31st Annual INCOSE international symposium

virtual event

July 17 - 22, 2021

Formulas and Guidelines for Deriving Functional System Requirements from a Systems Engineering Model

Created by John Shelton, Victoria Heisler, and Kristina Sebacher

Contents

Our Task

- Our Assignment
  - Create English-language Functional Requirements in a traditional pdf book for the client.
- Assumptions
  - We would be using MagicDraw with SysML and UPDM2 add-ons for modeling. There was no budget or schedule for additional tool acquisition or comparison.

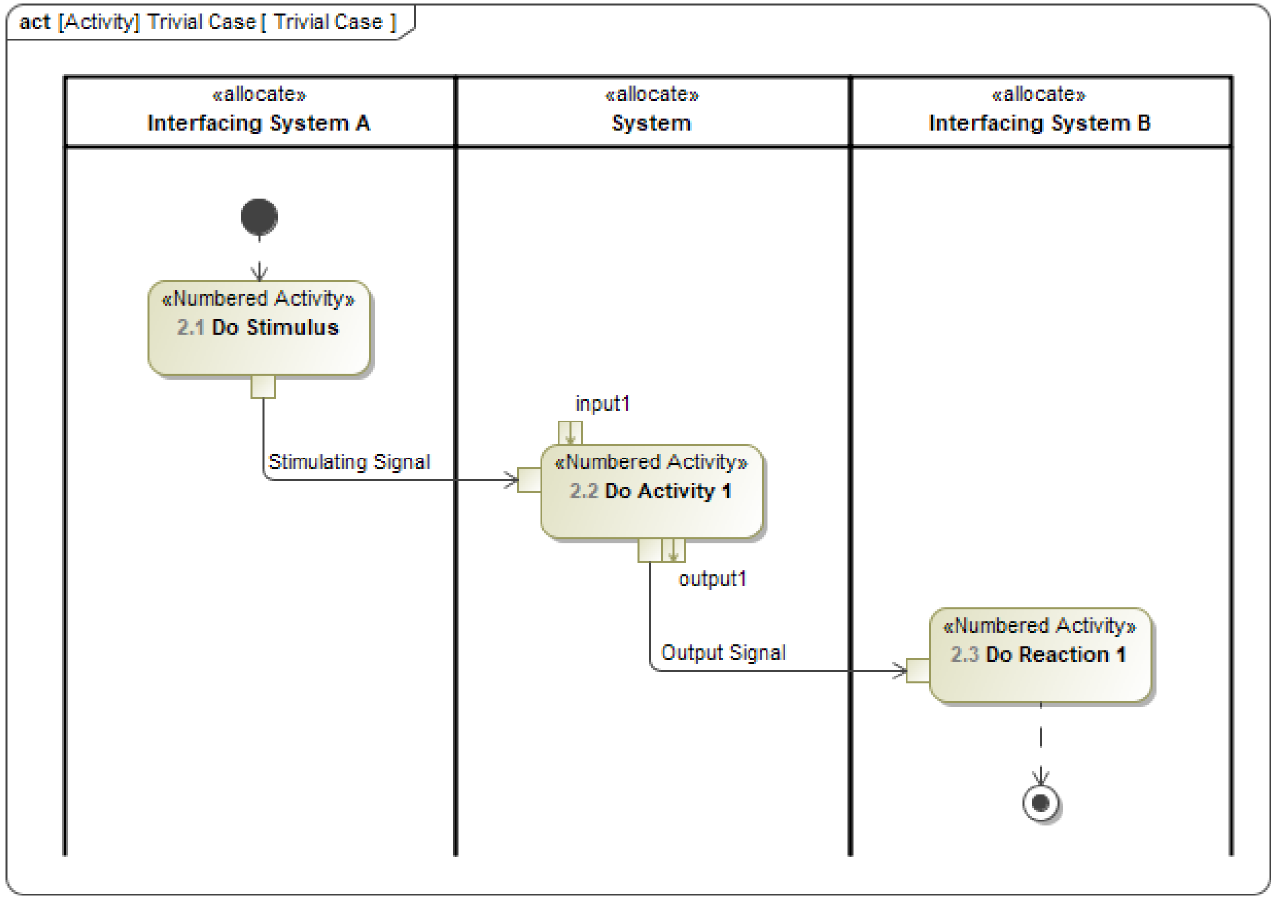
Our Conviction

- Modeling should have primacy.
  - Model first!
  - Center technical discussion around the model. Discuss it, feed back, and repeat until “done”.

