Model-Based Systems Engineering in Support of Complex Systems Development

J. Stephen Topper and Nathaniel C. Horner

Model-based systems engineering techniques facilitate complex system design and documentation processes. A rigorous, iterative conceptual development process based on the Unified Modeling Language (UML) or the Systems Modeling Language (SysML) and consisting of domain modeling, use case development, and behavioral and structural modeling supports design, architecting, analysis, modeling and simulation, test and evaluation, and program management activities. The resulting model is more useful than traditional documentation because it represents structure, data, and functions, along with associated documentation, in a multidimensional, navigable format. Beyond benefits to project documentation and stakeholder communication, UML- and SysML-based models also support direct analysis methods, such as functional thread extraction. The APL team is continuing to develop analysis techniques using conceptual models to reduce the risk of design and test errors, reduce costs, and improve the quality of analysis and supporting modeling and simulation activities in the development of complex systems.

INTRODUCTION

A team from APL has been using model-based systems engineering (MBSE) methods within a conceptual modeling process to support and unify activities related to system-of-systems architecture development; modeling, simulation, and analysis efforts; and system capability trade studies. These techniques have been applied to support analysis of complex systems, particularly in the net-centric operations and warfare domain, which has proven particularly challenging to the modeling, simulation, and analysis community because of its complexity, information richness, and broad scope.

In particular, the APL team has used MBSE techniques to provide structured models of complex systems incorporating input from multiple diverse stakeholders

to support understanding of critical components, interfaces, and processes of these systems, and to document explicit traceability between analytical needs and the simulations designed to meet them.

This article provides an overview of the conceptual modeling process used to support these goals, presents a simple example to illustrate the process, and explains some of the analytical techniques that can be applied directly to such models. It concludes with a discussion of the broader benefits of this approach.

OVERVIEW OF THE CONCEPTUAL MODELING PROCESS

In brief, a conceptual model is a complete, coherent representation of a system and its operating domain, including interactions with other systems and with its environment, that is common across the stakeholder community. Conceptual models are created to document and understand a problem and its context and to provide a foundation for developing and analyzing potential solutions—regardless of the method for that analysis, which may or may not include modeling and simulation (M&S).

The general goal behind conceptual modeling is to establish a framework that facilitates understanding of the problem space, synthesis of possible solutions, and analysis of those identified solutions. The process developed and used to build the conceptual model involves creating the following artifacts:

- **Domain model:** This artifact describes what the system and its environment are—it captures the high-level components of the system and its operating environment and establishes the normalized referential framework particularly important for multidisciplined stakeholder organizations.

- **Use cases:** These written descriptions of what the system will do capture its expected behaviors and its interactions with external actors.

- **Functional model:** The functional model describes how the system will accomplish its goals—it breaks the use cases into greater detail and shows activity flows and state transitions among components. Complex functionality, an increasingly common characteristic of modern systems, is difficult to address using traditional assessment techniques. In conjunction with other artifacts presented in this section, new techniques, outlined in the *Functional Thread Analysis* section, enable and enhance analysis, testing, and evaluation of complex systems, which are difficult to assess using traditional analytical methodologies and tools.

- **Structural model:** This specification of system structure allocates attributes and operations to system components, expanding and adding detail to the domain model.

The process is shown in Fig. 1 and is described in more detail below.

The conceptual modeling process provides an essential foundation as a project moves from problem space to solution space. In the early stages, the domain models and use cases provide a basis for analyzing needs and identifying problem parameters. Once the concept space is modeled, possible solutions can be developed during creative synthesis in the configuration space.

Finally, each configuration is evaluated using an applicable analysis process. It is during this evaluation step, not before, that M&S or another analysis methodology is carried out based on what has been learned and specified during the previous iteration of the conceptual development cycle. The results of this evaluation are then reflected in alterations to the conceptual model during a subsequent iteration; the cycle is repeated as many times as is necessary to develop a robust solution (e.g., a final system design or an M&S architecture). Ultimately, the process provides a coherent view of a system and its domain that can be used (and reused) as a basis for analysis of the system configuration, behavior, and alternatives. It can also form the basis for software model design in the event that an M&S effort is desired.

The process helps project stakeholders transform informal descriptions of systems into formalized, complete documentation. Each step produces output that provides traceability from the final product back to initial descriptions and assumptions. The process forces domain understanding before design, evaluation, and development.

Guiding Principles

Several overarching principles guide the conceptual modeling process.

Breadth-First Development (or, Scope Before Fidelity)

Scope is defined as the area within the boundaries of the problem space. Fidelity is defined as the level of detail incorporated into the modeling or analysis activities associated with the problem. The modeling process dictates a breadth-first approach in which the scope is covered comprehensively before fidelity is addressed. This mandate forces designers and analysts to thoroughly and objectively define the problem domain scope and prevents "jumping ahead" to selection of a particular analysis methodology, development of simulation tools, or premature conclusions.

We first capture the entire domain at a high level of abstraction and then drill down into more detail where

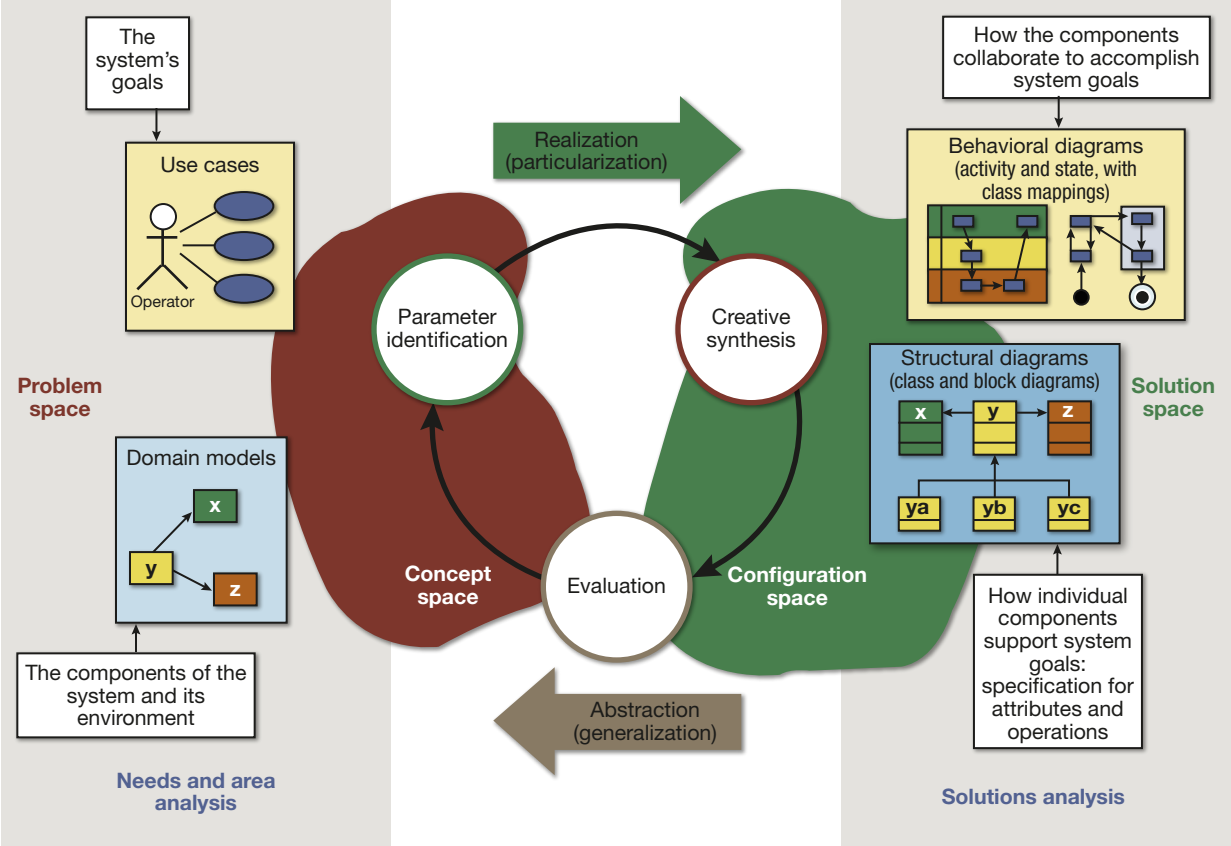

Figure 1. Conceptual modeling process and the systems development process. (Some concepts in this diagram are adapted from Ref. 1.)

