Using SysML Modeling To Understand and Evolve Systems of Systems

Jo Ann Lane1, * and Tim Bohn2

1Center for Systems and Software Engineering, University of Southern California, 941 West 37th Place, Los Angeles, CA 90089-7081
2IBM Strategy and Technology Office, IBM, 30501 Agoura Rd., Suite 100, Agoura Hills, CA 91301

Received 13 January 2012; Revised 6 April 2012; Accepted 6 April 2012, after one or more revisions
Published online in Wiley Online Library (wileyonlinelibrary.com).
DOI 10.1002/sys.21221

ABSTRACT

The recent Department of Defense (DoD) guidebook, Systems Engineering Guide for Systems of Systems, describes how traditional systems engineering activities have evolved to support systems engineering at the system of systems (SoS) level. Part of the research for this guidebook probed the application of modeling and simulation to support SoS systems engineering. The findings indicated that limited modeling and simulation are currently used, but that additional support would be useful if models could be quickly generated and used to support needed decision making. This paper presents an approach to using Systems Modeling Language (SysML) models to support these needs. © 2012 Wiley Periodicals, Inc.

Key words: system of systems; system of systems engineering; modeling; SysML

1. INTRODUCTION

System of systems (SoS) is a term often used to describe a set of related software-intensive systems that are used to provide capabilities that cannot be accomplished by any single system in the set. Definitions of SoS abound, as shown in Lane and Valerdi [2007], but the one most used is Mark Maier’s definition:

A system-of-systems is an assemblage of components which individually may be regarded as systems and which possesses two additional properties: operational independence of the components … and managerial independence of the components. ...[Maier, 1998: 271]

Each system that is part of an SoS is often referred to as a constituent system. In recent times, SoSs have become more complex and difficult to manage, due in part to the fact that SoS capabilities can conflict with constituent system requirements and plans. Furthermore, for systems that belong to more than one SoS, conflicts can occur between the various SoSs’ demands for changes to a given constituent system. To better guide SoS evolution and performance enhancement, many organizations have established an SoS engineering (SoSE) team.

To better understand the differences between systems engineering for a single system and SoSE and to support Department of Defense (DoD) SoS development and evolution, an initiative was established to analyze 18 SoSs and develop guidance to support SoSE. The results, published in the recent DoD guidebook, Systems Engineering Guide for Systems of Systems [DoD, 2008] (hereinafter referred to as the SoSE guidebook), characterized SoSE using seven core elements: translating capability objectives into requirements; understanding SoS constituent systems and their relationships; assessing actual performance to capability objectives; developing, evolving, and maintaining an SoS architecture; moni-
toring and assessing constituent system changes; addressing new requirements and options; and orchestrating upgrades to SoS. As part of Maier’s SoS work and the SoSE research to develop the SoSE guidebook, it became clear that there was a taxonomy of SoSs that was based on levels of authority and responsibility for a given SoS:

- Virtual SoS [Maier, 1998]: ad hoc evolution as characterized by today’s Internet.
- Collaborative SoS [Maier, 1998]: no formal management at the SoS level, but engineering teams of the constituent systems collaborate to develop solutions for new capability needs.
- Acknowledged SoS [DoD, 2008]: SoS evolution is guided by an SoSE team whose key engineering activities are described in the SoSE guidebook.
- Directed SoS [Maier, 1998]: SoS systems engineering efforts are managed through formal mechanisms as illustrated by the lead systems integrator contractual relationships for the Future Combat Systems program and described in Lane and Boehm [2008].

It has also been noted in the SoSE guidebook [DoD, 2008] that seldom does an organization decide to develop a “new SoS”. Rather, the more typical situation (especially in the US DoD) is that single systems are developed/acquired and these systems are designed to interface to other systems that either exist or are under development. Over time, these interfacing systems evolve with new systems being integrated and other systems replaced. At some point in time for some of these interoperating systems, it becomes clear that the cross-cutting capabilities of the “collective” are so important that an organization decides to guide the evolution through an SoSE team. It is at this point that the SoS transitions from a collaborative SoS to an acknowledged SoS.

