

Model-Based Structured Requirements in SysML

 Daniel R. Herber

 Colorado State University – Department of Systems Engineering
 daniel.herber@colostate.edu

INCOSE Requirements Working Group Meeting
March 30, 2023



→ Outline

1. Introduction
2. Model-Based Structured Requirements
3. Examples
4. Conclusion



①

Introduction

→ Introduction (1)

- Central to the rules governing a system are the requirements placed on it, often by various stakeholders
- These requirements help guide the system development process of a complex entity
- However, requirements do not specify how the system will meet these needs—it is up to the solution and project team to decide how the requirements shall be fulfilled
- Therefore, it is critical to have a well-defined, complete, and adaptable representation of the requirements
- There are a variety of approaches for developing and managing requirements¹
- For example, it has been established that a well-defined requirement shall possess the following general characteristics²:
 - *Necessary, Appropriate, Unambiguous, Complete, Singular, Feasible, Verifiable, Correct, and Conforming*

¹ Pohl 2010; ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering 2018; Ryan et al. 2019; R. Carson 2021; IEEE Guide for Developing System Requirements Specifications 1998 ² ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering 2018; Ryan et al. 2019

→ Introduction (2)

- Unfortunately, it is still a common problem of tremendous effort being spent understanding and interacting with the requirements along with multiple iterations because these guidelines and rules must be adhered to manually by engineers
 - These issues lead to increased costs and risks during development due to the poor quality of requirements defined during the early stages
-

- Now, architecture-centric practices are gaining widespread acceptance in systems engineering (SE)
- This process involves capturing the structure, behavior, and rules and their relationships to create an abstract representation of a system, often termed a model of the system
- A common language for creating such a model is the Systems Modeling Language (SysML)¹
- A system model does not inherently provide value to the SE effort—rather, benefits are realized through its support of various SE activities

¹ *OMG System Modeling Language* 2019; Friedenthal, Moore, and Steiner 2015

→ Introduction (3)

- An identified issue is that many approaches do not fully integrate requirements with the system model, including in classical SysML!
- For example, on page 309 of *A Practical Guide to SysML*:
“SysML includes a requirements modeling capability to provide a bridge between the text-based requirements that may be maintained in a requirements management tool and the system model. . . . This capability is intended to significantly improve requirements management throughout the lifecycle of a system by enabling rigorous traceability between the text-based requirements and the model elements that represent the system design, analysis, implementation, and test cases.
- This perspective has **less to do with defining** the requirements and **more with traceability** to the system model
- However, we put forth here that the requirements’ definitions themselves (in addition to traceability) can benefit from direct relationships to the system model and SysML constructs
- To this end, we will describe an approach for supporting rigorous model-based requirements using the existing concepts of structured requirements and SysML, termed model-based structured requirement (MBSR)

②

Model-Based Structured Requirements

→ Structured Requirements (1)

- A *structured requirement*, or a requirements template, defines an orderly requirement structure with specified attribute placeholders¹
 - Helps capture the precise meaning and communicate the required information to define a complete requirement
- For example, a structured requirement statement may look like:

The **[Who]** *shall* **[What]** **[How Well]** under **[Condition]**. (1)

- **[Who]** — Defines a subject term specified by an agent or user role that provides a capability or performs a function
- **[What]** — Refers to an action verb term specified by a required functionality or characteristic
- **[How Well]** — Indicates a comparison factor specified by constraints that can be applied to restrict the implementation of a required functionality or a design characteristic
- **[Condition]** — Describes the measurable qualitative or quantitative terms specified by characteristics such as an operational scenario, environmental condition, or a cause that is stipulated

¹ R. S. Carson 2015

→ Structured Requirements (2)

- For example, this a structured requirement written using Eq. (1) in natural language:

The [**actuation system**]_{Who} *shall* [**prevent inadvertent stowing**]_{What}
[**with at least three levels of safety**]_{How Well} under [**normal deploy op- (R1)**
eration]_{Condition}.

Remark

As currently presented, this is simply a refinement of the classical textual requirements approach and has only an imprecise relationship to the system model.