- MBSE is not SEBM.

Our Approach: Formulaic Requirement Authorship

Part 1: Produce SysML Activity Diagrams…


act [Activity] Trivial Case [Trivial Case]
«allocate» Interfacing System A
«allocate» System
«allocate» Interfacing System B
«Numbered Activity» 2.1 Do Stimulus
Stimulating Signal
input1
«Numbered Activity» 2.2 Do Activity 1
output1
Output Signal
«Numbered Activity» 2.3 Do Reaction 1

Part 2: …then formulaically produce English-language requirements, one per allocated action.

- Requirement Text: Upon receipt of [Stimulating Signal], the System shall [Do Activity 1], [producing/displaying] [Output Signal].
- Rationale: See [Activity #] in model. [Add custom thoughts as needed.]
- Requirement Type: Functional

Issues to be Overcome

Definitions

- What is a Functional Requirement?
  - Our client wanted functional requirements to be distinct from performance requirements. Many definitions do not distinguish between them.
  - Based on the context of our client's request, we defined a functional requirement as one which describes system behavior *without qualification or quantification*.
  - Requirements which provide qualification or quantification are defined as Performance Requirements, and are outside of the scope of this effort.
  - By this definition, Functional Requirements should generally be defined prior to Performance Requirements.
    - Once function is understood, performance parameters can be defined. Some will be common. Others will be custom.

- Examples:
  - Upon command from the driver, the garage door shall open. — Functional
  - Upon command from the driver, the garage door shall open within x seconds.
  - Upon command from the driver, the garage door shall open with x-sigma reliability.
  - Upon command from the driver, the garage door shall open to x percent of its capacity.

  Performance

Issues to be Overcome

Overall Process Description

- We wanted to abide by our conviction to be model-based. But we found little guidance for creating system-level requirements from a model.
  - OOSEM assumes we already have requirements in Step 1!

- We were initially given 3 months to complete the task. We suspected that we might get more time, once we demonstrated the value of the approach. So we established an agile approach to add value on Day 1. This was key to our success!
  - But what does an agile modeling process intended to support requirement authorship look like? Again, we found little guidance.

Issues to be Overcome

More Definitions

- What is an overly prescriptive requirement?
  - Common definition: A requirement that prescribes behavior at levels lower than the SOI.
  - Model Symptom: Actions in an Activity Diagram occurring serially, perhaps attributed to sub-systems, without interaction with other systems or actors. These behaviors should be consolidated / abstracted / stated more simply.
- What is an under-prescribed requirement?
  - Common definition: We found little guidance to prevent under-prescription of requirements.
  - Model Symptom: Actions in an Activity Diagram which cannot be solely assigned to a single system. Does it require interaction with another system or actor in order to complete? If so, it needs to be decomposed further!
- What is a testable requirement?
  - Repeatability – The conditions under which the requirement is applicable should be defined. Adding conditions to a requirement gave some people pause; we defined "Compound Requirements" more rigorously in order to make this work.
  - Observability – The behavior should be verifiable without invasive instrumentation.
  - Other considerations.

Issues to be Overcome

SysML and Modeling Norms

- Parameter Objects:
  - In such an early complex architecting effort, basic definitions and usages of parameters constantly change, making parameters a problematic object type.

- Requirement Objects:
  - Since our goal was to generate requirements, there was a temptation to use requirement objects in our model. We found them to be redundant with the Activity diagrams, so in the end, we did not use them.

Our Approach

Overview

- Use a series of formulas to translate SysML into English book-form requirements.
- There should be one functional requirement generated per action allocated to the System of Interest.
- Formulas should be combinable, to account for any contextual situation.

Approach Expansion: The Modeling Part

The 3 Rules of Activity Diagram Modeling

1. Allocate Actions to Systems Using Swimlanes
2. Model External Stimuli and Observable Outputs
3. Model Serial Actions with Care

The 3 Rules of Activity Diagram Modeling

Rule 1: Allocate Actions to Systems Using Swimlanes

- Only allocate an action to a system when all expected sub-actions are performed internal to that system.

- If it is possible that some sub-behaviors will be performed by external systems, then the behavior is too abstract. Additional discussion is needed!

- Use hierarchical activity diagrams to decompose until this level of maturity is reached.

- Use token manipulators (splitters/combiners, send/receive signals, decision/merge nodes, etc) frequently to model parallel activities, characterizing the system’s behavior generically. Scenario modeling is insufficient.