Finally, Table I summarizes various types of SoS development and evolution from an SoS maturity perspective and highlights the types of SoSE activities that are the subject of this paper: midlife activities within an acknowledged SoS that are focused on the development of new SoS capabilities or enhancement of existing SoS capabilities or current performance. (Note that the types of SoS development in Table I that are not addressed in this paper are much more complex and typically benefit from much more extensive modeling as is illustrated in Friedenthal, Meilich, and Izumi [2007].)

Through conversations with the SoSE teams that participated in the DoD SoS research, it is clear that traditional SE processes must be tailored at the SoS level to both guide the evolution of the SoS and at the same time allow the constituent systems to evolve to meet the needs of their stakeholders. To support this evolutionary process, several SoSE teams have indicated that modeling tools and methods are needed to understand constituent systems and their relationships, assess constituent system changes, and address new requirements and solution options [DoD, 2008]. The following sections describe the SoSE core elements and the desired modeling support, outline some innovative ways of using the Systems Modeling Language (SysML) tools to characterize SoS architectures and capabilities, and provide examples illustrating these techniques.

2. BACKGROUND

2.1. SoSE Core Elements

To understand what needs to be modeled and to what level it needs to be modeled, it is important to understand the engineering processes and related activities that can be supported by each type of model. The SoSE core elements that describe the engineering activities identified in the DoD SoS study [DoD, 2008] are illustrated in Figure 1. The following summarizes each of the SoSE core elements in more detail.

2.1.1. Translating Capability Objectives. Translating Capability Objectives starts with an SoS need or desired new capability and works to understand that new capability as well as various alternatives for providing that capability. Options can span the full spectrum of Doctrine, Organization, Training, Materiel, Leadership and Education, Personnel and Facilities (DOTMLPF), but typically require changes to one or more SoS constituent systems. When changes are required to one or more SoS constituent systems, the SoSE team analyzes the desired capability and translates that into a set of requirements that can later be prioritized and further refined as part of a future SoS upgrade cycle.

2.1.2. Understanding Systems and Their Relationships. Understanding Systems and Their Relationships is the core element that maintains information about the current state of the SoS and its constituent systems. The focus is on the constituent system capabilities and how they contribute to SoS capabilities, their interfaces, and the data that are shared through the constituent system interfaces.

2.1.3. Assessing Performance to Capability Objectives. Key to enhancing and evolving an SoS is understanding its current performance and how this performance meets current and future needs. Performance assessments include the ability to meet current and planned capability objectives and identification of capability gaps, as well as evaluations of quality characteristics such as response time, reliability, security, safety, and the ability of the SoS architecture to support future needs. Because SoS capabilities are performed by a set of systems, each with its own upgrade cycle, SoS assessments are more often done through analysis of demonstrations, tests, and data collection in the operational environment.
2.1.4. Developing and Evolving SoS Architecture. Typically, when a set of systems is acknowledged as an SoS, there is already an architecture of sorts in place. The goal of the SoSE team is to understand the existing architecture, evaluate architecture alternatives to mitigate limitations and improve performance, and then identify needed changes to migrate the SoS to the desired architecture.

2.1.5. Monitoring and Assessing Changes. Monitoring and assessing changes being made within the constituent systems are necessary to ensure that the constituent systems are not inadvertently making changes to their systems that will adversely affect SoS capabilities and performance.

2.1.6. Addressing Requirements and Solution Options. As new SoS requirements are identified, they are added to the SoS backlog for implementation. The requirements in the SoS backlog are evaluated and prioritized. Part of the evaluation and prioritization process identifies solution options and opportunities within the constituent system upgrade cycles for implementing the requirements in a given upgrade cycle.