→ Structured Requirements (3)

- The orderly sequence of attributes for a structured requirement can be aligned with other approaches or standards
- Following some minor changes to the attributes, the ISO/IEC/IEEE 29148:2018 standard can be represented as¹:

[Condition] [Subject] [Action] [Object] [Constraint of Action]. (2)

- An example is²:

[At sea state 1]_{Condition} **[the Radar System]**_{Subject} **[shall detect]**_{Action}
[targets]_{Object} **[at ranges out to 100 nautical miles]**_{Constraint of Action} · (R2)

- Additionally, structured requirement forms have been proposed for different types of requirements (e.g., functional, non-functional, interface, design constraint, and operational) where the form and attributes are tailored to the specific needs of the different requirement types³

¹ ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering 2018 ² Ribaupierre et al. 2021 ³ Pohl 2010

→ Classical SysML Requirements Modeling (1): Definition

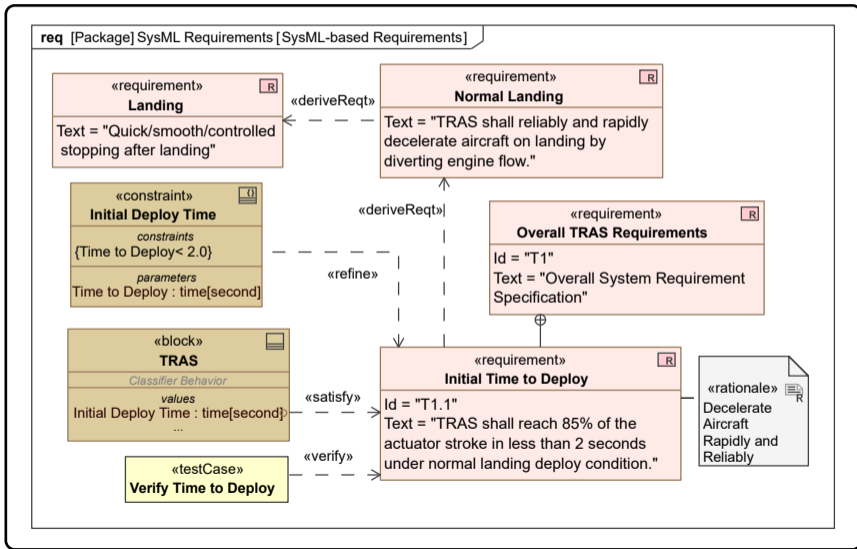
- As with many aspects of system modeling, SysML supports the inclusion of requirements¹
- A classical SysML-based requirement is developed by defining:
 - Some predefined **abstract attributes**: *Name*, *Id*, and *Text*
 - Some **traceability relationships** that include attributes such as: *Owner*, *Derived*, *Derived From*, *Satisfied By*, *Refined By*, *Traced To*, and *Verified By*
 - **Hierarchical relationships** between requirements with a containment relationship

Remark

Even though some relationships between the requirement and system model are now possible, the primary text statement is static, which is more likely to be incomplete, contain errors, and be inconsistent with the rest of the SysML-based system model.

¹ *OMG System Modeling Language* 2019; Friedenthal, Moore, and Steiner 2015

→ Classical SysML Requirements Modeling (2): Example

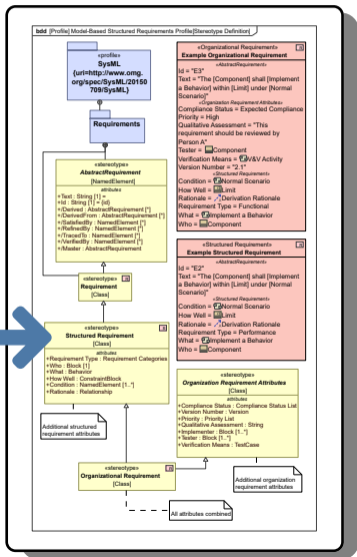


→ Model-Based Structured Requirement in SysML (1): Motivation

- To address the concerns regarding the use of textual structured requirements and classical SysML requirements modeling, we define here an approach that **combines the two together** to leverage the advantages of both¹
- We want a more rigorous approach for structuring requirements that also directly links to the SysML system model elements, similar to the existing traceability relationships
- With direct links to the system model, this approach to requirements is more heavily model-based and structured; hence, we term these **model-based structured requirements** (MBSRs)