The 3 Rules of Activity Diagram Modeling

Rule 2: Model Stimuli and Observable Outputs

- Functions which are testable should be stimulated from outside the system.
  - Not every Activity Diagram will show an external stimulus. But if you trace a behavior back through the series of diagrams to its origin, it should kick off with an external stimulus.
  - Sometimes the time scale is long, and we talk about “state-based behaviors”. But even a state-based behavior should kick off with a state change which is stimulated from outside of the system. No system behavior should be completely spontaneous; spontaneous behavior is not testable.

- Functions which are testable should be observable from outside the system.
  - Information should cross the system boundary, outbound, to be detected without invasive instrumentation.

The 3 Rules of Activity Diagram Modeling

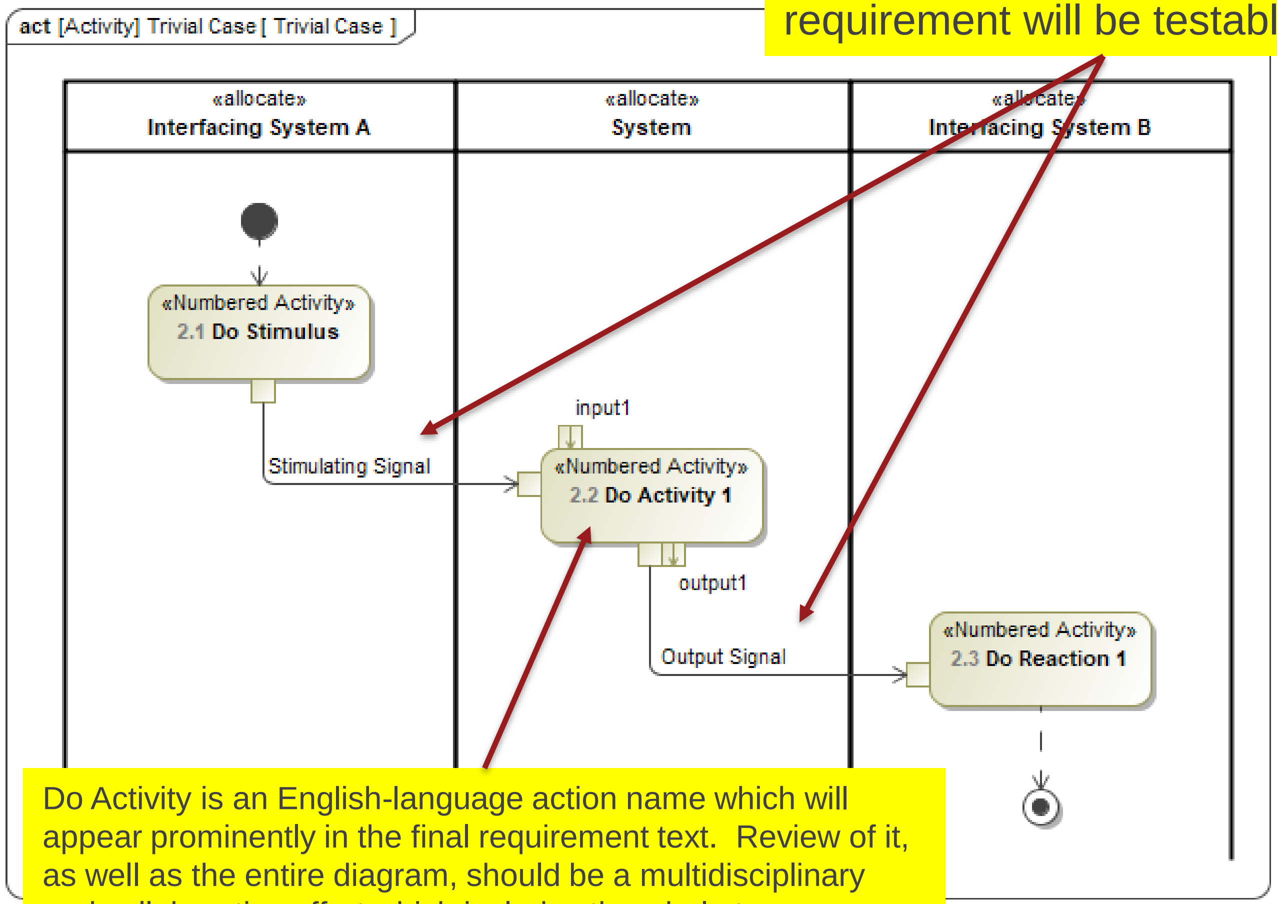
Rule 3: Model Serial Actions with Care

- Serial Actions are actions allocated to a single system in series, with no external inputs or outputs in between.
- Serial Actions are a symptom of over-prescription, and can lead to higher numbers of functional requirements to manage. They should be avoided when possible.
- They are not always avoidable. Sometimes they are necessary to reduce ambiguity, or to model a decision-making process where externally-observable outputs vary depending upon the outcome of that process.

