Model-Based Structured Requirements in SysML

Daniel R. Herber
Systems Engineering
Colorado State University
Fort Collins, CO, USA
daniel.herber@colostate.edu

Jayesh B. Narsinghani
Systems Engineering
Colorado State University
Fort Collins, CO, USA
jayesh.narsinghani@colostate.edu

Kamran Eftekhari-Shahroudi
Woodward, Inc.
Fort Collins, CO, USA
Systems Engineering
Colorado State University
Fort Collins, CO, USA
kamran.eftekhari_shahroudi@colostate.edu

***Abstract*—Architecture-centric practices are gaining widespread acceptance in systems engineering. 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. Central to the rules that govern a system are the requirements that are placed on it, often by various stakeholders. These requirements help guide the system development process of a complex entity. In this paper, we discuss an approach for extending the idea of structured requirements (requirements defined through an orderly structure with specific pieces of content that must be filled in) to SysML through customized stereotypes that help enforce the requirement structure through model-based attributes. This approach helps move requirements modeling and management further into the model-based paradigm from the classical textual definitions. In addition, requirements often are customized by the organization defining them through additional attributes. These additional attributes are added to the model-based structured requirement (MBSR) to create a well-defined organizational requirement stereotype. Several examples of the MBSRs are presented using a notional thrust reverser actuation system (TRAS). Several points are made on how this approach can help support more rigorous requirements modeling, analysis, management, and communication throughout the system development process. Future work will involve automatic generation of the textual requirement statements from the attributes, customized validation rules, and customized classifiers for the various attributes.**

***Index Terms*—model-based systems engineering; requirements development; systems engineering; digital engineering**

I. Introduction

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 system model does not inherently provide value to the SE effort; rather, benefits are realized through its support of various SE activities, from understanding needs to solution test and deploy. To better define how to both synergistically create a system model and support the system development process, various model-based systems engineering (MBSE) practices are being explored [1]–[3]. Fortunately, there is codified and disciplined support for MBSE in the form of methodologies, formal modeling languages, and tools [1]. Overall, there is increasing formal and informal evidence that MBSE can improve efficiency, rigor, and quality during system development [3]–[5].

The Systems Modeling Language (SysML) is a widely-supported graphical modeling language that prescribes a set of object-oriented elements and allowable relationships between the different elements supporting the creation of an integrated system model that is interacted with through diagrams and other mechanisms [6], [7]. Software tools are the key enablers of this form of system modeling and MBSE. One popular tool (and the one used in this work) is Cameo Systems Modeler, which implements SysML along with other features to support MBSE activities [8].

Central to the rules governing a system are the requirements placed on it, often by various stakeholders, and these requirements help guide the system development process of a complex entity. A requirement is a statement that "translates or expresses a need and its associated constraints and conditions", while requirements engineering is concerned with all aspects of discovering, validating, and managing requirements [9]. 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 in the system model.

There are a variety of approaches for developing and managing requirements [9]–[13]. However, there is still a common problem of tremendous effort being spent understanding the requirements and multiple iterations because these guidelines and rules must be adhered to manually by engineers. Furthermore, many requirement approaches do not fully integrate with the system model (i.e., there are limited relationships between the textual requirements and system model elements). These issues lead to increased costs and risks during complex systems development due to the poor quality of requirements defined during the early stages. To address these concerns, an approach for supporting rigorous model-based requirements is proposed using the existing concepts of structured requirements and SysML, termed model-based structured requirement (MBSR).

The contributions in this papers are summarized as 1) an MBSR stereotype in SysML is demonstrated, 2) the addition of organizational attributes to requirements is shown, 3) a metric suite is created to evaluate MBSR completeness, 4) and

This research was supported by the Woodward-CSU Master Research and Development Agreement.

several examples are made available comparing the MBSR approach to existing approaches. The rest of the paper is organized as follows: Sec. II describes the proposed approach, including stereotype and metric suite definitions; Sec. III provides several examples of requirements using two existing approaches as well as the proposed MBSR approach; and Sec. IV discusses the benefits and future work items. Finally, Sec. V presents the conclusions.