2.1.7. Orchestrating Upgrades. Orchestrating Upgrades is primarily an SoSE oversight activity, monitoring the progress of the constituent system upgrades that are implementing needed SoS changes and helping to mitigate any outside obstacles to the timely success of the upgrades. The other key aspect of Orchestrating Upgrades is monitoring the constituent system test and evaluation activities, the rollout of the constituent system upgrades, and conducting SoS development testing before deployment and operational testing of the SoS-related changes as the various constituent system pieces are deployed.

2.2. Desired SoSE Core Element Modeling Support

As mentioned earlier, during interviews of SoSE teams in support of the SoSE guidebook [DoD, 2008], it became clear that modeling and simulation (M&S) use was limited. Those SoSE teams that employed M&S used it to understand complex and emergent behaviors of systems that interact with each other, explore new capability options, identify potential integration issues, evaluate current SoS architecture and potential architecture changes, and support test and evaluation, especially end-to-end performance. However, use was limited mostly to those situations where there were existing models and simulations available and these existing models and simulations were at an appropriate fidelity for the area of interest [DoD, 2008].

To further probe the use of M&S in the SoS environment, a survey of the National Defense Industrial Association (NDIA) M&S Committee was conducted. The NDIA M&S committee membership includes M&S experts from industry, government, and academia. This group was asked to identify actual experiences with M&S in the SoS environment, potential applications of M&S that have not yet been realized, and enablers and inhibitors to M&S in the SoS environment. The NDIA M&S survey report [Dahmann et al., 2009] identified some limited use of models to support SoSE along with considerable modeling opportunities. These include:

- Analysis of organizational relationships among the systems
- Identification and analysis of mission scenarios and concepts of operation to support analysis of capability options and SoS-level testing
- Identification of SoS configurations and evolution options
- Constituent system interoperability assessment
- Capability gap analysis
- Cost modeling and estimation to support alternative analyses
Dynamic models and simulations to further understand capabilities, missions, and emergent behaviors and support assessment and trade analyses of capability alternatives and overall performance.

Table II presents an elaboration of the survey findings with respect to the key SoSE activities associated with each core element. The goal of the rest of this paper is to show how SysML can be used to better understand the current SoS and its

<table>
<thead>
<tr>
<th>SoSE Core Element [DoD, 2008]</th>
<th>Key SoSE Activities [DoD, 2008]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translating Capability Objectives</td>
<td>Understand the nature and dynamics of the SoS and anticipate areas likely to vary over time</td>
</tr>
<tr>
<td></td>
<td>Define functions to provide desired capability in a way that accommodates the variability in the environment and the different situations in which the capability may be executed</td>
</tr>
<tr>
<td></td>
<td>Evaluate the operational utility of the SoS and derive requirements</td>
</tr>
<tr>
<td>Understanding Systems and Their Relationships</td>
<td>Focus on components and dynamics rather than boundaries</td>
</tr>
<tr>
<td></td>
<td>Gain an understanding of the ensemble of constituent systems that affect the SoS capability and the way the constituent systems interact and contribute to capability objectives</td>
</tr>
<tr>
<td></td>
<td>Consider key systems that may be outside the direct control of SoS management but have large impacts on SoS objectives</td>
</tr>
<tr>
<td></td>
<td>Define functionality of the constituent systems and how they share data during operations</td>
</tr>
<tr>
<td></td>
<td>Understand the potential impact of plans for the constituent systems and SoS</td>
</tr>
<tr>
<td>Assessing Performance to Capability Objectives</td>
<td>Focus assessments on end-to-end performance</td>
</tr>
<tr>
<td></td>
<td>Pro-actively identify changes needed, emergent behavior, and impacts on the SoS resulting from constituent system changes</td>
</tr>
<tr>
<td>Developing and Evolving SoS Architecture</td>
<td>Establish a persistent technical framework for addressing the evolution of the SoS in response to user needs</td>
</tr>
<tr>
<td></td>
<td>Analyze and assess trades among different options</td>
</tr>
<tr>
<td></td>
<td>Conduct focused investigations of functionality and relationships to address core issues</td>
</tr>
<tr>
<td></td>
<td>Evaluate the extent to which architecture-related changes in the constituent systems are affordable and feasible</td>
</tr>
<tr>
<td></td>
<td>Capture the functionality that the individual constituent systems contribute to the SoS in a functional architecture</td>
</tr>
<tr>
<td></td>
<td>Define and maintain a functional architecture that details the complete set of functions to be performed within the SoS as well as the relationships among the functions</td>
</tr>
<tr>
<td></td>
<td>Define and maintain the physical architecture that identifies the constituent systems within the SoS</td>
</tr>
<tr>
<td></td>
<td>Understand the variability in the execution of SoS functions and factor it into the SoS architecture</td>
</tr>
<tr>
<td></td>
<td>Regularly assess the architecture to ensure that it supports the SoS evolution</td>
</tr>
<tr>
<td>Monitoring and Assessing Changes</td>
<td>Continually monitor constituent system proposed or potential changes and assess their impact to the SoS</td>
</tr>
<tr>
<td></td>
<td>For constituent system changes that may adversely affect the SoS, identify more desirable alternatives for implementing the changes and work to influence the constituent systems to adopt the alternatives</td>
</tr>
<tr>
<td>Orchestrating Upgrades</td>
<td>Integrate, verify, and validate changes across the suite of constituent system updates within an SoS increment</td>
</tr>
</tbody>
</table>
capabilities and inform SoSE key decisions in evolving the SoS over time.