Approach Expansion: The Translation Part

The Trivial Case

An external stimulus and externally-observable output heightens the chances that this requirement will be testable.



Do Activity is an English-language action name which will appear prominently in the final requirement text. Review of it, as well as the entire diagram, should be a multidisciplinary and collaborative effort which includes the whole team.

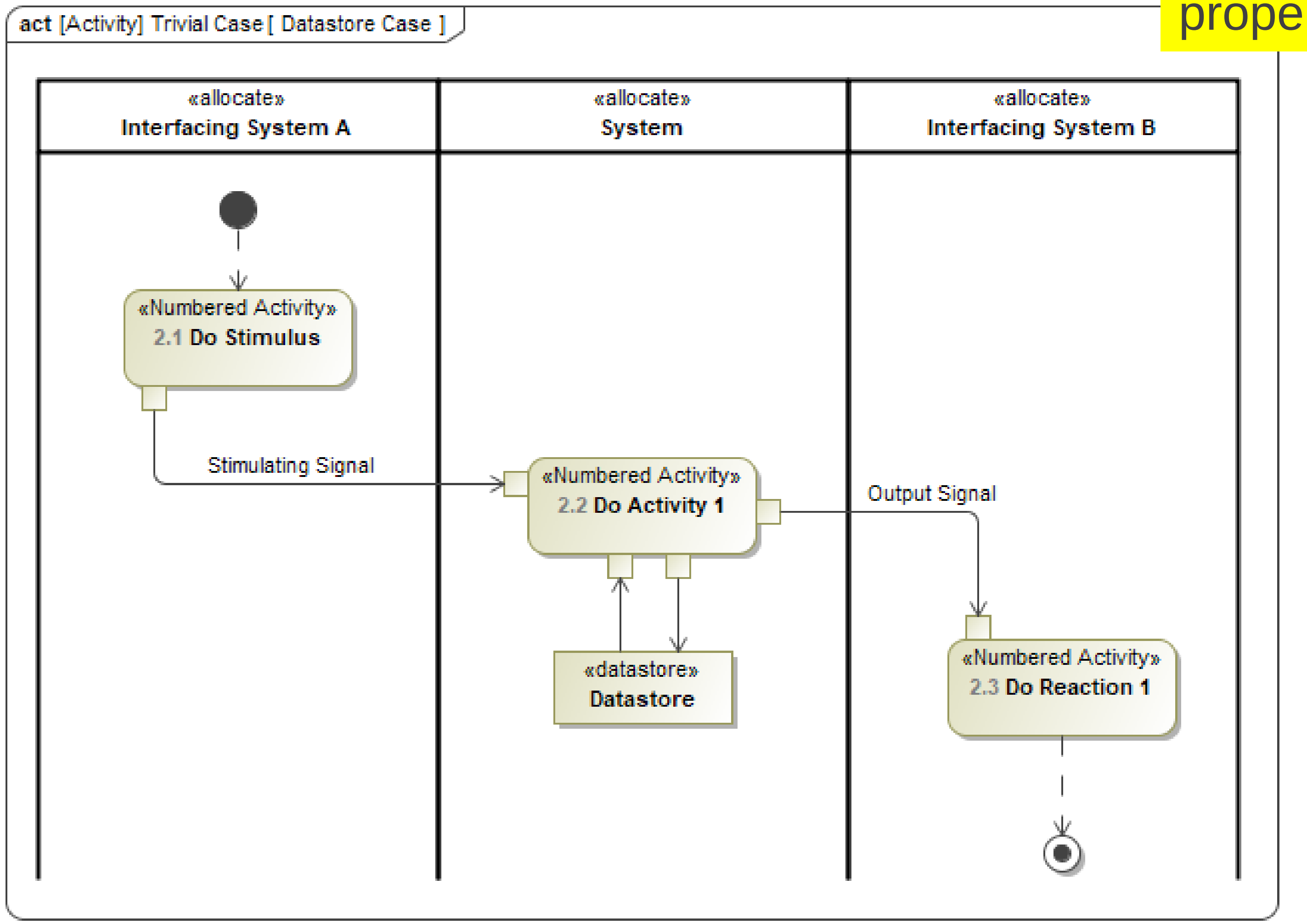
Formulaic requirement metadata reduces time spent in authorship. Other metadata can be added to the model, and output here, as well. Possibilities include unique identifier numbers, classification tags, safety tags, verification methodology tags, and more.