such detail is necessitated by the problem. This approach fulfills several important roles. First, by scoping the problem before developing fidelity, it ensures that effort and resources are not wasted on work that does not contribute to the current task (i.e., work that falls outside the problem scope). Second, it ensures that no pertinent areas of the domain are ignored. A project that moves too quickly toward detailed analysis of the most accessible elements in the problem space risks ignoring significant elements at the periphery of the domain. Each element in the domain outside of the problem scope should be placed there explicitly after consideration, not excluded by default. Third, the approach establishes a basic understanding of the problem domain composition at the beginning of the effort that, through extension, will support not only the current analysis task but also future work.

Reusable, Repeatable, Generic Processes and Products

Historically, the output from one architecture or analysis task frequently goes unused in future efforts, which instead start over from scratch. That is, the products are so narrowly scoped to one task that they fail to have applicability to other tasks, or previous results are documented or archived in a format that is not easily accessible or lacks supporting data necessary to understand the context of their use. A development strategy based on the conceptual modeling practices described in this article places emphasis on ensuring that current work will remain relevant and useful for future projects. Processes should be well defined, documented, and repeatable, products should be reusable and extendable, and analysis results should add to the general body of knowledge used in future studies. This philosophy means that, again, projects are started at a more general level than they would be if they were aimed at a single-use solution, but these generic initial outputs provide a common starting point for future work and reduce duplication of effort. Defined, repeatable processes also reduce project scope, schedule, and resources required for future projects.

Iterative, Agile-Informed Development

While conceptual model development begins at a generic level, the goal is usually to support a specific solution—an analysis framework, simulation design, or system architecture. We arrive at this end result through iteration—progressively refining generic elements into higher-fidelity representations of the system and its domain. Prior activities are revisited as necessary, and multiple cycles through the stages are required. Iteration allows a high-fidelity solution to have a logical

basis traceable back to generic assumptions and views of the problem.

The success of the process depends on periodic review and revision of the output products by designers, analysts, customers, subject-matter experts (SMEs), operators, and other project stakeholders. Only through iteration will the final result be a verified, validated, and complete solution within its problem domain.

Each iteration should be completed within a reasonable time and should have a specific goal in order to avoid the "analysis paralysis"[2] that is anecdotally cited as a problem by opponents of breadth-first, model-based approaches. The software world has benefited from the agile movement, a development philosophy that emphasizes short-term iteration, early and frequent product delivery, collaboration with the customer, and adaptation to changing requirements.[3] Rigorously following one of the numerous agile methodologies on an architecture development effort or an analysis study may not be appropriate, but following these general principles is important to avoiding a stalled modeling process. Iteration, as discussed above, prevents moving too deep too quickly. Agile developers recognize that customer reaction to early products reveals misconceptions of customer desires before they are too late to address. Changing requirements can alter project scope, and collaboration with a broad range of stakeholders improves the quality of the end result. Iteration, with incremental product delivery and stakeholder review after each round, helps ensure that the problem domain is adequately understood, architectures are complete and developed to the correct level of fidelity, analysis tools meet users' needs, and analysis results answer the right question at the right level of detail.

Decomposition to Primitive Elements

A reductionist approach seeks to decompose elements to their simplest, primitive forms. This approach is compatible with the focus on breadth-first generality described above. Decomposition of an entity into its fundamental components allows definition of common building blocks among the domain entities. For instance, when modeling a military system domain, the behavior of sensors, jammers, and radios can be subsumed into the common function *emit*. Naturally each of these emitters serves a different purpose, but classifying them according to their most basic attributes and operations allows leveraging of their commonalities and makes standardization easier. Once the generic primitives are defined, detailed specifics extend the model as needed during subsequent iterations.

Consistency, Standardization, Traceability, and Organization

Consistency means that the elements of a product or process are not self-contradictory. Standardization means that these elements are defined according to common conventions that apply universally across the domain. Traceability indicates that the context, source, and assumptions of a particular element are readily apparent. Organization means that relationships between domain elements are defined. These four characteristics promote understanding and transparency in the products being developed, be they architectures, software models, analysis processes, or results.

Conceptual Modeling Process Elements

While conceptual model development does not prescribe a particular representation language or tool, a common approach—and the one adopted by the APL team—is to use the Unified Modeling Language (UML), a standardized representation of system design developed by the software community, or its systems engineering counterpart, the Systems Modeling Language (SysML).

UML is an industry-standard modeling language that provides, through a collection of diagrams, a standardized syntax for representing system components, their relationships, and how they operate.[4] SysML is "a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities."[5] These languages facilitate modeling of systems with standardization and rigor yet remain flexible enough to allow the user to represent systems using vocabulary and other conventions appropriate for his or her domain. This combination of rigor and flexibility, together with the broad range of model diagram types, makes these languages a good choice for representing system-of-systems configurations, operating environments, and behaviors. Furthermore, a UML or SysML model can be translated into an HTML format, allowing the user to navigate using hyperlinks and cross-references. The multidimensional richness of such a presentation allows different stakeholders to see the views, or slices, of the model of interest to them, while maintaining internal consistency across the range of viewpoints. Primarily for this reason, a body of knowledge thus codified in a standard modeling language has greater value to acquisition or research and development organizations than does a lengthy document consisting of one-off representations of systems and information.

ICONIX is a software development process that focuses on developing use cases and domain models to understand a proposed system before beginning implementation of the system.[6] While it was originally designed as a software process, the APL team has extended it to facilitate systems architecture analysis. It is appropriate for this task because it supports the guiding principles listed in the previous section. First,

it supports consistency and organization by making use of UML diagrams to document the process stages. Second, ICONIX is a "lightweight" process supporting agile development. While it supports desired rigor and documentation, it allows iteration, flexibility, and response to change—characteristics that are also important for analysis projects in the dynamic net-centric operations and warfare realm. Finally, the process supports breadth-first development by emphasizing understanding the domain as a first step, and it does not allow moving into design and modeling until the system domain has been understood.

Brief descriptions of each conceptual model artifact follow.

Model Input

To begin the modeling process, the team can start with any existing models of similar systems, input from clients and SMEs, and documentation pertaining to the target system and its mission such as concepts of operations, high-level requirements lists, and mission descriptions. Sustained stakeholder involvement is crucial.