II. Model-based Structured Requirements

It has been established that a well-defined requirement shall possess the following general characteristics [9], [11]:

- *Necessary*—Has a clear source demanding its inclusion.
- *Appropriate*—The specific intent and amount of detail of the requirement is appropriate to the level of the entity to which it refers.
- *Unambiguous*—Defined simply and is easy to understand so that it can be interpreted in only one way.
- *Complete*—Has no missing information.
- *Singular*—Articulates a single requirement.
- *Feasible*—Reasonable enough to be implemented within the available resource constraints (e.g., cost, schedule, and technical limitations).
- *Verifiable*—Defined with a verification means and a clear acceptance criterion.
- *Correct*—Accurate representation of the entity without errors and confirmed by stakeholders.
- *Conforming*—Conformance to an approved standard template and style for writing requirements, when applicable.

Writing a set of requirements that adheres to these characteristics is challenging, particularly as system development and understanding evolve and requirement volume balloons (often into the thousands of interrelated shall statements). In the remainder of this section, different techniques are described to aid in developing requirements that more rigorously and naturally adhere to the principles above when compared to both traditional and modern advancements.

A. Structured Requirement

We start by describing the existing concept of a structured requirement [14]. A structured requirement defines an orderly structure with specified attribute placeholders to help 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)

where each of four bold attributes are described as follows:

- [**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 for a requirement.

As with more general requirement statements, the *shall* clause indicates that a requirement statement is mandatory and must be followed. For example, the following is a structured requirement written in natural language:

> The [**actuation system**] *shall* [**prevent inadvertent stowing**] [**with at least three levels of safety**] under [**normal deploy operation**]. (R1)

Similarly, the orderly sequence of attributes for a structured requirement can be aligned with other approaches or standards. For example, following some minor changes to the attributes, the 29148-2018 standard can be represented as [9]:

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

with an example structured requirement in this format being:

> [**When aircraft on ground status signal is received**], [**the pilot**] [**shall engage**] [**the thrust lever to reverse position**] [**initiating the reverser cycle within 3 seconds**]. (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 for the specific needs of the different requirement types [10].

This approach to requirements modeling helps improve the overall quality of the requirement statements by adding needed structure while defining the requirement. This includes helping meet the general characteristics listed above (in particular, appropriate, unambiguous, complete, singular, correct, and conforming). Overall, it is generally easier to understand and manage requirements when adhering to the structure-based approach. However, as currently presented, this is simply a refinement to the classical textual requirements approach and has only an imprecise relationship to the system model.

B. Classical SysML Requirements Modeling

As with many aspects of system modeling, SysML supports the inclusion of requirements [6], [7]. A classical SysML-based requirement is developed by defining some predefined abstract attributes such as:

- *Name* typed by a general name of a requirement.
- *Id* typed by an identification tag/serial number.
- *Text* typed by a text statement in natural language.

as well as some traceability relationships that include attributes such as:

- *Owner* specified with the location in the model.
- *Derived* specified with a derived child requirement.
- *Derived From* specified with a parent requirement that provides the necessary information to write the requirement.

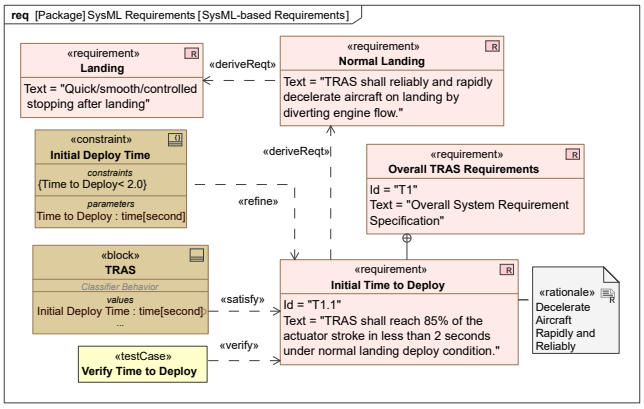

Fig. 1: Classical SysML requirement diagram utilizing text attribute for the *shall* statement and several relationships.

- *Satisfied By* specified with a model entity that provides the required capability or characteristic.
- *Refined By* specified with a model entity that provides additional information to improve the requirement.
- *Traced To* specified by a model entity that describes a stakeholder input or source artifact.
- *Verified By* specified with a model entity that confirms that the requirement is justified.