- Requirement Text: Upon receipt of [Stimulating Signal], the System shall [Do Activity 1], [producing/displaying] [Output Signal].
- Rationale: See [Activity #] in model. [Add custom thoughts as needed.]
- Requirement Type: Functional

Approach: Translation

Datastores

Datastores were not used conventionally! Parameters would have been more proper, and more problematic.


act [Activity] Trivial Case [Datastore Case]
«allocate» Interfacing System A
«allocate» System
«allocate» Interfacing System B
«Numbered Activity» 2.1 Do Stimulus
Stimulating Signal
«Numbered Activity» 2.2 Do Activity 1
Output Signal
«datastore» Datastore
«Numbered Activity» 2.3 Do Reaction 1

When tokens flow from Datastore into Do Activity 1, but not the other way.
Requirement Text: Upon receipt of [Stimulating Signal], the System shall utilize [Datastore] to [Do Activity 1], [producing/displaying] [Output Signal].

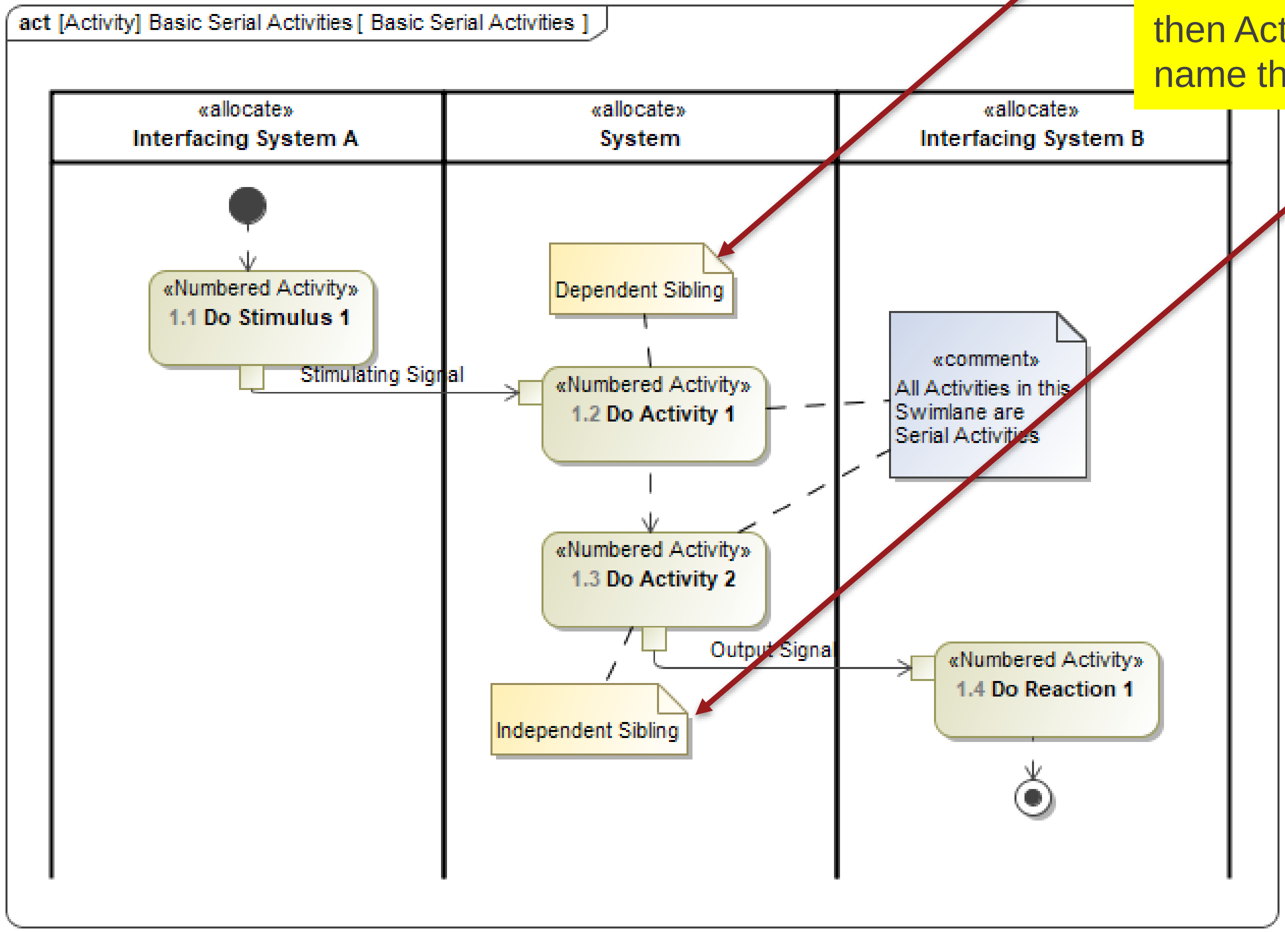
Or

When tokens flow both into and out of Do Activity 1 from the datastore.
Requirement Text: Upon receipt of [Stimulating Signal], the System shall [Do Activity 1], [producing/displaying] [Output Signal], and updating [Datastore].

Solution: Translation

Serial Actions

We defined "dependency" from a tester's perspective. A requirement is "independently verifiable" if the output is directly observable. A requirement is "dependently verifiable" if it is verified by observing output from other serial activities downstream. This dependency has certain assumptions that the modeler should be cognizant of in their language choices. If Activity 2 could occur without Activity 1 occurring, then Activity 1 is not verifiable. The language chosen to name these actions must reflect this logic.


act [Activity] Basic Serial Activities [Basic Serial Activities]
«allocate» Interfacing System A
«allocate» System
«allocate» Interfacing System B
«Numbered Activity» 1.1 Do Stimulus 1
Stimulating Signal
Dependent Sibling
«Numbered Activity» 1.2 Do Activity 1
«comment» All Activities in this Swimlane are Serial Activities
«Numbered Activity» 1.3 Do Activity 2
Output Signal
Independent Sibling
«Numbered Activity» 1.4 Do Reaction 1

Used for the Dependent Sibling

Requirement Text: Upon receipt of [Stimulating Signal], the System shall [Do Activity 1].

And

Used for the Independent Sibling

Requirement Text: Upon [Do Activity 1], the System shall [Do Activity 2], [producing/displaying] [Output Signal].

Our Approach

Other Formulas

- Send/Receive Signals in Activity Diagrams
- Interruptible Regions in Activity Diagrams
- Redundancy of Language in Activity Diagrams