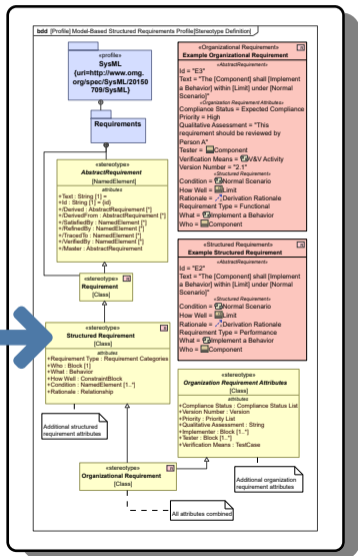
¹ Herber, Narsinghani, and Eftekhari-Shahroudi 2022; Narsinghani 2021

→ Model-Based Structured Requirement in SysML (2): Stereotype



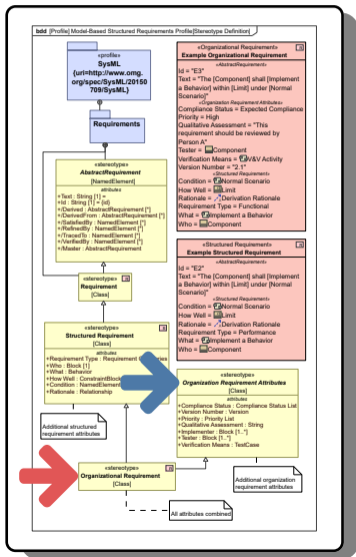
- «*Structured Requirement*» stereotype is defined in the figure on the left for the structured requirement template in Eq. (1)
- Each attribute is typed by a SysML model element kind with several powerful consequences:
 - We can enforce that only particular types of model content is valid for a particular attribute (e.g., the **[Who]** attribute must be a block)
 - Allows us to pick from an auto-generated list of available elements in most SysML tools (or see the element needs to be created)
 - Can specify how many elements can be included with a particular attribute using multiplicity (e.g., [1] or [1..*]).

→ Model-Based Structured Requirement in SysML (3): Application



- Now, an MBSR element can be created by:
 - Applying the stereotype to a requirement or creating a requirement with this stereotype applied
 - Specifying the SysML attributes necessitated by this stereotype
- A basic example, titled Example Structured Requirement, is shown on the left where we see model elements (indicated by icons) linked to the attributes of the MBSR

→ Enhancing MBSRs with Organizational Attributes



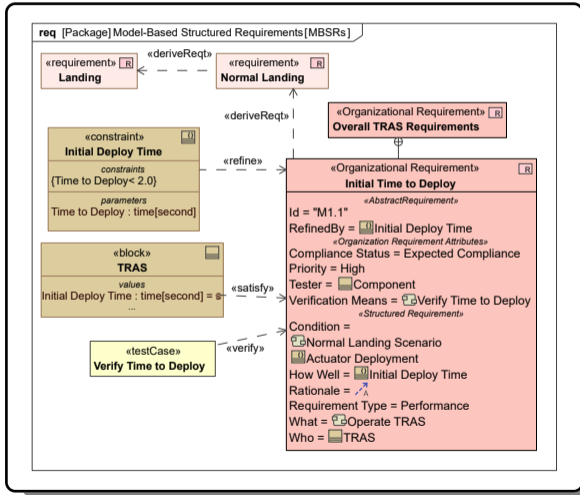
- Additionally, we seek to integrate organizational attributes into the fundamental definition of requirements that align with the specific rules and policies of the organization

Remark

Such additions might allow MBSRs in SysML to capture all information required for a high-quality, compliant requirement.