Domain Model

The domain model provides an overview of the system being analyzed by laying out its constituent components, showing the elements in the broader environment, and specifying the relationships between these entities. Its purpose is to capture the makeup of the problem domain and show how the pieces fit together. It is represented as a high-level UML class diagram or SysML block-definition diagram without inclusion of attributes and operations. Candidate domain entities are taken from the model input, and relationships are drawn between them. The domain model should focus on breadth, encompassing all the components of the target system and all the entities that relate to it or may affect its operation in some way. Subsequent stages in the process reduce the domain model to a minimal set of elements critical to the particular system architecture or analysis project, but at this stage, the model is comprehensive (though general). Broad coverage ensures that the problem scope can be set correctly, with explicit knowledge of domain elements included or excluded from the scope. The domain model may be developed as a set of hierarchically layered diagrams. The top-level domain model may, for instance, only include a few very broad classes, each of which may be further detailed in its own diagram.

As with other products, the domain model should be repeatedly reviewed by SMEs and other stakeholders for completeness, correctness, and standardization. When the domain model is complete, it should represent the entire domain, its core components, system boundaries, and entity relationships. This model facilitates a common understanding of the domain by the project stakeholders, allows all relationships between constituent systems to be considered during analysis, and provides a uniform basis for analysis in the domain.

Use Cases

Once the domain model has been created, formalizing the constituent elements of the domain, the interactions and behaviors of those elements must be documented. A use case is "a goal-oriented set of interactions between external actors and the system under consideration."[7]

Use cases are developed by users, SMEs, and domain experts to describe generally how the system should function and are listed in a use case diagram, which depicts the descriptive name of each use case along with other participating actors. Like any UML/SysML artifact, use cases can be developed in a hierarchy specifying different levels of detail. Once all use cases have been identified, they are formally documented in use case description documents. At this stage, some initial scoping of the domain can begin—only use cases of interest to the problem need to be documented in detail. A use case description typically includes a name and summary description, the participating actors, preconditions for execution, triggers causing execution, activities occurring within the use case, possible exceptions and alternative execution paths, and postconditions defining use case completion. Numerous sources, such as the book by Cockburn,[8] describe how to build use cases.

Writing use case documentation is an important initial step where general system operation statements are formalized into explicit descriptions of system functionality. Use cases form the basis for determining participation of system elements in activities and identifying common recurring activities throughout the domain. Well-documented use cases resolve ambiguity about scope by precisely specifying the system's functions. Because use cases collectively become the definition of project scope, continuous stakeholder vetting and agreement is required.

Note that the domain model and the use cases are particularly dependent on each other. It is likely that use case development will uncover additional domain entities absent from the domain model; these should be added as they are discovered.

Functional Model

Activity and state diagrams show the behavior of system objects as they collaborate to fulfill the goals identified in the use cases.

Activity diagrams trace functional and informational flow through the different components of a system. They are often generated by translating the operational steps documented in a use case into graphical form and further decomposing these steps. Candidate system configurations are then created by allocating these activities to domain model entities.

State diagrams are another type of functional model that focus on a system component's states. They depict boundaries between states, activities occurring in each state, and conditions governing the state transitions.

Both activity diagrams and state diagrams are useful for documenting the behavior of a system. Activity diagrams focus on a single thread, which may link many system objects, while state diagrams focus on how a single system object participates in many different system activities.

When some rigor is employed in their construction, both types of functional diagrams adhere to graph theory formalisms. As a result, the underlying data structure of these diagrams can be used to enable analysis of complex system functionality using graph theoretic techniques and computational methods.

Structural Model

Structural models expand the domain model to represent system configurations. The domain model classes are combined with behavioral models via functional allocation and assignment of attributes and constraints to reveal how system components behave.

Developing the structural model fleshes out the depth of the domain model and scopes it to the analysis problem. Classes that do not bear on the problem are removed, and candidate configurations are identified. Remaining classes are developed further through the addition of attributes, operations, inheritance hierarchies, and more specific relationships. Classes need not be developed to uniform depth throughout the model; in some cases, a high degree of detail is required, while in others, an extremely general view of a particular component may be sufficient.

SysML block definition diagrams and internal block diagrams can be used in place of UML class diagrams and composite structure diagrams, respectively, to model physical systems and their interactions.

SUPPORTING SYSTEM DEVELOPMENT AND ANALYSIS WITH A CONCEPTUAL MODEL

A conceptual model is primarily viewed as a means of improving documentation, communication, and design consistency within a project. However, development of a rigorous model supports development and analysis processes across the spectrum of systems engineering activities. This section explores these benefits. It begins with a description of functional thread analysis, which is an analytical technique that can be applied to the relational data underlying a conceptual model, continues with a simple example from the net-centric domain, and concludes by cataloging the benefits this and other MBSE approaches have for various aspects of the systems engineering process.

Underlying Database

Many modeling tools used for conceptual modeling store model data in an underlying database. While the graphical user interface is appropriate for constructing and viewing the model, the model data can be extracted using a relational database management system and processed using other analytical tools. This process also enables linkage between model data and external analysis data, documentation, and other program artifacts. Many tools also support configuration control via common applications. Model data is exportable to XML-based formats, allowing sharing between modeling tools. These capabilities enable the conceptual model to form the foundation of a program's information management system.

Functional Thread Analysis

The state of a complex system changes continuously as the designed functionality is executed within changing mission phases and environmental conditions. These systems can invoke a large number of functional threads to accomplish (or fail) a required task, and as system complexity grows, it can be difficult to identify critical threads and accurately assess key system performance requirements.

Graph-Theoretic Algorithmic Techniques for Functional Thread Extraction

Analysis of functional threads represented by the activity model uses the underlying model data discussed above. Activity diagrams can be interpreted as directed graphs, where activities are the nodes and the activity transitions are the edges between them.

The APL team has successfully extracted these node and edge data from conceptual model databases. Processes based on graph-theoretic path-finding algorithms are then executed on these data to provide a list of all unique paths through the activity model. The primary challenges are to properly handle parallelism and recursive looping in the activity paths; the current implementation of the prototype tool for thread extraction uses a modified breadth-first search to discover activity threads (see the example at the end of this article).

The number of extracted threads from even a simple model shows the complexity inherent in many systems. The extracted thread data can also be analyzed to determine critical activities and, via associated activity-to-block allocations, a system's critical components. Bottlenecks and heavily used interfaces are identifiable.

Thread Extraction Example

A methodology that enables machine processing of functional designs is especially useful in complex, collaborative systems such as those involved in network-

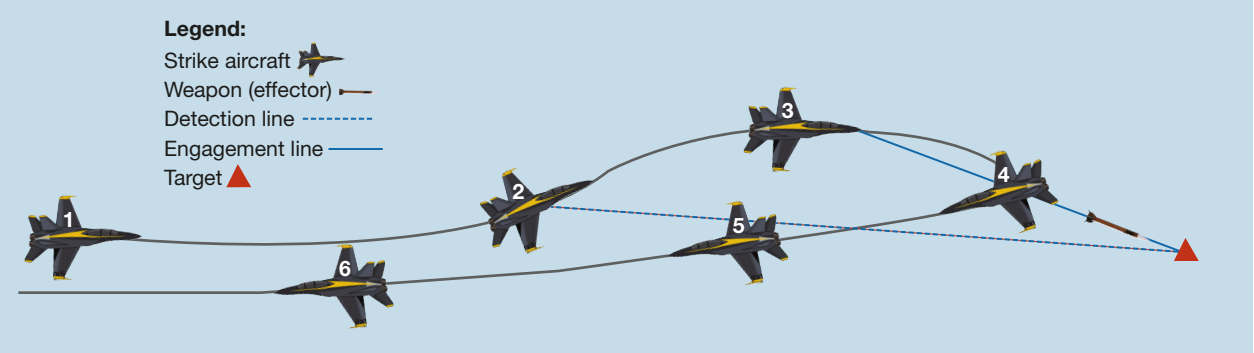

Figure 2. A scenario representing a simple strike mission.

centric warfare. In these cases, the design focus must consider both the individual platforms and the capabilities and impacts of the network on these platforms. The continuous adaptation of network-centric systems to the operating environment[9] creates functional processes that are much more complex than those of traditional systems.