3. SysML SUPPORT OF SoSE

System modeling languages and frameworks such as SysML and the DoD Architecture Framework (DODAF) provide a rich set of model types to support the engineering and development of systems from the very small to the very large. These modeling languages and frameworks were never intended to be a "one-size-fits-all" tool, but rather a set of tools to help engineers reason about a problem and a problem space. With today’s large, complex systems and SoS, there are not enough resources (people, schedule, funding) to model everything in extreme detail. Experienced system modelers have found that they need to focus on the system (or SoS) aspects that are important to the problem and problem space and then abstract the rest away as described in Moneva and Vissers [2010]. Consistent with the mantra of Moneva and Vissers to focus on the problem at hand and abstract the rest away, the SysML modeling discussed here is a tailoring of the more traditional systems engineering modeling that focuses on what is important to the SoS engineer: how to understand and maintain existing SoS capabilities while composing new capabilities from the set of systems that are considered to be part of the SoS.

While all of the SoSE core elements can benefit from modeling, the modeling activities listed above primarily support Translating Capability Objectives, Understanding Systems and Their Relationships, Monitoring and Assessing Changes, Developing and Evolving SoS Architecture, and Addressing Requirements and Solution Options. This section describes how several SysML models can be used to support most of these needs, starting with some general concepts on model-driven software-intensive system development.

3.1. Overview of Model-Driven System Development and Evolution for SoS

Model-Driven Systems Development (MDSD) [Bohn et al., 2008] is a technique for modeling a complex system or SoS that is itself composed of complex subsystems (or constituent systems). The value of this approach comes from its ability to look at customers or users of the system or SoS and how they expect to operate in an environment. This approach provides a holistic view of the system (or SoS) and the means to ensure that the requirements of the individual parts effectively integrate to meet the needs of the stakeholders.

The key benefits of the MDSD approach are:

- The use cases, requirements and interfaces defined by the parts can be validated against the top level use cases.
- Interfaces can be clearly distinguished and described in terms of attributes and operations.
- The MDSD model can be used to derive end-to-end test cases for the system or SoS, as appropriate.
- The architecture provides the SoSE team with a context that can ensure complete coverage of the SoS and each of its constituent elements.

Figure 2 illustrates key model constructs used to characterize SoSs and SoS capabilities. These are use cases that describe scenarios, actors that are either people or systems performing activities within the scenario, system blocks that either represent a constituent system or the SoS, and arrows that illustrate interactions from either outside entities or other constituent systems.