- Attributes in «*Organization Requirement Attributes*» include **Compliance Status, Version Number, Priority, Qualitative Assessment, Implementer, Tester, and Verification Means**
- Then, «*Organization Requirement*» combines «*Organization Requirement Attributes*» and «*Structured Requirement*»

→ Model-Based Structured Requirement in SysML Example




→ Model-Based Structured Requirement in SysML (4): Potential Benefits

- Connecting requirement attributes with model elements allows requirement information (both definition and traceability) to remain current and available
- Therefore, this can further reduce the errors and iterations while developing requirements, saving time and other resources
- Also simplify activities such as dynamic change impact assessment (e.g., if this block changed, what requirement definitions depended on it?)
- By formally breaking down the requirement statement into required attributes as model elements, completeness metrics can be automatically computed, and its gradual definition might be less risky
- Finally, there might be a deeper, more natural connection between the requirements team and other engineering groups through the single-source-of-truth SysML model

③

Examples

→ Example Overview

- We now present several more examples of requirements implemented as MBSRs (Slide 20), as well as pure textual structured (Slide 18) and classical SysML requirements (Slide 19) for comparison
- All of the examples are based on a notional thrust reverser actuation system (TRAS)
 - A necessary subsystem in most commercial aircraft to achieve and maintain safe ground stopping distance after a touchdown in adverse conditions such as wet/slippery runways by reversing fan bypass air flow¹
- The model for these examples and MBSR stereotype definition is publicly available on  GitHub²
 - Model created in No Magic's Cameo Systems Modeler 19.0 LTR SP4

¹ Maré 2018; Yetter 1995 ² <https://github.com/danielrherber/model-based-structured-requirements>

→ Textual Requirements for TRAS

ID	Name	Requirement Type	Text	Rationale
1.1	Initial Time to Deploy	Performance	TRAS shall reach 85% of the actuator stroke in less than 2 seconds under normal landing deploy condition.	Decelerate Aircraft Rapidly and Reliably
1.2	TRAS MTBF	Non-Functional	TRAS MTBF shall be greater than 15000 mission flight hours under normal landing condition.	Decelerate Aircraft Rapidly and Reliably
1.3	TRAS Average Power Consumption During Deploy Operation	Interface	During Thrust reverser deploy operation, from ECU/DCR opening to fully extended actuator position, TRAS average power consumption shall be lower than 35 kW.	Decelerate Aircraft Rapidly and Reliably
1.4	TRAS Fluid Interface	Interface	TRAS shall be able to withstand the hydraulic fluid temperature within a range of -70F to 280F on ground, under storage conditions.	Improve Efficiency, Safety and Sustainability
1.5	TRAS Weight Limit	Physical	System total mass shall be less than 320 pounds.	Differentiate with More Electric Aircraft
1.6	Jam During Reverser Deployment	Design Constraint	TRAS shall withstand the actuator lock jam without deformation when subjected to a compressive load of -5075 lbf.	Improve Efficiency, Safety and Sustainability
1.7	Interruption	Functional	TRAS shall be capable of changing the direction of reverser motion on command at any point in the actuation cycle under normal loading conditions.	Decelerate Aircraft Rapidly and Reliably

- Much of the necessary information is captured in the pure spreadsheet version
- However, the pieces of information are generally static, unlinked from any other representation of the system of interest

→ Classical SysML Requirements for TRAS

#	△ Id	Name	Text	Rationale	Derived From	Satisfied By	Refined By	Verified By
1	T1	T1 Overall TRAS Requirements	Overall System Requirement Specification					
2	T1.1	T1.1 Initial Time to Deploy	TRAS shall reach 85% of the actuator stroke in less than 2 seconds under normal landing deploy condition.	Decelerate Aircraft Rapidly and Reliably	3.1 Normal Landing	Initial Deploy Time : time[second]	Initial Deploy Time	Verify Time to Deploy
3	T1.2	T1.2 TRAS MTBF	TRAS MTBF shall be greater than 15000 mission flight hours under normal landing condition.	Decelerate Aircraft Rapidly and Reliably	3.6 TR Probability of Failure	TRAS MTBF : Mission Flight Time	TRAS MTBF	Verify TRAS MTBF
4	T1.3	T1.3 TRAS Average Power Consumption During Deploy Operation	During Thrust reverser deploy operation, from ECU/DCR opening to fully extended actuator position, TRAS average power consumption shall be lower than 35 kW.	Decelerate Aircraft Rapidly and Reliably	3.5 Energy Efficiency Gain	Average Power Consumption : power[kilowatt]	TRAS Average Power Consumption During Deploy Operation	Verify Average Power Consumption
5	T1.4	T1.4 TRAS Fluid Interface	TRAS shall be able to withstand the hydraulic fluid temperature within a range of -70F to 280F on ground, under storage conditions.	Improve Efficiency, Safety and Sustainability	3.1 Normal Landing	Fluid Temperature : thermodynamic temperature[kelvin] = 366.0 K	Hydraulic Fluid Temperature Range	Measure Hydraulic Fluid Temperature
6	T1.5	T1.5 TRAS Weight Limit	System total mass shall be less than 320 pounds.	Differentiate with More Electric Aircraft	3.4 TR Level Weight Constraint	Total Mass : mass[pound]	Total Mass	Verify Total Mass
7	T1.6	T1.6 Jam During Reverser Deployment	TRAS shall withstand the actuator lock jam without deformation when subjected to a compressive load of -5075 lbf.	Improve Efficiency, Safety and Sustainability	3.1 Normal Landing	TRAS	Overcome Jamming Loads	Verify Jamming Loads
8	T1.7	T1.7 Interruption	TRAS shall be capable of changing the direction of reverser motion on command at any point in the actuation cycle under normal loading conditions.	Decelerate Aircraft Rapidly and Reliably	3.1 Normal Landing	Control Motion	Deployment Direction	Verify Deployment Direction