Figure 2 depicts a simple strike mission where an aircraft follows a timed route to apply an effect on a target. The numbers in the aircraft represent the time step of the mission. The simplicity of this example belies increasing levels of complexity as the system is decomposed into fundamental elements that define its functions and structures. The aircraft ingresses to the target area at step 1, detects the target at step 2, engages the target at step 3, and egresses the target area via steps 4, 5, and 6.

Figure 3 depicts the functional model, which represents the mission-level functions of the aircraft. Given alternative pathways through the process and the possibility for feedback loop processes, this simple model contains more than 100 different functional paths. There are three major pathways through the model/graph,and these three pathways represent different control architectures:[10] (*i*) deliberate control, which exclusively uses the *Dynamic replanning* activity path and evaluates and then selects a specific course of action (COA)/plan to follow; (*ii*) reactive control, which exclusively uses the *Select preplanned COA* path, which uses existing preplanned responses to external events; and (*iii*) hybrid control, which blends the reactive and deliberate control processes.

For the example in Fig. 2, no deviations from the planned situation are detected, and the system functionality is represented by the following path through Fig. 3:

Collect data → *Locate and classify* → *Apply effects* → *Assess effects* → *End mission*

As the mission environment becomes more complex, alternative pathways are invoked. Figure 4 introduces a threat system to the environment with a detection

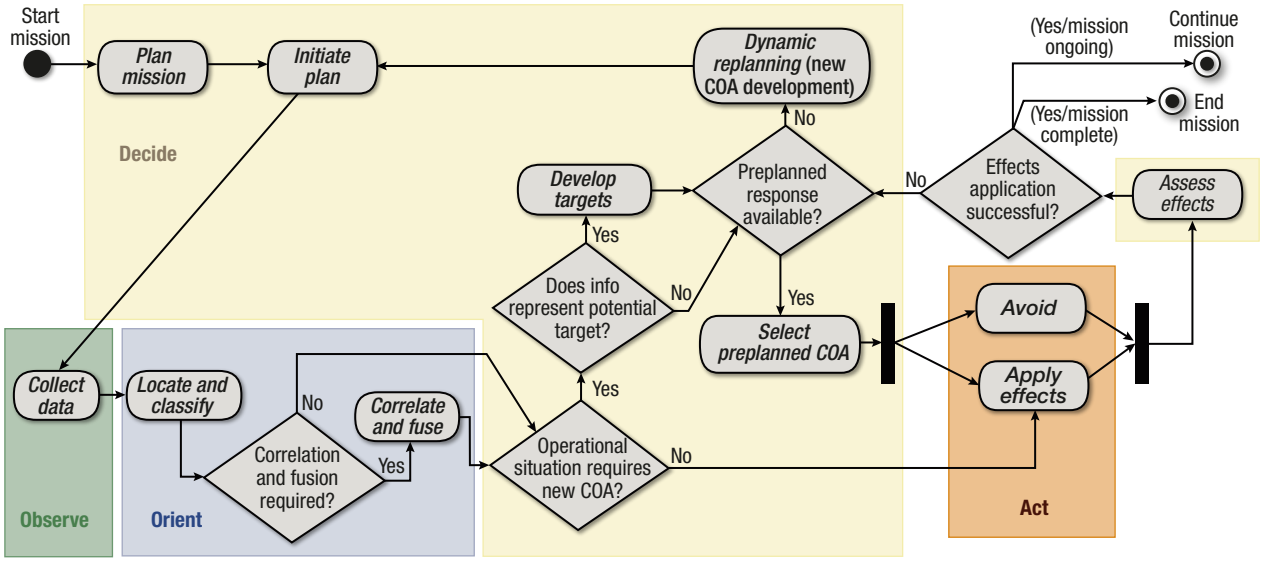

Figure 3. Generic mission-level activities framed in the context of the observe–orient–decide–act loop.[11]

and engagement envelope. In this case, system functions follow the reactive control path (Fig. 3), where the strike aircraft detects and locates the threat system (*Collect data* and *Locate and classify*), selects a preplanned COA, engages the threat (*Apply effects*), assesses effects of the engagement result, and resumes the mission to the primary target—collecting target information, locating and classifying the target, applying and assessing effects—and finally ending the mission.

When the mission environment includes additional strike aircraft and the ability of the aircraft to exchange information using communications systems and estab-

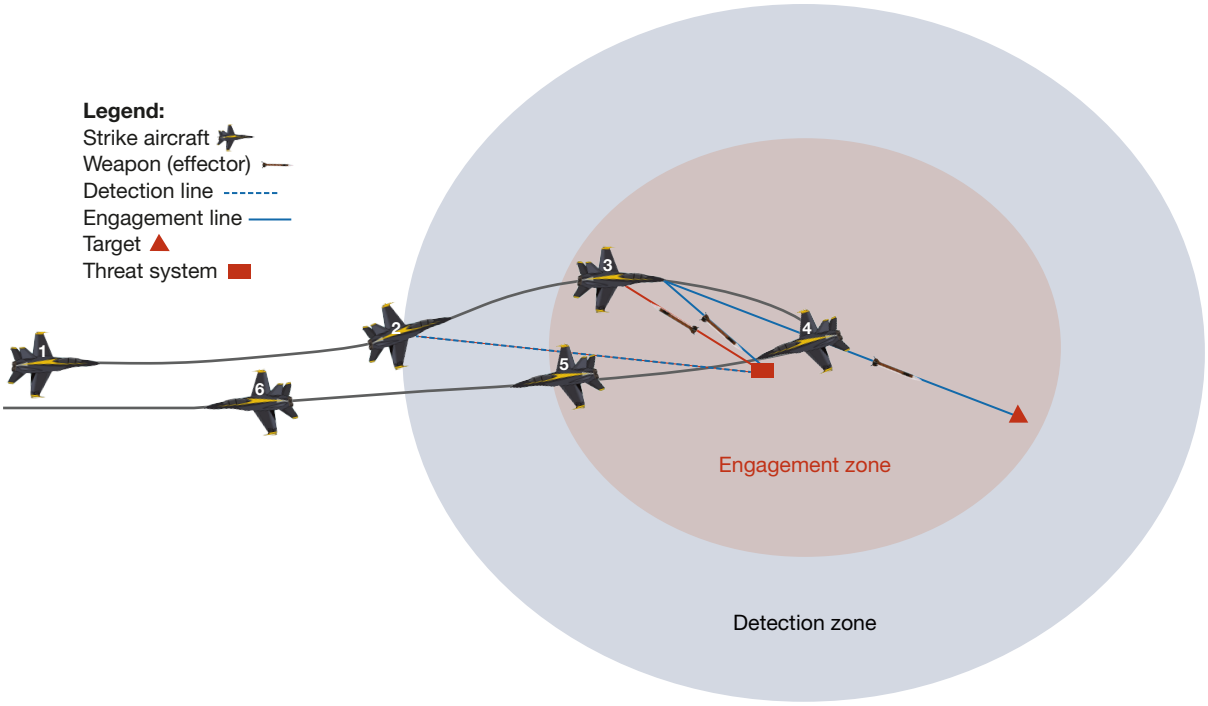

Figure 4. A scenario representing a strike mission with a threat response element.