Further, hierarchical requirements can be nested into constituent requirements with a containment relationship. Note that these constituent requirements may provide the necessary information to specify one or more lower-level requirement(s) represented by a «*derive*» relationship between requirements.

To illustrate the classical SysML requirement modeling, Fig. 1 is a requirement diagram which is a view into the system model for several requirements and their traceability relationships. For example, Overall TRAS Requirements is decomposed further to contain requirements Initial Time to Deploy and TRAS Average Power Consumption During Deploy Operation. To capture the traceability of the requirement Initial Time to Deploy, SysML allows the creation of specific relations. For instance, a «*deriveReqt*» relationship can be created between the requirement and a parent requirement, Normal Landing. Further, a «*deriveReqt*» relation can be created between the parent requirement and a stakeholder need, Landing. Another example is «*rationale*» typed by a textual comment that is linked as a rationale to one or more requirements.

Some relationships are meant to indicate more specific connection. A system performance value property Initial Deploy Time that satisfies the requirement can be linked with a «*satisfy*» relationship. A performance limitation specified by a constraint Initial Deploy Time that refines the requirement (meaning that it provides the necessary information to improve the requirement statement) can be linked with a «*refine*» relationship. A verification activity specified by the test case Verify Time to Deploy that verifies the requirement can be related with a «*verify*» relationship.

It is important to note that this approach involves the creation of requirement artifacts and relationships with model

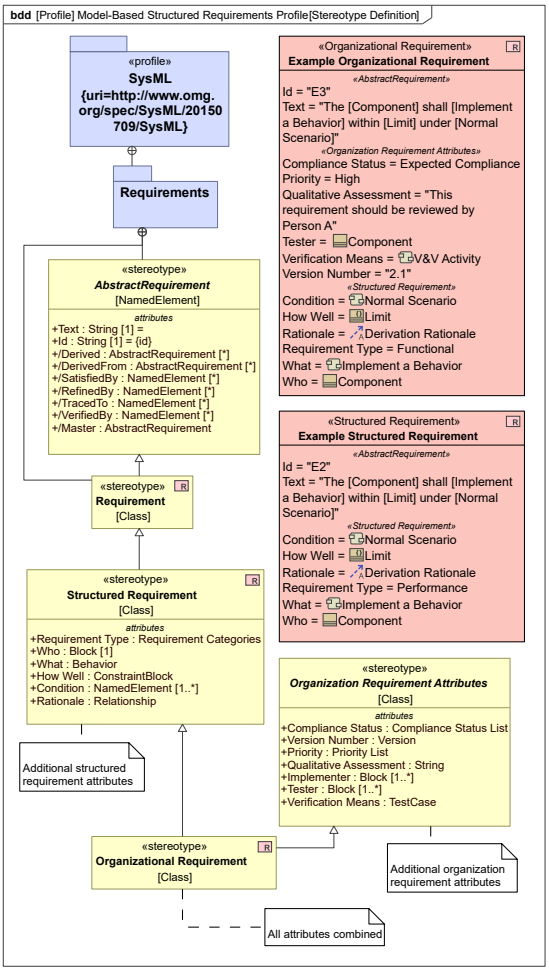

Fig. 2: Stereotype definitions for model-based structured and organizational requirements.

elements (blocks, constraints, test cases, requirements, etc.). Moreover, 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. However, the relationships between model elements (e.g., ones with «*satisfy*» and «*verify*») are dynamically linked to the overall system model, so remain consistent if a model element changes (or there is an issue that can be automatically identified). Therefore, the classical SysML approach for including requirements has several excellent capabilities for defining requirements in a model-based context but is missing some element features that structured model-based requirements can provide.

C. Model-Based Structured Requirement in SysML

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 both [15]. As discussed in Sec. II-A, 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 described in Sec. II-B. 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).

To accomplish this goal, we leverage several features of SysML. Referring to Fig. 2, we import «*Requirement*» stereotype from the SysML profile (which already has the attributes described previously). We then define a new «*Structured Requirement*» stereotype inherits all the properties of «*Requirement*» using the generalization relationship. Now with common properties, we augment the base SysML «*Requirement*» with the attributes from the structured requirement format in Eq. (1) as well as Requirement Type and Rationale.