The system or SoS modeling process, illustrated in Figure 3, starts by setting the context: understanding what is in the system (or SoS), and what is not. The context also includes who and what will interact with the system (or SoS), and what information will be passed to and from the system (or SoS). These things are usually captured in a context diagram.

After doing the context work to understand the scope and context, the next step is the work of deriving the top level services or capabilities (Use Cases—sometimes referred to as enterprise level use cases) that the SoS provides to the environment and other external entities (Actors). This list of services (or capabilities) is analyzed and refined by the stakeholders and analysts from the engineering team.

The engineering team then works on the Use Case Specifications. Each specification lists the major actions necessary to perform the use case and all of the alternate actions. Each action that requires an interaction between the environment (represented by the actors) and the SoS forms the basis for a
request on the Black Box\textsuperscript{1} Sequence Diagram which may or may not involve a response.

\textsuperscript{1}"Black Box" indicates that the details of what is inside the box are hidden. These types of diagrams focus on the interfaces to/from the outside world.

Next the engineering team builds the Black Box Sequence Diagrams to show the flow of the requests that pass between the SoS and the environment (Actors). These requests form the basis for the SoS-level operations.

The team then takes each SoS-level operation and produces an Operation Specification. The Operation Specifica-
tion documents the actions necessary to complete the operation and describes the interactions between the constituent systems and the external entities. These actions form the basis for the requests in the White Box\(^2\) Sequence diagrams.

The last part of the SoS-level work is producing the White Box Sequence Diagrams. These diagrams depict the flow of requests between the constituent systems, and between the constituent systems and the external entities.

This process of decomposition using the Operation Specifications and White Box Sequence diagrams continues until the level of detail required to generate the solution is reached. And while these steps are described in an orderly fashion, much is done in parallel and in iterations in real practice.

To summarize, when first starting to model an SoS, it is important to start at the highest level, first viewing the SoS as a black box and focusing on the SoS capabilities and associated external inputs and outputs, then working down to lower levels of detail by replacing the SoS-level black box with an SoS white box that represents each of the constituent systems as a black box. Then those constituent system black boxes can be translated into constituent system white boxes with assistance from the constituent systems engineers, as needed to better understand constituent system capabilities and functions.

This process is consistent with the findings in Dahmann et al. [2009] that emphasize that the SoS focus is at the SoS level, paying attention to the user needs and associated scenarios, the constituent systems that support the SoS capabilities, and the interfaces and data that the constituents use to provide the capability. The SoSE team works with the constituent systems to understand the aspects of the constituent systems important to the SoS capabilities, but seldom gets involved in the details of the constituent system internals—that is left to the constituent system engineers. Therefore, SoSE modeling support typically stays focused at the SoS level.

### 3.2. SysML Model Elements To Support SoSE

To understand an SoS, one models the SoS of interest at different levels with one or more views. At the highest level of an SoS, the key models convey the SoS capabilities and the constituent systems that work together to provide the identified capabilities. For the capability view, one might define the SoS as a single “black-box” object, with a list of capabilities. To characterize the SoS constituent systems, one would start by developing a black box context diagram for each constituent system. For the white box characterization of the constituent system and its capabilities, one would generate an object block or class\(^3\) that describes the characteristics and capabilities of the constituent system.

Each SoS capability can be modeled from two viewpoints: the user perspective and the constituent system perspective. For the user perspective, one generates a set of use cases. These use cases can be used to characterize both system and SoS capabilities from the different user perspectives. For the constituent system perspective, one generates a sequence diagram. Sequence diagrams are used to characterize and analyze the operational flow for an SoS capability. The SoS sequence diagrams can be developed from different viewpoints, such as logical and distribution, in order to reason about system functionality as well as where the functionality should be deployed.

To understand the SoS interfaces between constituent systems and external entities, one can generate an interface class for each interface that describes the characteristics of the interface. This is essentially an object class used to characterize the interface. To further understand and characterize an interface, one generates a set of input/output (I/O) entity classes to describe the associated data attributes of each data item transferred over that interface.