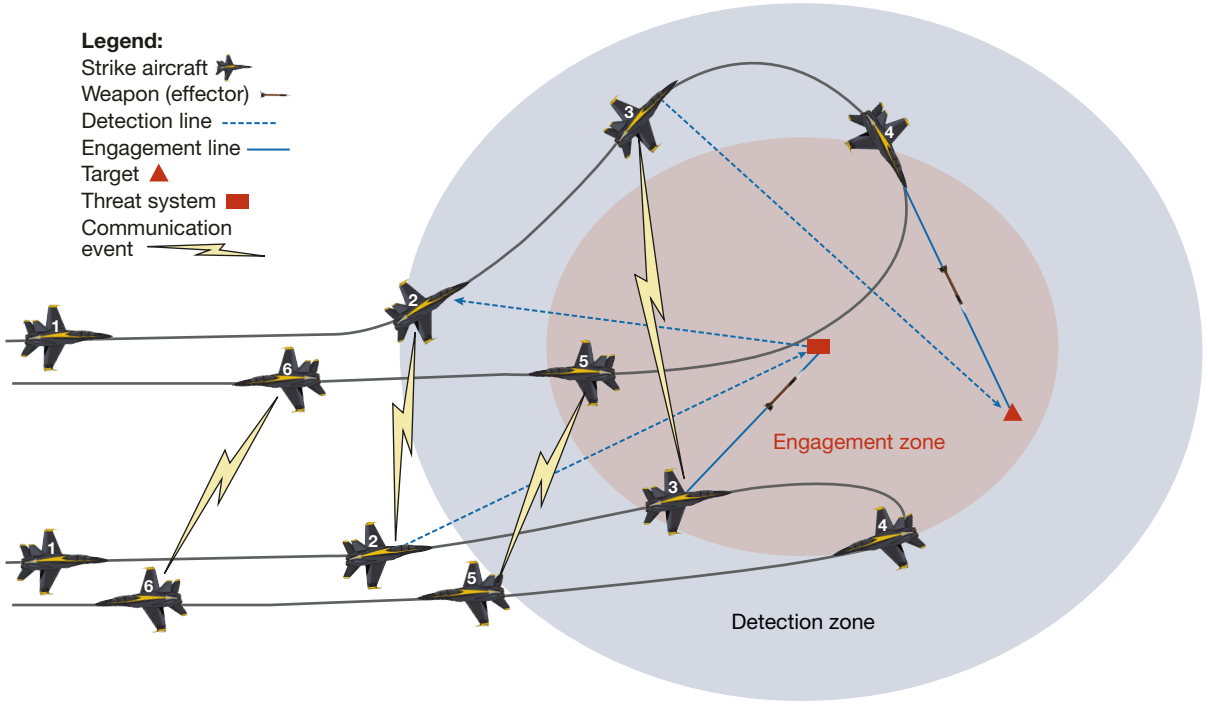

Figure 5. A scenario representing a strike mission in a net-centric environment with a threat-response component.

lished networks, specific functionality can be invoked on or triggered from either platform. This network-centric operational mode, in which platforms collaborate to achieve certain functions, enables emergent functionality that is greater than the sum of its parts. This added flexibility and capability is accomplished at a cost of increased complexity, and evaluation of this functionality becomes difficult to accomplish using traditional tools and techniques. Figure 5 illustrates a simple example of a network-centric operation where two similar platforms leverage mutual situational awareness enabled by the sharing of information.

This information sharing introduces communication systems and networks into the example problem. Each mission-level activity on an individual platform in Fig. 3 now includes the possibility of interaction with other platforms' functionality via communications messages, greatly increasing the complexity of the mission thread.

When the threat information is received by the first strike aircraft in the scenario in Fig. 5, the *Dynamic replanning* activity in Fig. 3 is triggered because a new COA is required, the information represents a potential target, and there is not a predefined COA available. This activity invokes defensive system functionality depicted in Fig. 6. In this scenario, aircraft execute a defensive response selected via the *Select COA* activity in Fig. 6 (this activity can use, given the system configuration, the control architectures presented in Fig. 3)—the first aircraft dynamically reroutes (*Avoid threat detection*