Each of these new attributes is typed by a SysML model element kind, a particular feature that has several powerful consequences. First, we can enforce that only particular types of model content is valid for a particular attribute. For example, the [**Who**] attribute must be a block. This also allows us to pick from an automatically-generated list of available blocks in most SysML tools (and if the model element that is desired does not appear, then it needs to be created in the system model). Second, we can specify how many elements can be included with a particular attribute using multiplicity (e.g., [1] or [1..*]).

The specific SysML classifiers that are enforced in Fig. 2 for each attribute are summarized as follows[1]:

- [**Who**] refers to a subject model entity so is specified by a SysML classifier block that can be typed by system, subsystem, interface, constraint, and more.
- [**What**] refers to a required capability so is specified by a UML classifier behavior that can be typed by a use case, activity, state machine, interaction, opaque behavior, and more.
- [**How Well**] refers to a limitations or constraints so is specified by a SysML classifier constraint block.
- [**Condition**] refers to a mode of operation or a scenario so is specified by a SysML named element for flexibility.
- **Requirement Type** refers to a category of requirement so is specified by a type from an enumerated list of string types including performance, functional, non-functional, interface, and physical requirement literals.
- **Rationale** refers to the underlying reason why a requirement exists so is specified by a dependency, a type of relationship, such as «*DeriveReqt*».

Now, with «*Structured Requirement*» defined, an MBSR element can be created by applying the stereotype to a requirement and specifying the SysML attributes necessitated by this stereotype. A basic example, titled Example Structured Requirement, is shown in Fig. 2 where we see model elements (indicated by icons) linked to the attributes of the MBSR. More detailed examples are provided in Sec. III.

[1]In the current implementation, only existing SysML classifiers are utilized for this typing activity, and these can be customized to fit different definitions of the attributes. Future work is expected to support customized classifiers based on combining different types and limiting types based on stereotypes.

This process is a disciplined approach for capturing requirements as well as specific aspects of a system model during requirements elicitation. This technique of connecting requirement attributes with model elements allows requirement information to remain current and available and simplifies activities such as dynamic change impact assessment. Therefore, this can further reduce the errors and iterations while developing requirements, saving time and other resources. It is aligned with the general philosophy put forth by those that advocate for structured requirements in general by enforcing the structure in a requirement specification [16].

D. Enhancing with Organizational Attributes

We can go further with the notion of model-based requirements to better meet the practical needs of organizations and other groups developing their requirements in a model-based context. Extending the capability of MBSRs, we seek to integrate organizational attributes into the fundamental definition of requirements that align with the specific rules and policies of the organization (recall the conforming characteristic of requirements for motivation). Such additions are generally needed to consider the requirements to be valid/acceptable in the context of the organization's SE policies.

To support these additions, a list of additional attributes in «*Organization Requirement Attributes*» can be captured as an independent abstract stereotype, as shown in Fig. 2. Some specific included organization specific attributes, defined with classifier types and optional multiplicities, are:

- **Compliance Status** indicates how well a requirement complies to the overall rules that must me met following some standards, and is typed by an enumerated list with literals expected compliance, compliant, noncompliant, or to be rated.
- **Version Number** indicates a text or serial number typed by a specialized version value type.
- **Priority** indicates the importance of a requirement on the basis of stakeholder interest, typed by an enumeration list.
- **Qualitative** Assessment refers to description of a requirement's quality specified by a string.
- **Implementer** refers to the stakeholder(s) responsible for implementing a requirement and is typed by a SysML block.
- **Tester** refers to a stakeholder responsible for verifying whether a capability achieved from the requirements is as per the specification.
- **Verification Means** refers to an action plan for verifying the requirement, specified by a SysML test case.

These attributes can be customized to fit a particular organization's needs. To combine these attributes with the MBSR, «*Organizational Requirement*» is defined that inherits the properties from both «*Organization Requirement Attributes*» and «*Structured Requirement*», as shown in Fig. 2. A basic example, titled Example Organizational Requirement, is shown in Fig. 2.