The final desired area of modeling support for SoSE identified in Dahmann et al. [2009] is cost estimation. While SysML does not perform cost modeling, the models developed using SysML can support cost estimation. A cost model that supports the estimation of SoSE effort for a new or enhanced capability is the Constructive Systems Engineering Cost Model (COSYSMO) for SoSE [Lane, 2009]. The SoSE cost model size drivers that are used to estimate SoSE effort are the number of requirements, number of interfaces, number of algorithms, and number of operational scenarios. In addition, each of these size drivers can be rated with respect to complexity or difficulty. The SysML models can be used to capture the number of interfaces and number of operational scenarios size drivers for the SoSE cost model. In addition, one could capture system algorithm information for a constituent system by specifying this detail either as part of the constituent system’s services or functions or as part of the data element characteristics (e.g., how the data element was calculated, what coordinate system was used, units associated with the value, timing associated with the calculation and refresh durations).

### 4. SoS EXAMPLE USING SysML

To illustrate the use of these SysML models, consider the example of the Regional Area Crisis Response SoS (RACRS). RACRS is a fictitious SoS and is for illustrative purposes only. While it may reflect existing capabilities being currently developed and employed in various regions, it is not based upon any actual SoS. For this modeling example, assume that the goal is to model the “as-is” SoS in order evaluate alternatives for a major upgrade.

The motivation for RACRS is based upon catastrophes that have happened in recent years within the United States: Hurricane Katrina, devastating fires in California, powerful earthquakes in the western United States, and tornadoes in the midwest United States. Early responders to these incidents found that communications between the different local agencies were difficult at best and often not integrated. When state and federal agencies became involved, the communications problems escalated. As a result, efforts have been initiated to establish a way to better integrate the needed agencies in response to a given incident. The goal is that the agencies will generally operate on their own outside of the SoS and then
quickly be able to dynamically reconfigure and join the re-
gional SoS as needed, typically in response to an incident.
The scenario and related capabilities of interest for this
example are fire-fighting for large, fast moving fires. The
candidates of interest for this version of the SoS are:

- **Satellite imaging system**: Provides images and video of
  interest to requestor.
- **Fire department**: Manages the fire response units.
- **Police department/sheriff’s department**: Provides
  safety and crime-fighting support that includes evacu-
  ation support and protection from looters.
- **Handheld devices**: Provides connectivity to crisis re-
  sponders on the ground via voice and video.
- **Reverse-911**: Automatically sends voice messages to
  people that reside or work in areas that need to be
  immediately evacuated.
- **Regional area planning and land use**: Includes building
  plans, building locations, and maps for utilities (elec-
  tricity, water, sewer) for regional areas of interest, as
  well as the capabilities for getting the information.
- **Unmanned aerial vehicles (UAVs)**: Used for surveil-
  lence, lightweight fire retardant drops, or to fire upon
  looters/rioters.
- **Unmanned ground vehicle (UGV)**: Provides on
  the ground video feeds in situations where it is too danger-
  ous for personnel, clearing of brush/small trees to create
  fire breaks, and can also be armed to start needed
  backfires.
- **Aerial water tanker**: State/national asset shared among
  multiple regional areas to drop water or fire retardant
  on hot spots.
- **News helicopter**: Used to capture video feeds for news
  programs—includes news events as well as traffic flows
  and may also be used to monitor for signs of looting.
- **Command and control center (CCC)**: Central site to
  monitor and help coordinate activities and support de-
  cision makers.

The constituent systems that interoperate with the CCC for
the RACRS fire-fighting scenario are illustrated in the CCC
context diagram shown in Figure 4. This is the “black box”
view of the CCC and is used to understand at the top level the
constituent systems related to fire-fighting from the CCC
point of view.

Context diagrams aren’t something new; they’ve been used
for decades. The problem has been that, in the past, they were
just diagrams. The elements had no semantic connection to
what was supposed to be the same thing on another diagram.
This example illustrates the use of a SysML Block Definition
Diagram (along with Stereotypes and custom icons) to create
a context diagram where the elements on it are the same
wherever they’re used. This can be extremely useful when
maintaining a large model.