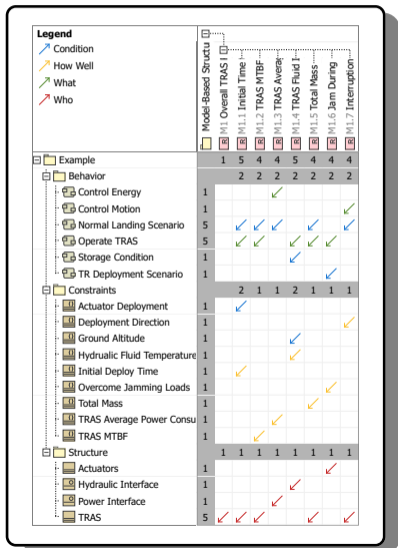
- Now, we have a lot more information connected to the system model
- However, we are still left to understand much of a specific requirement through its static text-based statement

→ Model-Based Structured Requirements (MBSRs) for TRAS

#	△ Id	Name	Requirement Type	Text	Who	What	How Well	Condition	Rationale	Satisfied By	Verification Means	Priority
1	M1	M1 Overall TRAS Requirements		Overall System Requirement Specification	TRAS							
2	M1.1	M1.1 Initial Time to Deploy	Performance	TRAS shall reach 85% of the actuator stroke in less than 2 seconds under normal landing deploy condition.	TRAS	Operate TRAS	Initial Deploy Time	Normal Landing Scenario Actuator Deployment	DeriveReq[Initial Time to Deploy -> Normal Landing]	Initial Deploy Time : time[second]	Verify Time to Deploy	High
3	M1.2	M1.2 TRAS MTBF	Non-Functional	TRAS MTBF shall be greater than 15000 mission flight hours under normal landing condition.	TRAS	Operate TRAS	TRAS MTBF	Normal Landing Scenario	DeriveReq[TRAS MTBF -> TR Probability of Failure]	TRAS MTBF : Mission Flight Time	Verify TRAS MTBF	Medium
4	M1.3	TRAS Average Power Consumption During Deploy Operation	Interface	During Thrust reverser deploy operation, from ECU/DCR opening to fully extended actuator position, TRAS average power consumption shall be lower than 35 kW.	Power Interface	Control Energy	TRAS Average Power Consumption During Deploy Operation	Normal Landing Scenario	DeriveReq[TRAS Average Power Consumption During Deploy Operation -> Energy Efficiency Gain]	Average Power Consumption : power[kilowatt]	Verify Average Power Consumption	Medium
5	M1.4	M1.4 TRAS Fluid Interface	Interface	TRAS shall be able to withstand the hydraulic fluid temperature within a range of -70F to 280F on ground, under storage conditions.	Hydraulic Interface	Operate TRAS	Hydraulic Fluid Temperature Range	Ground Altitude Storage Condition	DeriveReq[Normal Landing -> Landing]	Fluid Temperature : thermodynamic temperature[kelvin] = 366.0 K	Measure Hydraulic Fluid Temperature	Low
6	M1.5	M1.5 Total Mass	Physical	System total mass shall be less than 320 pounds.	TRAS	Operate TRAS	Total Mass	Normal Landing Scenario	DeriveReq[Total Mass -> TR Level Weight Constraint]	Total Mass : mass[pound]	Verify Total Mass	Low
7	M1.6	M1.6 Jam During Reverser Deployment	Design Constraint	TRAS shall withstand the actuator lock jam without deformation when subjected to a compressive load of -5075 lbf.	Actuators	Operate TRAS	Overcome Jamming Loads	TR Deployment Scenario	DeriveReq[Jam During Reverser Deployment -> Safety Against IAS]	TRAS	Verify Jamming Loads	Medium
8	M1.7	M1.7 Interruption	Functional	TRAS shall be capable of changing the direction of reverser motion on command at any point in the actuation cycle under normal loading conditions.	TRAS	Control Motion	Deployment Direction	Normal Landing Scenario	DeriveReq[Interruption -> Normal Landing]	Control Motion	Verify Deployment Direction	Low
9	M2	M2 [Incomplete Requirement]										