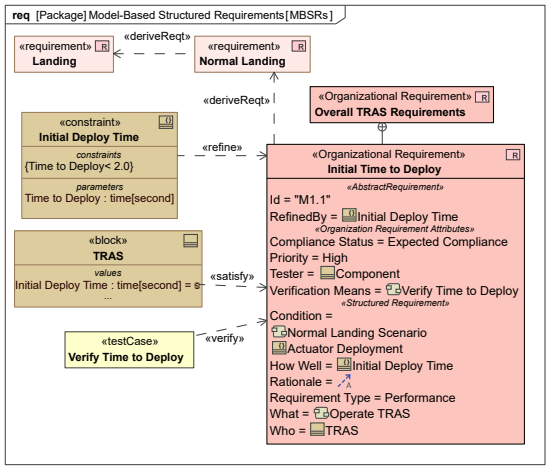

Fig. 3: Model-centric SysML requirements utilizing the structured organizational requirement stereotype defined in Fig. 2 (and is an alternative representation of the requirements first presented in Figs. 1 and 4b).

E. Completeness Metrics

By further breaking down a requirement statement into required attributes, both engineers and managers can more readily quantify and visualize several indicators of requirement quality and identify gaps/missing information. Different requirements will be incomplete throughout the development, but the eventual goal is complete requirements definition in the model and corresponding system design. If various metrics that reflect the model's current state (and system development effort) are tracked, then the evolution of quality and completeness can be rigorously evaluated. Based on the definition of an MBSR in Sec. II-C, several metrics can be readily defined and computed:

- Percentage of MBSRs that are complete where completeness is defined as a requirement having nonempty [**Who**], [**What**], [**How Well**], and [**Condition**] attributes
- Total number of MBSRs.
- Nonempty attribute MBSRs counts for each of the four structured attributes.

The specific Groovy script used to compute the nonempty [**Who**] counts is shown in App. A. The information captured through MBSRs permits additional metric calculations, and the metrics above might be combined with other traditional SysML requirement relationships (e.g., «*satisfy*») for a more comprehensive completeness condition.

III. Examples

In this section, we present several more examples of requirements implemented as MBSRs, as well as pure textual structured and classical SysML requirements. All three are summarized in Fig. 4. The model for these examples and MBSR stereotype definition is available at [17]. All of the examples are based on a notional thrust reverser actuation system (TRAS), which is 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 flow [18]. TRAS also reduces brake wear and provides control under emergency scenarios [19].

A. Textual Requirements

As we will see, much of the necessary information is captured in the pure spreadsheet version in Fig. 4a with columns for attributes like ID, Name, Requirement Type, Text, and Rationale. However, the pieces of information are generally static, unlinked from any other representation of the system of interest, so requirements cannot be readily developed in a manner consistent with the current understanding of the system (often as a system model in model-centric approaches). Furthermore, traceability (both first and higher-order), such as source/parent, is extremely hard to manage and analyze with this spreadsheet.

B. Classical SysML Requirements

Moving to the classical SysML requirements described in Sec. II-B, we have a lot more information connected to the system model, as indicated in Fig. 4b. The foundational attributes of ID, Name, Text, and Rationale are still present but now we have some direct links to the model elements (as well as the visualization power of SysML diagrams, cf. Fig. 1). In Initial Time to Deploy, we readily see that it is derived from the requirement Normal Landing, satisfied by value property Initial Deploy Time, refined by the constraint block Initial Deploy Time, and verified by activity Verify Time to Deploy.

Each of these elements can have a whole set of rich relationships and complex specifications, but the relationship in the requirements model is always up to date and linked. We also start to see that some model elements are reused in these relationships, such as some of the rationale statements, leveraging the single definition of model elements. However, we are still left to understand much of a specific requirement through its static text-based statement.

C. Model-Based Structured Requirements

We now implement the same seven requirements from the previous two sections as MBSRs (with some organizational attributes as well). These are summarized in Fig. 4c, and Initial Time to Deploy is visualized in Fig. 3.

From either artifact, we have more specific information about each requirement, and this information comes in the form of links to various model elements. For example, consider Initial Time to Deploy. The [**who**] attribute is the TRAS block, the [**what**] condition is the activity Operate TRAS (a primary system behavior), the [**how well**] condition is a constraint on the Initial Deploy Time (which must be less than 2 s), and the [**condition**] under consideration is the Normal Landing Scenario and Actuator Deployment. Now, if the Normal Landing Scenario definition changed, the requirement has a direct link to those changes.