Figure 5 shows the high level uses cases for the fire-fight-
ing scenario. This figure identifies four fire-fighting uses
cases: Evacuate Area, Conduct Fire Suppression, Get Topo
Map Info, and Get Map Info. It also shows the actors that
participate in each use case. Because this is the fire-fighting
situation, it is the Fire Control actor that initiates each activity
through the CCC. (For other situations, there might be other
actors that could initiate one of the listed use cases.) Each use
case diagram is then further analyzed and elaborated by
writing a use case specification. The use case specification

---

**Figure 4.** CCC context diagram. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com.]
contains a brief description of the use case and describes the flow of events, any alternative flows for special situations or conditions, any preconditions that must exist in the system prior to the use case being performed, and any post conditions that identify the states that the system can be in after the use case has finished.

The next step in the process is to translate and expand each use case specification into a sequence diagram. The sequence diagram for the Evacuate Area is illustrated in Figure 6. This sequence diagram shows the messaging between Fire Control, CCC, Police Control, Reverse 911, and UAV and the information that flows between them.

Figure 6 shows the initiation of the “evacuate area” command from the Fire Control center, and then how the CCC triggers the Police Control, Reverse-911, and UAV activities. Once these three actors report success, the “evacuated” status is sent to Fire Control. However, the UAV is still monitoring for looters in the evacuated area. If the UAV reports an intruder, CCC notifies the police who have the option of telling the UAV to “eliminate” the looter(s). This is illustrated in the “alt” scenario in Figure 6.

Next is the process of modeling the interfaces and the data that are transmitted across each interface. Figure 7 shows the interface diagram for the CCC, the diagrams for the constituent systems participating in the Evacuate Area use case, and the associated I/O entities for each system. This diagram identifies all of the interfaces from/to CCC for the evacuate-area use case.

The I/O entities for the CCC fire-fighting evacuate area use case are: Area which is indicated by latitude and longitude, Reporting Entity for each actor that reports, Acknowledgment for evacuate command, and Call Type for evacuate command. Finally, I/O entity diagrams are provided for each actor, showing the relationships of the various I/O entities by actor. The I/O entities for the actors participating in the Evacuate Area use case are shown in the lower part of Figure 7 along with each actor’s related I/O entities.

Once one has modeled the “as-is” SoS, the models can be used to explore alternatives for new capabilities and performance enhancements. This includes identification of alternatives, existing gaps for each alternative, data or functional inconsistencies between the constituent systems (e.g., algorithm compatibility, coordinate systems used, data element units, data timing/aging), and views of the “to-be” SoS architecture and capabilities.

5. CONCLUSIONS

Recent research with respect to SoSE has indicated that modeling support would be useful if models could be quickly generated and used to support needed decision making. The goal of this paper was to show how SysML models can be used to support some of these SoSE needs. This paper has shown how context diagrams, use cases, object blocks, interface classes, and I/O classes can be used to model the SoS, SoS constituent systems, and the associated capabilities at both the SoS and constituent system level. In addition, these models provide needed insights into SoS and constituent system interfaces and associated data attributes to ensure interoperability of the constituent systems. However, it should
Figure 6. Evacuate area sequence diagram.

Figure 7. CCC interface class and evacuate area I/O entities by actor.
be noted that SysML is not the total SoS modeling solution. For example, being able to dynamically execute models to identify and evaluate SoS emergent behaviors or performance is not something SysML was designed to do. IBM’s Rhapsody will do simulated execution using state transition diagrams, but that is more of a tool capability than something SysML has brought to the table. Other tools like Matlab® or Engineous4 iSIGHT are much more powerful for executable simulations using models.

