taxing on the technical staff, whose primary function was to execute critical activities for meeting the E09 Campaign Goals. When a conflict did occur (such as an airship being committed to two different experiments that overlapped in schedule), the consequence was costly in time and dollars. To address this issue, PM C4ISR OTM is migrating towards the use of an integrated database to enable the automated assessment of its data for early indications of such scheduling and logistics conflicts.

One specific example of a high-level quality aspect that the PM desires to continuously improve is its “Execution Readiness”, a quality that is often informally discussed and subjectively assessed. Some of the factors that influence how ready its architecture model is for virtual and physical execution in experimentation are listed in Table 2.

Table 2: Sample Measures for Assessing Event Execution Efficiency

Qualitative Characteristic	Natural Language Definition
	The degree to which...
Component Assignment Efficiency	…physical components[3] are scheduled in a manner that makes maximum use of them without creating a schedule conflict.
Component Availability	...a physical component is on hand and operating satisfactorily when needed in an experiment.
Component Supportability	...spare and replacement parts for a component are on hand when needed in an experiment.
Personnel Assignment Efficiency	…participating people are scheduled in a manner that makes maximum use of them without creating a schedule conflict.
Personnel Availability	...a participating person is on hand and able to work when needed in an experiment.
Range Availability	...a range is open and cleared for use when needed in an experiment.

These measures are still qualitative in nature, and can be further broken down into more quantitative measures. Table 3 illustrates sample quantitative measures for the first qualitative measure listed in Table 2 (Component Assignment Efficiency). The values associated with each quantitative measure can be used to assess the qualitative measure, either independently or together in mathematical formulas designed to compute high level metrics.

Table 3: Sample Measures for Component Assignment Efficiency

Measure	Units
Number of components with no experiment assignment	# per Event
Number of experiments with no components assigned	# per Event
Average length of time during an Event for which an available component has no experiment assignment	days
Average length of time during an Event for which an experiment is missing a needed component	hours
Total number of component conflicts among all experiments in the Event	# per Event
Average number of component conflicts in each experiment	# per experiment

For the purposes of this example, a formal expression useful for at least one of the example measures from Table 3 is presented. The expression is written abstractly, to illustrate how first-order logic can be used to provide a consistent definition for terms used across various architectures to define measures. The names of the variables may change from one architecture to the next, but the logic is the same.

[3] e.g. ground vehicles, air assets, radios, routers

The following rules can be used to specify the logical definition of a component conflict among scheduled experiments:

$X = \{x \mid x \text{ is an experiment}\}$
$C = \{c \mid c \text{ is a component}\}$
Uses(x, c): x uses c
Conflict(c): c has a conflict
s: a start date
e: an end date

$$\forall x_1, x_2 \in X\ ,\ \exists c \in C\ ,\ x_1 \neq x_2$$
$$((x_2.s \leq x_1.e)\ \wedge\ (x_1.s \leq x_2.s)) \vee$$
$$((x_1.s \leq x_2.e)\ \wedge\ (x_2.s \leq x_1.s)) \wedge$$
$$\big(\big(Uses(x_1, c) \wedge Uses(x_2, c)\big) \rightarrow$$
$$Conflict(c)$$

This expression essentially states that if two different experiments (called "Verification Events" in CORE) overlap, and these two experiments intend to use the same component, then the component in question has a scheduling conflict.

Thus, any experiments (a.k.a. tests or events) that are shown in the architecture model to use the same components (or some other constrained physical resource) during the same window of time will be flagged well before the conflict actually occurs. This rule, and others like it, can be used to assess the suitability of the current set of data for use in SoS-level experiment planning activities. The expression is abstract enough to be tailored and implemented in any architecture tool with the facilities to relate the necessary information and execute the script.[4]

The example above illustrates one set of criteria that an architect may use in conjunction with others to support assessment of a qualitative attribute called "Component Assignment Efficiency". More measures (such as those in Table 3) would need to be logically defined or computed to fully assess "Component Assignment Efficiency", and additional measures (such as those in Table 2) would need to be defined to assess the higher-level "Execution Readiness" quality.

Such specifications can be used by systems engineers to build custom checklists or automated scripts for checking data sets against predefined criteria, tailored to meet their definitions of "correctness", "consistency", "completeness", "maturity", and other characteristics that can help analysts to determine the suitability of the architecture for different uses (detailed design, analytics, logistics, decision points, etc.). These rules may be generated by the architect, or provided by another stakeholder or governance authority.