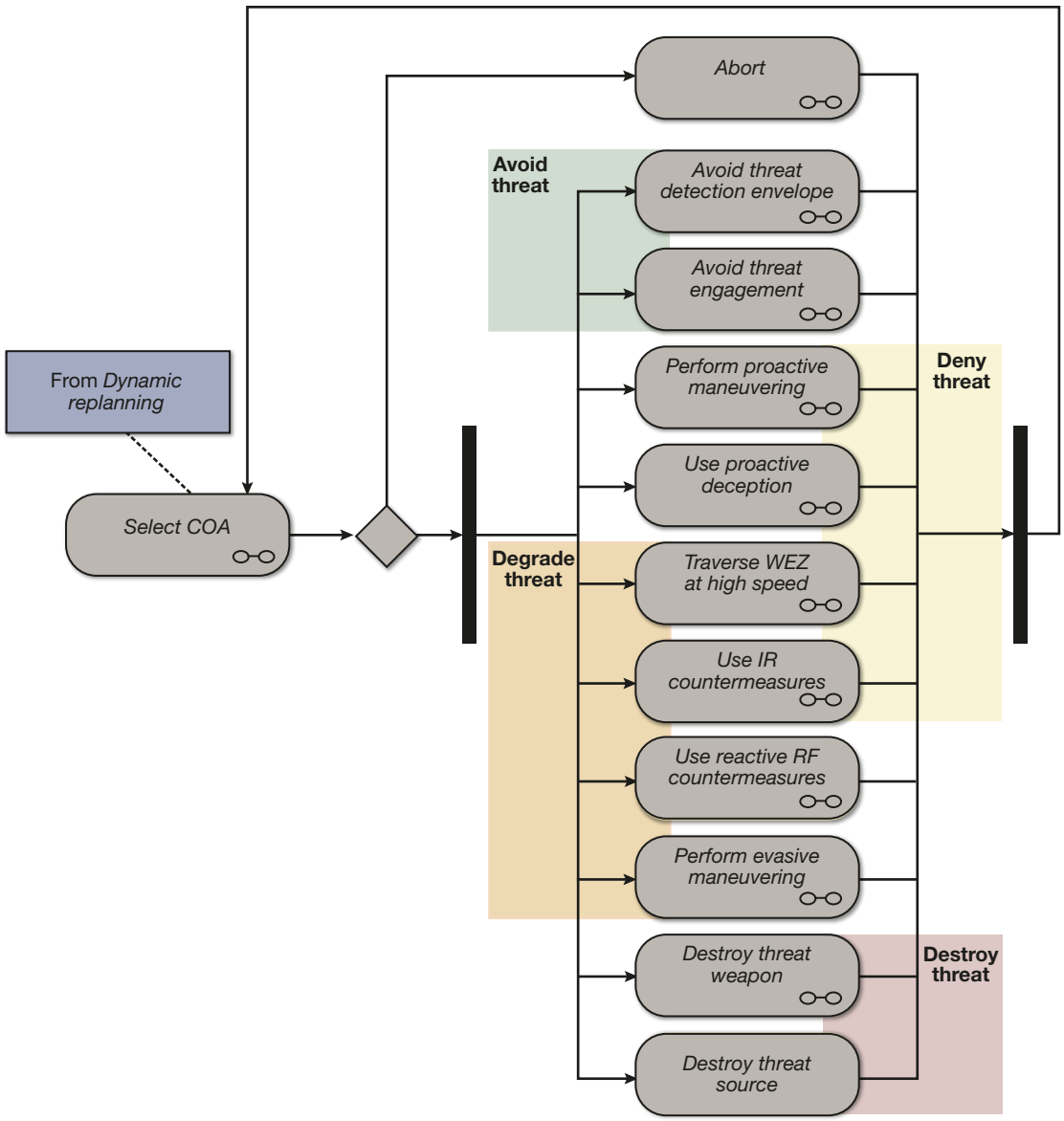

Figure 6. High-level defensive activities. WEZ, weapon engagement zone.

envelope) while the other strike aircraft applies an effect to neutralize the threat (*Destroy threat source*). Note that activity models are compartmentalized to manage the complexity of the model. In Fig. 6, each high-level activity depicted corresponds to a defensive system use case, and multiple layers of additional detail can be subsumed by each of these activities.

Model complexity increases because of the increasing intricacy of the modeled process, as in the evolution from the scenario in Fig. 2, to the scenario in Fig. 4, to the scenario in Fig. 5, as well as via decomposition of system functionality. Understanding the set of the possible functional threads and how frequently they are invoked is an important consideration for identifying critical mission threads and key system parameters.

Conceptual modeling enables discovery of mission threads, information and physical elements, and measures of effectiveness, which are retained in MBSE artifacts and their underlying database to support a wide range of systems engineering activities.

Conceptual Modeling and Systems Engineering Activities

Pace[12] has discussed the value of developing simulation conceptual models using UML concepts or similar

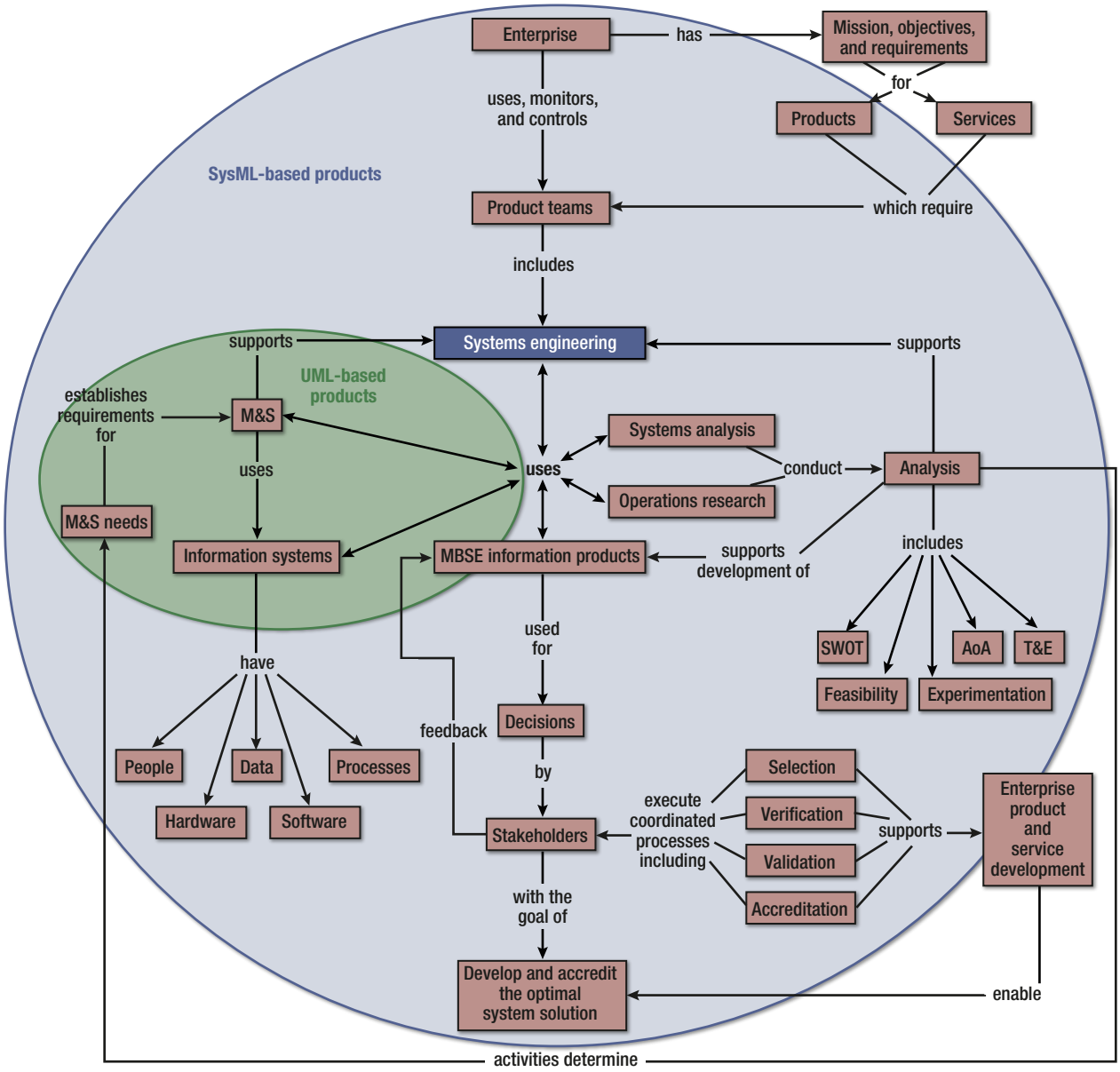

Figure 7. Systems engineering domain elements. AoA, analysis of alternatives; SWOT, strengths, weaknesses/limitations, opportunities, and threats; T&E, test and evaluation.

processes and recognized that decomposition of a simulation into its fundamental elements capturing completeness, consistency, coherence, and correctness of simulation requirements was an essential element of any engineering effort. Since then, the UML specification has been extended to systems engineering efforts using SysML, and the field has expanded the scope of model-based activities from simulation software development to nearly every aspect of a systems development effort (Fig. 7).

Informatically rigorous conventions for semantics and notation help create MBSE UML/SysML artifacts that improve quality and reduce preparation time of systems engineering products. This section discusses how the MBSE approach supports key engineering activities. These activities include conceptual design, previously covered in the *Overview of the Conceptual Modeling Process* section, along with these other important functions:

- **Analysis and experimentation:** Evaluation of candidate system configurations through analysis and experimentation occurs during iteration of the system design process shown in Fig. 1. As system complexity increases, the number of functional threads increases exponentially, increasing the probability of system errors, faults, and failures.[13] Model-based approaches support automation of analysis processes and focus human-centered analysis and experimentation efforts to help reduce errors and efficiently use resources.