- We have more specific information about each requirement, and this information comes in the form of links to various model elements
 - Do you want to know what Normal Landing Scenario is? Go to the element that defines it
 - If the Normal Landing Scenario definition changed, the requirement has a direct link to those changes

→ Model-driven Dependency Matrix



- More so than the classical SysML approach, we have shared model elements across the MBSRs
- This is automatically visualized and counted in a dependency matrix that includes the key «Structured Requirement» attributes
- TRAS is shared among multiple requirements as well as Operate TRAS and Normal Landing Scenario
- Requirements might also be (partially) built up from views like this dependency matrix (i.e., link model elements to the MBSR attributes)

→ Metrics Measuring MBSR Completeness

#	△ Name	Metric Suite	📁 Scope	📊 Complete Requirement Percentage	📊 Total Requirements	📊 Includes Who	📊 Includes Condition	📊 Includes How Well	📊 Includes What
1	📄 Version 1	📄 MBSR Completeness Metrics	📁 Example	37.5	8	5	5	6	6
2	📄 Version 2	📄 MBSR Completeness Metrics	📁 Example	77.7778	9	8	7	7	7

- We can automatically assess requirement completeness with respect to the MBSR specification
- For example, an MBSR is considered complete when it has nonempty [**Who**], [**What**], [**How Well**], and [**Condition**] attributes
- These metrics are automatically computed in the model using customized metric suites and scripts (see the next slide)
- However, TRAS might be considered complete as all child requirements are complete, and more appropriate handling of this scenario is left as future work

→ MBSR Nonempty Attribute Counting Script

```
import com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
int boolToInt(Boolean b) { // convert boolean to integer
    return b.compareTo(false);
}
stereotype = "Structured Requirement"; // stereotype to use
attribute = "Who"; // attribute to check for
n = 0; // initialize value
for(element in element_list){ // go through each element in list
    id = StereotypesHelper.getStereotypePropertyValueAsString(element, stereotype, attribute);
    n += boolToInt(!id.isEmpty()); // add 1 if not empty
}
return n;
```

- Above is the Groovy script with Cameo Systems Modeler used to count how many MBSRs have a nonempty **[Who]** attribute where `element_list` contains all MBSRs in a particular context
- This illustrates that MBSRs in SysML provides the possibility for deeper programmatic interaction with requirements
- Full details available in the model¹

¹ <https://github.com/danielrherber/model-based-structured-requirements>

④

Conclusion

→ Summary

- When requirements are written in the classical text format, significant resources (including many brains) are required to develop and manage them, which can lead to identifying problems late in the development cycle
- Overall, the proposed MBSR approach is more aligned with the model-centric philosophy of system development through its more broad use of elements in a system model
- It adds systems thinking rigor when developing the model (and requirements) that support the wide range of SE activities
- The MBSR *restricts* us to create and define the right elements and relationships (or readily see that they are missing)
- It also directly connects pieces that help define a requirement to the functional/physical architecture elements and system verification/validation artifacts with more specific and relevant relationships¹