If an architecture data set fails to meet the criteria specified, a report of the failures can provide the architect or other stakeholders with specific, actionable information for improving the model's quality. These quantitative results could also be mapped to the qualitative reds, ambers and greens that are often used to provide dashboard-level status information to project managers and senior leaders, and tracked over time.

It is important to note that not all rules will hold true for all architectures all the time – the intent is to tailor the set of rules for the architecture model and its expected state of maturity. For example, one may wish to use the rule:

x: a component
y: an operational node

$$\forall y \in Y\ \ \exists x \in X\quad Implements(x, y)$$

meaning "For every defined operational node, there is at least one component that implements it."

If the above rule is implemented in an architecture modeling tool, instances of operational nodes that do not comply can be

[4] It should be noted that the expression was written without the aid of a model building tool, and that additional constraints may capture details that may be needed to produce the exact desired result during implementation (see future work).

flagged. The example rule above would ensure that all operational nodes in an architecture model are implemented by one or more physical components. Any operational nodes that do not have a corresponding physical component could be reported so that appropriate action can be taken. However, what if this check was applied after the operational nodes were modeled, but before the corresponding physical components were selected and modeled? It would not be a useful rule, since the architecture model is not "mature" enough to supply any meaningful results for this check. This rule is too restrictive to be imposed early in architecture development, but can be a very appropriate check to run prior to some decision point requiring a physical architecture, at which knowledge of operational nodes that have yet to be implemented in physical components would be important. It thus makes sense to have different collections of rules that apply in different phases of architecture development, and doing so can be a useful indication of architecture "maturity" with respect to specific expectations as the architecture advances through decision point reviews.

Conclusions

This research investigated how formal methods can be used by systems engineers before detailed system design to specify architecture model constraints, and associate them with quality characteristics to assess an architecture model's suitability for different uses. Using mathematic- and logic- based expressions, systems engineers can enable the specification of architecture model quality requirements independently of solution-oriented products and tools, thus enabling stakeholders to develop an objective set of criteria that can be used to assess and potentially compare architecture models developed by different organizations and different tools. The formal logical expressions can be used to unambiguously describe architectural elements, their relationships, and quantitative constraints that apply to these relationships. The benefits of applying such formal methods to systems engineering was demonstrated using notional examples as well as an example from an actual systems engineering program at CERDEC, in particular demonstrating their use in assessing architecture model suitability for use in different types of analyses (e.g. SoS-level experiment planning and decision point review).

One of the essential findings that resulted from constructing formal logic statements about architecture models is that these expressions can provide systems engineers and integrators of complex systems and SoS (such as governments) with the ability to create clear, unambiguous, methodology- and tool-agnostic sets of criteria or instructions that can be tailored for different events, analytics, decision points or other uses. These sets of criteria can be used to formally define what precisely the government (or other customer/stakeholder) means by terms like "integrated", "complete", "mature", and other quality attributes for which many principles, guidelines and policy statements exist in natural language, but for which formal specifications are largely absent. Armed with formal, objective criteria for architecture model quality, stakeholders would have sets of tailorable, clear, testable requirements and expectations that can be used to assess and measure the degree to which architecture models comply with expectations for integrity, maturity, completeness, and other quality attributes over time. Formal methods can be use for setting and validating architecture model quality criteria, rather than assuming the criteria will implicitly be met by the systems engineering process or in the tools used in the architecture development and validation effort. Since the rules can be stated independently of methodology, tool, and programming language, there is no restriction imposed on organizations and companies requiring them to change from their preferred tools. Architects and tool vendors could use these criteria to conduct pre-assessments of their architecture models and increase their quality before delivery, formal assessment, or use in decision points. From a vendor's perspective, having definitive criteria from a customer could help them develop tools and models that better meet customer needs, and provide more exact descriptions of (and estimates for) "follow-on" work needed to close the gap between their current architecture model and the objective model expected by the customer.

Among the uninitiated, formal methods have a reputation for being expensive, requiring special expertise, being time consuming, and being cryptic (Bowen and Hinchey 1995). The level of investment in their use should be set appropriately according to the complexity and criticality of the system to which they are applied. In the PM C4ISR OTM example, investment in formal methods is being taken as a measure to reduce excessive experimentation delays, which are costly in time and talent. Increasingly complex systems are becoming increasingly difficult to model informally. Formal methods provide precise and concise specifications (Wing 1990), and have promising domain-specific applications (Luqi and Goguen 1997) in systems engineering analyses, as exemplified in this paper. Formal methods can be used to provide some science behind the greens, ambers, and reds presented to senior leadership on the state of a system's evolving architecture, giving a much earlier, more accurate, more actionable indication of trouble.