The other relationships from the classical SysML should certainly be included. For example, «*deriveReqt*» is used to

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

(a) Table summarizing the structured requirements captured in a spreadsheet.

#	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

(b) Table summarizing the requirements captured in a classical SysML context (another view of the model elements in Fig. 1).

#	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	DeriveReqt[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	DeriveReqt[TRAS MTBF -> TR Probability of Failure]	TRAS MTBF : Mission Flight Time	Verify TRAS MTBF	Medium
4	M1.3	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	DeriveReqt[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	DeriveReqt[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	DeriveReqt[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	DeriveReqt[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	DeriveReqt[Interruption -> Normal Landing]	Control Motion	Verify Deployment Direction	Low
9	M2	M2 [Incomplete Requirement]										

(c) Table summarizing the requirements captured as structured organizational requirements (another view of the model elements in Fig. 3).

Fig. 4: Several ways of implementing structured requirements.

understand the source of requirement as shown in Fig. 3. We have also explicitly defined the verification means through a system element that models the organization's verification and validation activity for this particular attribute (both digital and physical). Additionally, we see that Priority as been filled in.

While most of the summarizing table in Fig. 4c is complete (i.e., no missing cells), some requirements are not such as the dummy [Incomplete Requirement]. It is quite easy to pin point requirements that are missing certain attributes. In fact, with desired multiplicities being defined for the attributes, tool validation checks will automatically warn when there are too few tag values in an incomplete requirement. Furthermore, because the definitions of the structured requirement attributes are based on model elements, it is much easier to determine if the desired model element is already in the model or not[2].

More so than the classical SysML approach, we have shared model elements across the MBSRs. The block TRAS is [**Who**] is shared across several requirements. This is automatically visualized and counted in a dependency matrix that includes some of the key «*Structured Requirement*» attributes. Such a matrix is shown in Fig. 5 where we readily visualize that TRAS is shared among multiple requirements as well as Operate TRAS and Normal Landing Scenario, while other elements such as Actuators and Control Energy are only associated with a single requirement (in this view).

[2]This is still subject to the ability for a human to find the desired element in a list. If a duplicate element is created, it is relatively straightforward to merge them.

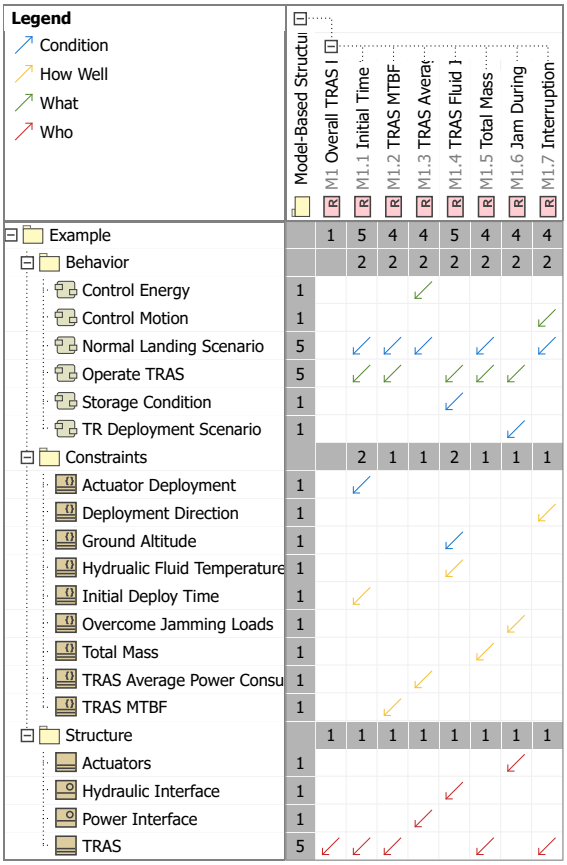

Fig. 5: Model-driven dependency matrix between the requirements and model elements based on structured attributes.

D. Metrics with Structured Requirements

With the metrics defined in Sec. II-E, we can automatically assess requirement completeness with respect to the MBSR specification. Observing Fig. 4c, we can readily see that many of the requirements are complete except Overall TRAS Requirements and a dummy [Incomplete Requirement]. Therefore, the completeness percentage is 78% (7/9). This metric is automatically computed for us in Fig. 6. However, TRAS might be considered complete as all child requirements are complete, and more appropriate handling of this scenario is left as future work.