- **Architecture:** Design processes describe and structure information about the system and its interaction with the environment to create a system architecture.[14] As system complexity increases, maintaining a document-based architecture becomes difficult, increasing the risk of overlooking critical information and important interfaces. A model-based approach allows for representations of the system that are easier to use, reuse of system component models throughout the architecture, and configuration control. Such an approach also makes it easier to create architectures incrementally, starting with simple parts and developing more complex systems using the existing artifacts.

- **Acquisition process support:** Similarly, document-based approaches supporting management, systems engineering, and analysis activities become disjointed as complexity increases. Documents, data, and processes can be linked directly to the system conceptual model, which creates a structure for program information management and configuration control. Model-based approaches also enhance cost estimation and governance processes, which are critical for successful management of acquisition programs.

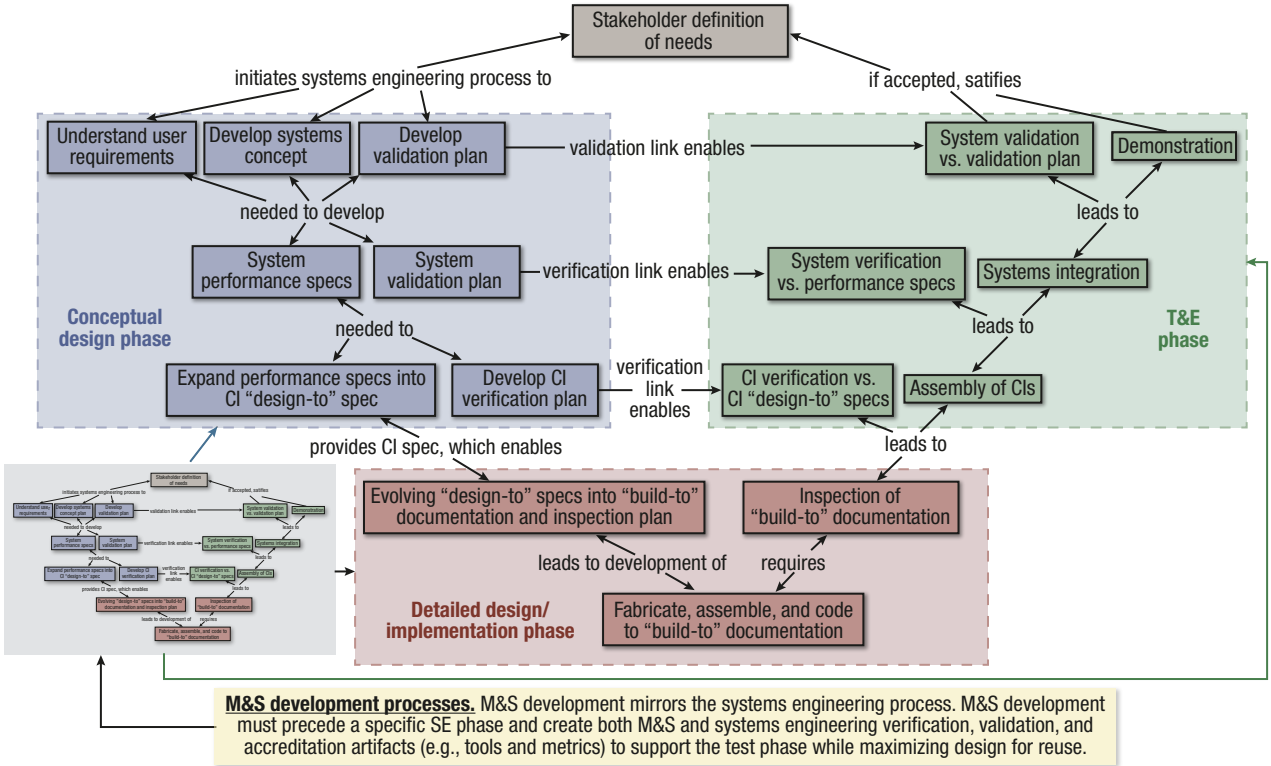

Figure 8. Systems engineering process with a supporting M&S process. CI, configuration item.

- **Modeling and simulation:** M&S is frequently used to support analysis and experimentation activities, as shown in Fig. 8. Often, complex simulations are reused on multiple programs without adequate assessment of whether their capabilities meet analysis needs. When M&S requirements are developed using MBSE processes, the resulting model enables assessment of a simulation's risks, capabilities, and appropriateness for analysis, test, and evaluation activities.

- **Test, Evaluation, and Validation, Verification, and Accreditation (VV&A):** As shown in Fig. 8, validation, verification, and accreditation depends on linkage between the conceptual design and the test and evaluation (T&E) phases of the systems engineering process model. The model developed in the conceptual design phase contains the system structure, functionality, interfaces, and metrics used for verification and validation during the system T&E phase. Verification of simulation-based T&E activities is supported through the direct linkage of the simulation and system conceptual models, showing where and how each system entity is being represented in simulation.

The following example extends the functional thread analysis presented earlier to show how the data underlying the model support T&E activities.

Matarić[10] characterized three different control architectures, which are implicit in Fig. 3: *deliberate*, which requires COA development; *reactive*, which initiates preplanned activities; and *hybrid*, which uses an allocation process to decide on the preferred path. In this example, flight test planners are interested in evaluating the reactive control architecture. Because *Select preplanned* COA represents the primary activity for reactive system responses, the set of possible functional threads can be reduced to only those paths incorporating this node. In this case, there are 29 such paths associated with a reactive process. The data underlying the UML model for one of these 29 paths are depicted in Table 1. These data were used by thread extraction tool to generate the graphical representation of this path shown in Fig. 9.

Analysis of functional processes can further reduce the number of possible threads. By using the system model to focus on the relevant COAs (in this case, selecting the appropriate defensive activities from Fig. 6), the planner can identify key system test points either qualitatively or quantitatively (for instance, using design of experiments processes). By combining the key test points with selection of specific mission profiles (Figs. 2, 4, and 5) and operational environment conditions (e.g., the scenario), the planner can develop a comprehensive test plan.

MBSE techniques support activities across the entire life cycle of a given system. The processes simultaneously create structured information system-based models that link management, engineering, analysis, and M&S artifacts and activities via a common architecture.

CONCLUSION: MBSE EFFECTIVENESS

With the increase in system complexity precipitated by the advent of network-centric systems, MBSE techniques offer a way to capture, archive, and use information that is essential for complex system design, analysis, implementation, and T&E throughout a system's life cycle. The conceptual model includes entities, their important attributes and interrelationships, how they operate and behave, and any assumptions being made about them. It provides a basis for future analysis studies, model development, simulation efforts, system requirements definition, and program information management. As a means of exploring and documenting requirements and assumptions, the process is the groundwork for system analysis, design, and development.

A robust conceptual model does the following:

- Facilitates communication and collaboration among project stakeholders by standardizing and documenting a common reference blueprint for the project. This basis allows the team to exhaustively explore the system's conceptual and configuration spaces and identify and assess key parameters in the evaluation of system alternatives.

- Promotes reuse of components and analytical results among projects across a shared domain.

- Enables information management and integrates business and engineering processes into a single model. A conceptual model of the project, especially one that reuses components from previous projects and includes elements from the enterprise architecture as well as the system, allows managers to better estimate the scope, schedule, and resources needed to develop and deploy a complex system.