- Streaming Behaviors
- Timer Objects
- Functional Constraints

And all of them need to be combined, when they all co-exist on a diagram.

Our Results

Strengths and Weaknesses of the Approach Made Apparent in Implementation

- Current Status
  - The system is yet to be defined and built. But we are confident that the client and bidders have a unified understanding of the SoS's expected behaviors and the system's functional requirements.
- The Silver Bullet Process
  - We needed an outlet valve to write requirements for behaviors that we didn't model all the way due to time or budget constraints. We called this our "silver bullet process". Our silver bullet process was to add the behavior as a high-level SoS action, without decomposing it. We then wrote a requirement that the system "support" that capability.
- Number and Detail of Requirements
  - This process outputs a lot of functional requirements. There were many more, and more detailed, than some clients were accustomed to.

Our Results

Strengths and Weaknesses of the Approach Made Apparent in Implementation

- Team Structure
  - MBSE is best when it's collaborative and multi-disciplinary. Teams should be set up to support these precepts. Team silos are the enemy of MBSE.

- Definition of Truth
  - This approach does not resolve the question: "What is truth?" This is especially important in a situation where multiple artifacts describe the same phenomena. Here, we have a model and a requirements document both describing system behavior.
  - Typically, a client-contractor governing contractor will point to a set of requirements which is the "true set". The English-language requirement document is likely to be that "true set", as it was for us. But this could evolve as MBSE gains traction in the community.

- The Future
  - Extending The Process to Other Requirement Types
    - Authorship of other requirement types, such as architectural and performance requirements, could benefit from a similar approach.
    - Our team has already created a formulaic approach to architectural requirement authorship.
  - Firming up This Process
    - Use of parameters instead of datastores might be incorporated with greater tool and language fluency.
    - We found that having guidance which describes the appropriate level for creating signals and datastores is important.
    - We also found that having team SOPs for modeling is important, if more than a few people are directly contributing to the model.

- You will hear from us again!

31st Annual INCOSE international symposium

virtual event

July 17 - 22, 2021

www.incose.org/symp2021