Future Work

The completeness of the specifications and assertions written about architecture relationships should be validated using a model building tool such as Alloy Analyzer (Jackson 2006) to help researchers construct a more complete set of constraints for potential use as criteria for checking that the desired conditions hold, and highlight any implicit assumptions not documented in the specifications. This type of abstract reasoning about architectures fits well with the purpose of modeling languages such as Alloy, and has potential application in high level languages written for architecture development and analysis. To facilitate broader application, the expressions should also be abstracted, for example, into the vernacular of the DoDAF Meta-Model (DM2). An experiment should also be conducted to implement the schedule conflict reporting logic for the physical components in the C4ISR OTM E09 integrated architecture.

This work feeds the early stages of author's PhD research. As the author begins a literature review, comments and pointers to related work are solicited so that this work can be put into context with the efforts of the larger systems and software engineering communities.

References

Berry, D. "Formal Methods: The Very Idea, Some Thoughts About Why They Work When They Work", *Proceedings of the 1998 ARO/ONR/NSF/ARPA Monterey Workshop on Engineering Automation for Computer Based Systems*, Monterey CA, Oct. 1998, pp. 9-18.

Bowen, J. and Hinchey, M. "Seven More Myths of Formal Methods", *IEEE Software*, 12(4), July 1995, pp. 34-41.

Brooks, Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol. 20, No. 4 (April 1987) pp. 10-19.

CJCSI 3170.01G, "Joint Capabilities Integration and Development System." Retrieved September 17, 2009, from http://www.dtic.mil/cjcs_directives/

Darnton, Geoffrey, “Problem Statement Language/Problem Statement Analyzer”, http://www.pslpsa.com/index_orig.htm, Copyright Geoffrey Darnton 2001-2007

DoDAF v1.5, *DoD Architecture Framework, version 1.5*, April 23, 2007.

DoDAF v2.0, *DoD Architecture Framework, version 2.0*, December 24, 2008.

DoDI 4630.8, "Procedures for Interoperability and Supportability of Information Technology (IT) and National Security Systems (NSS).” Retrieved December 1, 2009, from http://www.dtic.mil/whs/directives/corres/html/463008.htm

IEEE 1471, “Systems and software engineering — Recommended practice for architectural description of software-intensive systems”. ISO/IEC 42010 International Standard, First Edition, July 15, 2007.

INCOSE SE Tools Database, Retrieved on December 1, 2009 from http://www.incose.org/ProductsPubs/products/toolsdatabase.aspx

Jackson, Daniel, *Software Abstractions: Logic, Language and Analysis*. The MIT Press, April 7, 2006.

Luqi and Goguen J., “Formal Methods: Promises and Problems”, *IEEE Software*, Vol.14, Jan. 1997, pp. 73-85.

Maier, M. and Rechtin, E. *The Art of System Architecting*, 2nd Edition, CRC Press, LLC, 2002

Maier, Mark, “Quantitative QFD for System Engineering”, 1995, Retrieved September 17, 2009 from http://www.casde.iitb.ac.in/springschool/AppendixG.pdf

Verma, Dinesh, Chilakapati, Rajesh and Blanchard, Benjamin S., “Quality Function Deployment (QFD): Integration of Logistics Requirements into Mainstream System Design”, *Proceedings of the SOLE Symposium*, August 1995.

Verma, Dinesh; Chilakapati, Rajesh and Fabrycki, Wolter J. “Analyzing a Quality Function Deployment (QFD) Matrix: An Expert System-Based Approach to Identify Inconsistencies and Opportunities”. *Journal of Engineering Design*, 1998.

Vitech Corporation, Schema of the CORE 5.1.5 tool, 2004

West, M. and Fowler, J. “Developing High Quality Data Models”, Version 2.0, Issue 2.1, *European Process Industries STEP Technical Liaison Executive*, 1999

Wing, J., “A Specifier’s Introduction to Formal Methods”, *IEEE Computer*, 23(9), Sept. 1990, pp. 8-24.

Biography

Kristin Giammarco is a Lecturer of Systems Engineering at the Naval Postgraduate School. She holds an M.S. in Systems Engineering from NPS and a B.E. in Electrical Engineering from Stevens Institute of Technology. At the time of this publication, she is a doctoral candidate in Software Engineering at NPS, researching the application of formal methods to architecture and systems engineering analysis.

Special thanks go to Valerie de Leon, CERDEC Space & Terrestrial Communications Directorate (S&TCD), as well as to Jason Sypniewski and Kathy Ahmad, PM C4ISR OTM, for their enthusiastic support of this research.