IV. Discussion and Future Work

Overall, the MBSR approach is more aligned with the model-centric philosophy of system development through its more broad use of system elements. It adds systems thinking rigor when developing the model (and requirements) that support the wide range of SE activities. 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 relationships [20].

As an organization, we could follow and mandate conformance to various standards and guidelines for requirements, such as [9]. However, this does not generally ensure adherence because to adhere properly, many elements of the system structure, behavior, verification, etc., need to be rigorously defined and available. This is very hard to accomplish with the classical text-centric requirements approach, where requirement text is considered the center and highest artifact to create. The MBSR *restricts* us to create and define the right elements and relationships (or readily see that they are missing). 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. Furthermore, customer needs and goals often change throughout development, requiring major modifications to the requirements. Having such a structured approach, based on the system model that is also being developed and reviewed, has the potential to help produce requirements that are continuously well-defined. For example, instead of using text to capture the rationale behind a particular requirement (which itself can give rise to ambiguity), an MBSR approach simply selects the higher-level requirement, operational scenario, or stakeholder need/concern element that forms the rational or reason for being.

Additionally, while traditional requirement approaches can capture information, they are often lacking in their ability to adapt when changes occur as previously discussed. Lots of time is often spent reviewing and updating all the requirements when changes are introduced; this laborious activity is greatly automated when the right model-based relationships exist between elements. Basic attributes such as a element names are kept in sync as well as the digital artifacts (e.g., diagrams and tables) that use them. Furthermore, the impact of a change can be assessed more readily reviewed; hence the analysis is more automated and rigorous. Most tools support some form of suspect links, which indicate if something should be reviewed for validity because a related element is modified, and this is only possible through leveraging modeled relationships.

Furthermore, quality metrics on the requirements can be readily computed with MBSRs, such as current percentage that are complete, individual attribute complete percentages, rankings of elements with the most relationships to the requirements, etc. (with some examples highlighted in Sec. III). Because the requirements are more deeply integrated into the system model, the automatic generation of requirement artifacts is more readily available for different reviews.

Several important enhancements and open questions can be explored for MBSRs. First, there is a potential source of error in the current implementation: the textual statement of the requirement is still manually created and updated. Utilizing APIs and scripting available in the tools, it should be possible to automatically generate the text statement in Fig. 4c only using the structured attributes. After all, the structured requirement template is simply filling in the placeholders with the natural language names for the attributes in Eq. (1).

Supporting the mission of automating requirements analysis for completeness, correctness, and consistency, there is an opportunity to develop customized validation rules. Many such rules are often already built into tools, but with the specified

#	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

Fig. 6: Metric table for two versions of the model measuring MBSR completeness and attribute counts over time.

MBSR structure, we could readily assess the severity of errors. For example, an already included validation rule is based on the typing of the attributes. An error will be shown if a behavior is included in the [**Who**] attribute because it should be a block. Thus, including customized validation rules will help eliminate errors and improve the quality in the early stages of requirements elicitation in a more automated manner.

Additionally, 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. This includes customized classifiers based on combining different types and limiting types based on stereotypes as mentioned in Sec. II-C.

V. Conclusion

In this paper, an approach for supporting rigorous model-based requirements was proposed using the existing techniques of structured requirements and SysML. Termed model-based structured requirements (MBSRs), these new requirement types can be readily included in a SysML model similar to existing SysML-based requirements but with the added attributes for structuring the requirements specification. This approach included defining who, what, how well, and condition attributes to construct a structured requirement statement in natural language.Additionally, the expansion with organization attributes was demonstrated, which included adding attributes towards compliance with an organization's policies and rules. MBSRs were demonstrated with a notional TRAS with several requirements related to deploying time, reliability, average power, among other functions.

One of the core tenets of MBSE is bringing together viewpoints to create a shared understanding of the system. This point necessitates direct collaboration among multidisciplinary teams, not siloed activities performed by requirement, system, and domain-specific engineers. Overall, MBSRs can be seen as bringing requirements fully into the model-based paradigm allowing these groups to collaborate and communicate in a consistent, rigorous manner.