- Documents traceability from needs to results, supporting verification and validation. With respect to M&S in particular, the conceptual model "addresses the simulation's context, how it will satisfy its requirements, and how its entities and processes will be represented."[12] That is, once the a basic conceptual model is developed for the system being analyzed, that model can be extended to document how that system will be represented in software, explicitly depicting how each component will be modeled and to what level of fidelity.

These benefits are similar to those seen in the software industry after the adoption of object-oriented analysis and design methods supported by UML modeling.

The justification for a conceptual modeling process for complex analysis, simulation, and system development activities is straightforward, and its ramifications

Table 1. Conceptual model thread data

StartObjectName	StartObjectType	EndObjectName	EndObjectType
Start mission (394)	StateNode	*Plan mission* (362)	Activity
Plan mission (362)	Activity	*Initiate plan* (356)	Activity
Initiate plan (356)	Activity	*Collect data* (327)	Activity
Collect data (327)	Activity	*Locate and classify* (357)	Activity
Locate and classify (357)	Activity	Correlation and fusion required? (381)	Decision
Correlation and fusion required? (381)	Decision	*Correlate and fuse* (337)	Activity
Correlate and fuse (337)	Activity	Operational situation requires new COA? (390)	Decision
Operational situation requires new COA? (390)	Decision	Does info represent potential target? (382)	Decision
Does info represent potential target? (382)	Decision	*Develop targets* (343)	Activity
Develop targets (343)	Activity	Preplanned response available? (391)	Decision
Preplanned response available? (391)	Decision	*Select preplanned* COA (365)	Activity
Select preplanned COA (365)	Activity	Fork from *Select preplanned* COA (412)	Synchronization
Fork from *Select preplanned* COA (412)	Synchronization	*Avoid* (323)	Activity
Avoid (323)	Activity	Join to *Assess effects* (411)	Synchronization
Join to *Assess effects* (411)	Synchronization	*Assess effects* (316)	Activity
Assess effects (316)	Activity	Effects application successful? (383)	Decision
Effects application successful? (383)	Decision	Preplanned response available? (391)	Decision
Preplanned response available? (391)	Decision	*Select preplanned* COA (365)	Activity
Select preplanned COA (365)	Activity	Fork from *Select preplanned* COA (412)	Synchronization
Fork from *Select preplanned* COA (412)	Synchronization	*Apply effects* (311)	Activity
Apply effects (311)	Activity	Join to *Assess effects* (411)	Synchronization
Join to *Assess effects* (411)	Synchronization	*Assess effects* (316)	Activity
Assess effects (316)	Activity	Effects application successful? (383)	Decision
Effects application successful? (383)	Decision	End mission (380)	StateNode

Color coding associates each activity with a phase in the observe (green)–orient (blue)–decide (yellow)–act (orange) process. Numbers in parentheses in the table and in Fig. 9 are internal node indices in the thread data.

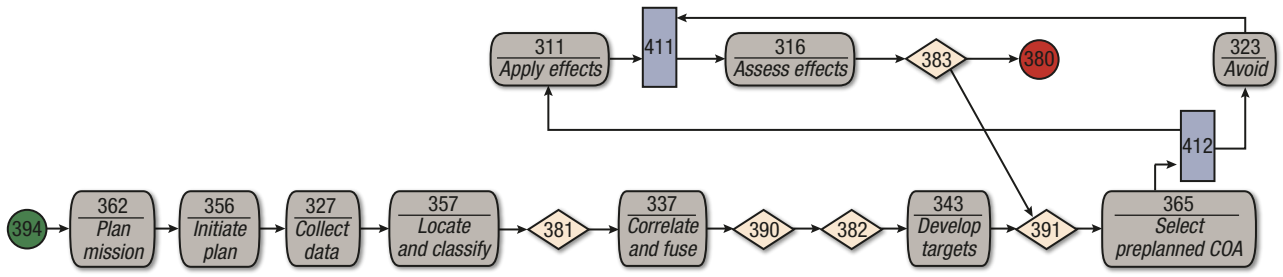

Figure 9. A mission functional thread.

can be far-reaching. Conceptual modeling has the potential to reduce acquisition costs while enhancing analysis, design, and T&E processes. Although introducing conceptual modeling in a new organizational environment can involve overcoming cultural and educational barriers, the APL team has received positive feedback from sponsors in cases when a conceptual modeling approach was adopted.

REFERENCES

[1]Kroll, E., Condoor, S., and Jansson, D., *Innovative Conceptual Design: Theory and Application of Parameter Analysis*, Cambridge University Press, Cambridge, UK (2001).

[2]Wiegers, K. E., "Karl Wiegers Describes 10 Requirements Traps to Avoid," *Software Testing & Quality Engineering* **2**(1), 35–40 (2000).

[3]Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., et al., "Manifesto for Agile Software Development," http://agilemanifesto.org/ (2001).

[4]Fowler, M., and Scott, K., *UML Distilled: Applying the Standard Object Modeling Language*, Addison Wesley Longman, Reading, MA (1997).

[5]Object Management Group, *OMG Systems Modeling Language*, http://www.omgsysml.org/ (accessed 29 Apr 2013).

[6]Rosenberg, D., and Stephens, M., *Use Case Driven Object Modeling with UML: Theory and Practice*, Apress, Berkeley, CA (2007).

[7]Malan, R., and Bredemeyer, D., "Functional Requirements and Use Cases," www.bredemeyer.com/pdf_files/functreq.pdf (2001).

[8]Cockburn, A., *Writing Effective Use Cases*, Addison Wesley, Boston, MA (2001).

[9]Cebrowski, A. K., and Garstka, J. H., "Network-Centric Warfare: Its Origin and Future," *Proceedings* **124**(1), 1139 (1998).

[10]Matarić, M., *The Robotics Primer*, MIT Press, Cambridge, MA (2007).

[11]Boyd, J. R., "The Essence of Winning and Losing," http://www.danford.net/boyd/essence.htm (1996).

[12]Pace, D. K., "Ideas About Conceptual Model Development," *Johns Hopkins APL Tech. Dig.* **21**(3), 327–336 (2000).

[13]McCabe, T., "A Complexity Measure," *IEEE Trans. Softw. Eng.* **SE-2**(4), 308–320 (1976).

[14]Rechtin, E., *Systems Architecting: Creating & Building Complex Systems*, Prentice-Hall, Inc., Upper Saddle River, NJ (1991).

The Authors

J. Stephen Topper, a member of the Senior Professional Staff in the Force Projection Department, has been Project Manager for United States Air Force conceptual modeling efforts in support of simulation validation, verification, and accreditation. He has implemented conceptual modeling processes on several external and internal APL projects. **Nathaniel C. Horner**, a former Associate Professional Staff member who was also in the Force Projection Department, served as Project Manager and Technical Lead on conceptual modeling projects for various Air Force sponsors. He developed conceptual models in support of systems development and has also built analytical software tools and simulations. He is currently pursuing a Ph.D. in the Engineering and Public Policy Department at Carnegie Mellon University. Both Mr. Horner and Mr. Topper teach conceptual modeling-related classes in The Johns Hopkins University Whiting School of Engineering's Systems Engineering Department. For further information on the work reported here, contact Mr. Topper. His e-mail address is steve.topper@jhuapl.edu.

The *Johns Hopkins APL Technical Digest* can be accessed electronically at **www.jhuapl.edu/techdigest**.