¹ Wheatcraft et al. 2022

→ Future Work

- Customized **MBSR validation rules** will help eliminate errors and improve the quality in the early stages of requirements elicitation in a more automated manner
- Utilizing APIs and scripting available in the tools, we might **automatically generate the requirement text statement** only using the structured attributes¹
 - After all, the structured requirement template is simply filling in the placeholders with the natural language names for the attributes in Eq. (1).
- More refinements to the attributes (both in naming, typing, and completeness) should be investigated to ensure that they holistically capture the concerns in requirements development
- Further investigations into requirement and model metrics (see Slide 23) that help define requirement completeness utilizing the structured attributes
- Development of **specific profiles for different types of structured requirements** and organizational needs
- Additional examples push forward and refine this concept

¹ R. S. Carson 2015

→ General Parting Questions

- ❓ What do you like and not like about the proposed MBSR approach in SysML?
What might you change?
- ❓ Should requirements be done within a SysML modeling tool with the MBSR concept (or what would still be preventing the proposed transition)?
- ❓ How do we effectively merge existing SysML requirements modeling capabilities with MBSRs (i.e., is there overlap)?
- ❓ What might a better completeness metric/formula look like for MBSRs (or requirements in general)?
- ❓ Do you want to contribute to this effort?
 - Please consider engaging with the open-sourced model on  GitHub at <https://github.com/danielrherber/model-based-structured-requirements>
 - Comments, proposed refinements, additional examples, etc., are all encouraged

→ Acknowledgements

- Thanks to my collaborators at CSU on this work:
 - Jayesh B. Narsinghani
 - Kamran Eftekhari-Shahroudi
- This research was supported by the Woodward-CSU Master Research and Development Agreement



→ References

- R. Carson (2021). *Developing Complete and Validated Requirements*. INCOSE Seattle-Metropolitan Chapter Monthly Meeting. DOI: 10.13140/RG.2.2.28526.74561
- R. S. Carson (2015). "Implementing Structured Requirements to Improve Requirements Quality". *INCOSE International Symposium* 25.1. DOI: 10.1002/j.2334-5837.2015.00048.x
- S. Friedenthal, A. Moore, and R. Steiner (2015). *A Practical Guide to SysML*. 3rd. Elsevier. DOI: 10.1016/c2013-0-14457-1
- D. R. Herber, J. B. Narsinghani, and K. Eftekhari-Shahroudi (2022). "Model-based structured requirements in SysML". *2022 IEEE International Systems Conference*. DOI: 10.1109/syscon53536.2022.9773813
- *IEEE Guide for Developing System Requirements Specifications* (1998). IEEE. DOI: 10.1109/ieeestd.1998.88826
- *ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering* (2018). IEEE. DOI: 10.1109/ieeestd.2018.8559686
- J.-C. Maré (2018). *Aerospace Actuators 3: European Commercial Aircraft and Tiltrotor Aircraft*. John Wiley & Sons, Inc. DOI: 10.1002/9781119505433
- *Model-based structured requirements* (n.d.). URL: <https://github.com/danielrherber/model-based-structured-requirements>
- J. B. Narsinghani (2021). "Towards a model-based implementation in technology/platform life cycle development processes applied to a thrust reverser actuation system (TRAS) concept". MS thesis. Colorado State University

→ References (continued)

- *OMG System Modeling Language* (2019). Object Management Group
- K. Pohl (2010). *Requirements Engineering*. Springer
- H. de Ribaupierre et al. (2021). *Automatic extraction of requirements expressed in industrial standards : a way towards machine readable standards ?* arXiv. DOI: 10.48550/arXiv.2112.13091
- M. Ryan et al. (2019). *Guide for Writing Requirements*. Tech. Prod. INCOSE-TP-2010-006-03. INCOSE Requirements Working Group
- L. Wheatcraft et al. (2022). *Needs, Requirements, Verification, Validation Lifecycle Manual*. Tech. Prod. INCOSE-TP-2021.002-01. INCOSE Requirements Working Group
- J. A. Yetter (1995). *Why Do Airlines Want and Use Thrust Reversers? A Compilation of Airline Industry Responses to a Survey Regarding the Use of Thrust Reversers on Commercial Transport Airplanes*. Tech. rep. NASA-TM-109158. NASA Langley Research Center

Thank you.

Model-Based Structured Requirements in SysML

Daniel R. Herber

<https://github.com/danielrherber/model-based-structured-requirements>