Even though SysML models cannot meet all of the SoSE modeling needs identified in DoD [2009] and Dahmann et al. [2009], the SysML models can capture information distilled from a multitude of constituent system documents and engineering experts and then integrate them to better support SoSE. The models can use the captured information to more easily analyze end-to-end performance of SoS mission scenarios and evaluate new capability alternatives or proposed architecture changes. These models can also highlight the impacts of proposed changes: which constituent systems will be affected by a set of changes as well as how much the constituent systems will be affected. The “how much” assessment can be accomplished through the use of the cost model, COSYSMO for SoSE [Lane, 2009], by estimating the engineering effort associated with each option.

Table III highlights some of the key core element support and SoSE needs that can be provided by each SysML model presented in this paper as well as the support for cost estimation of new SoS capabilities.

The key to success in modeling SoSs is to only model the aspects that are important for the engineering activities performed at the SoS level and to abstract the rest away. Also important is to consider using models in ways that they were not originally intended, for example, context diagrams using system block definition diagrams and the I/O entity classes to describe data attributes for data shared between multiple constituent systems. One should look at the SysML model types as tools that can be repurposed to help understand areas or aspects of interest. Through creative uses of SysML, this paper has shown how SoSE engineers can capture and evaluate SoS-related information critical to exploring options for SoS capabilities and estimating the engineering effort associated with each option.

REFERENCES


J. Lane and B. Boehm, System of systems lead system integrators: Where do they spend their time and what makes them more or less efficient, Syst Eng 11(1) (2008), 1–11.


H. Moneva and J. Vissers, Putting chaos under control: On how modeling should support design, Proc 20th Int Symp INCOSE, 2010.


*Engineous was acquired by Dassault Systèmes in 2008.

<table>
<thead>
<tr>
<th>SysML Model</th>
<th>SoSE Activity/Needs Supported</th>
</tr>
</thead>
</table>
| Context diagram                 | • Depiction of relationships among the systems  
|                                 | • Identification of SoS configurations                                                       |
| Use case diagram                | • Identification of scenarios and capability options  
|                                 | • Support SoS-level testing  
|                                 | • Identification of capability gaps  
|                                 | • Understand variability in the execution of SoS functions  
|                                 | • SoSE capability cost estimation                                                           |
| Sequence diagram                | • Sequence of tasks to perform SoS capability  
|                                 | • Identification of capability gaps  
|                                 | • Addressing requirements and solution options  
|                                 | • SoSE capability cost estimation                                                           |
| Interface class diagram         | • Understanding systems: Depiction of relationships among the systems  
|                                 | • Developing SoS architecture  
|                                 | • SoSE capability cost estimation                                                           |
| I/O entity class diagram        | • Understanding systems  
|                                 | • Constituent system interoperability assessment/identification of data inconsistencies (or gaps)  
|                                 | • Monitoring and assessing changes  
|                                 | • SoSE capability cost estimation                                                           |

Table III. SysML Support of SoSE for the Evolution of Existing SoSs
Jo Ann Lane is a research assistant professor at the University of Southern California Center for Systems and Software Engineering, conducting research in the areas of software engineering, systems engineering, and system of systems engineering (SoSE). She was a coauthor of the 2008 Department of Defense Systems Engineering Guide for Systems of Systems. Current areas of research include SoSE processes, SoSE cost modeling, and SoS constituent system interoperability. Prior to her current work in academia, she was a key technical member of Science Applications International Corporation’s Software and Systems Integration Group for over 20 years, responsible for the development and integration of software-intensive systems and systems of systems. She received her Ph.D. in Systems Engineering from the University of Southern California and her Master’s in Computer Science from San Diego State University.

Tim Bohn has worked as a systems engineer and software engineer for 30+ years. He was recently the Worldwide Community of Practice Leader for IBM Rational’s Systems Development Community. This community is comprised of a diverse worldwide community of practitioners within IBM and its partners. Tim has been active in the systems community, helping customers adopt Model-Driven Systems Development practices for system and enterprise engineering. Currently Tim is with IBM’s Strategy and Technology Office, doing research and presenting competitive analyses of IBM’s systems hardware and software. Tim received his Bachelor’s and Master’s in Computer Science from the University of Southern California Los Angeles.