Acknowledgment

The authors would like to thank Woodward employees that provided various feedback. The opinions expressed herein are those of the authors and do not necessarily represent the views of Woodward, Inc. or any other individual.

References

[1] J. M. Borky and T. H. Bradley, *Effective Model-Based Systems Engineering*. Springer, 2019, doi: 10.1007/978-3-319-95669-5

[2] J. Navas, S. Bonnet, J.-L. Voirin, and G. Journaux, "Models as enablers of agility in complex systems engineering," *INCOSE International Symposium*, vol. 30, no. 1, pp. 339–355, Jul. 2020, doi: 10.1002/j.2334-5837.2020.00726.x

[3] T. Huldt and I. Stenius, "State-of-practice survey of model-based systems engineering," *Syst. Eng.*, vol. 22, no. 2, pp. 134–145, Sep. 2018, doi: 10.1002/sys.21466

[4] E. R. Carroll and R. J. Malins, "Systematic literature review: how is model-based systems engineering justified?" Sandia National Laboratories, Tech. Rep. SAND2016-2607, Mar. 2016, doi: 10.2172/1561164

[5] A. Madni and S. Purohit, "Economic analysis of model-based systems engineering," *Systems*, vol. 7, no. 1, Feb. 2019, doi: 10.3390/systems7010012

[6] *OMG System Modeling Language*, Object Management Group Std. 1.6, Dec. 2019.

[7] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML*, 3rd ed. Elsevier, 2015, doi: 10.1016/C2013-0-14457-1

[8] Dassault Systèmes, "Cameo systems modeler," 19.0 SP4.

[9] *ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering*, IEEE Std. 29 148-2018, Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686

[10] K. Pohl, *Requirements Engineering*. Springer, 2010.

[11] M. Ryan, L. Wheatcraft, R. Zinni, J. Dick, and K. Baksa, "Guide for writing requirements," INCOSE Requirements Working Group, Tech. Prod. INCOSE-TP-2010-006-03, Jul. 2019.

[12] R. Carson. (2021, Jun.) Developing complete and validated requirements. INCOSE Seattle-Metropolitan Chapter Monthly Meeting. doi: 10.13140/RG.2.2.28526.74561

[13] *IEEE Guide for Developing System Requirements Specifications*, IEEE Std. 1233-1998, Dec. 1998, doi: 10.1109/IEEESTD.1998.88826

[14] R. S. Carson, "Implementing structured requirements to improve requirements quality," *INCOSE International Symposium*, vol. 25, no. 1, pp. 54–67, Oct. 2015, doi: 10.1002/j.2334-5837.2015.00048.x

[15] J. B. Narsinghani, "Towards a model-based implementation in technology/platform life cycle development processes applied to a thrust reverser actuation system (TRAS) concept," Master's thesis, Colorado State University, Fort Collins, CO, USA, 2021.

[16] R. S. Carson, E. Aslaksen, G. Caple, P. Davies, R. Gonzales, R. Kohl, and A.-E.-K. Sahraoui, "5.1.3 requirements completeness," *INCOSE International Symposium*, vol. 14, no. 1, pp. 930–944, Jun. 2004, doi: 10.1002/j.2334-5837.2004.tb00546.x

[17] Model-based structured requirements. [Online]. Available: https://github.com/danielrherber/model-based-structured-requirements

[18] J.-C. Maré, *Aerospace Actuators 3: European Commercial Aircraft and Tiltrotor Aircraft*. John Wiley & Sons, Inc., Jan. 2018.

[19] J. A. Yetter, "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," NASA Langley Research Center, Hampton, VA, USA, Tech. Rep. NASA-TM-109158, Jan. 1995.

[20] L. Wheatcraft, T. Katz, M. Ryan, and R. B. Wolfgang, "Needs, requirements, verification, validation lifecycle manual," INCOSE Requirements Working Group, Tech. Prod. INCOSE-TP-2021.002-01, Jan. 2022.

Appendix
Nonempty Attribute Counting Script

Below is the Groovy script used to count how many MBSRs have a nonempty [**Who**] attribute where `element_list` contains all MBSRs in a particular context:

```
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;
```