Model-Based Structured Requirements in SysML

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

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

Kamran Eftekhari-Shahroudi
Woodward, Inc.
Fort Collins, CO, USA
Department of 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 (SE). This 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, 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 engineering; systems engineering; digital engineering**

I. Introduction

Architecture-centric practices are gaining widespread acceptance in systems engineering (SE). This 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. Fortunately, there is codified and disciplined support for MBSE in the form of methodologies, formal modeling languages, and tools [1].

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

The Systems Modeling Language (SysML) is a widely-supported graphical modeling language that prescribes a set of object-oriented elements and allowable relationships between elements creating an integrated system model that is interacted with through diagrams and other mechanisms [2], [3]. Software tools are the key enables 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 [4].

Central to the rules that govern a system are the requirements that are 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 (RE) is concerned with all aspects of discovering, validating, and managing requirements [5]. 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 well-defined, complete, valid, and adaptable representation of the requirements in the system model.

There are a variety of approaches to developing and managing requirements [5]–[9]. However, there is still a common problem of much effort spent understanding the requirements and many iterations because these guidelines and rules must be adhered to manually by engineers, which are prone to errors and cognitive barriers in understanding large, complex ideas. Furthermore, many requirement approaches do not fully integrate with the system model (i.e., there are limited relationships between the textual requirements and model elements). This issue leads to increased cost and risk associated with 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.

The contributions in this papers are summarized as 1) a model-based structured requirement (MBSR) stereotype in SysML is demonstrated, 2) the addition of organizational attributes to requirements is shown, 3) and several examples are made available comparing the MBSR approach to existing

approaches. The rest of the paper is organized as following: Sec. II describes the proposed approach; Sec. III provides several examples of requirements using two existing approaches as well as the proposed 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 [5]:

- *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.
- *Achievable*—Reasonable enough to be implemented within the available resource constraints (e.g., cost, schedule, technical).
- *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 quite 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 for developing requirements.

A. Structured Requirement

We start by describing the existing concept of a structured requirement [10]. 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 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**] during [**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 [5]:

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

with an example structured requirement 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 [6].

This approach to requirements modeling helps improve the overall quality of the requirement statements by adding needed structure while defining a 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 a imprecise relationship to the system model.

B. Classical SysML Requirements Modeling

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

- *Id* typed by an identification tag/serial number.
- *Name* typed by a general name of a requirement.
- *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.

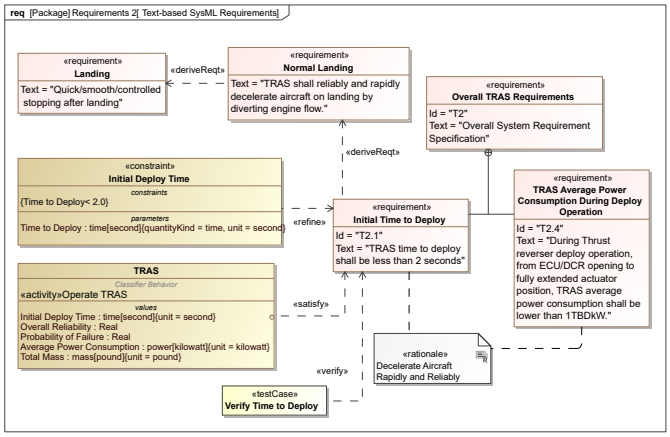

Fig. 1: Classical SysML requirement diagram utilizing text attribute for the *shall* statement and several relationships.

- *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 document.
- *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 the requirements.

To illustrate the classical SysML requirement modeling, Fig. 1 is a requirements diagram which is a view into the system model for several requirements and their traceability relationships. For example, T2; Overall TRAS Requirements is nested further to contain requirements T2.1; Initial Time to Deploy and T2.4; TRAS Average Power Consumption.

To represent the traceability of a requirement T2.1; 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 can be linked as a rationale element 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 test verification activity specified by a test

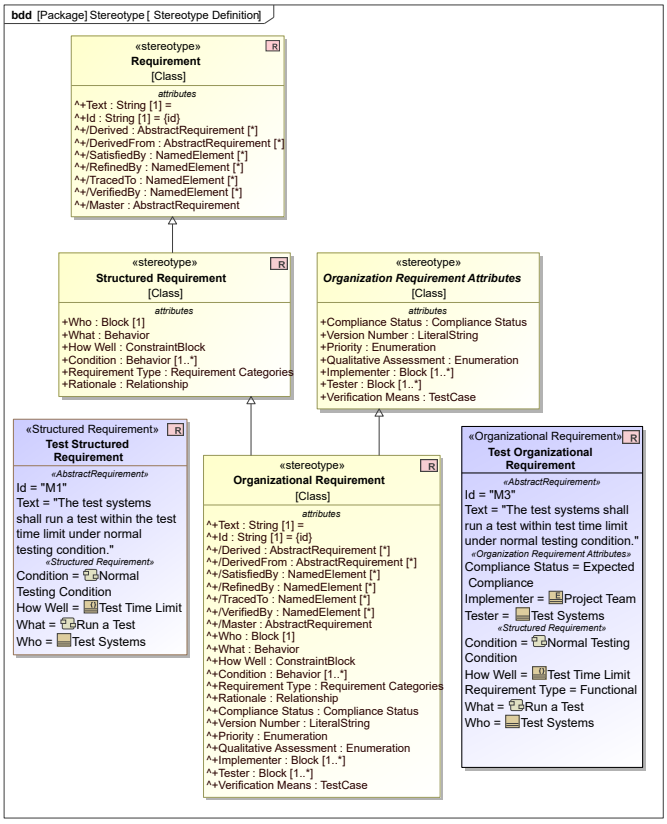

Fig. 2: Stereotype definitions for model-based structured and organizational requirements.

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 elements. 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 system model. However, the relationships between model elements (e.g., 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. 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 are 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 at this time 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 SysML 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 classifier behavior that can be typed by the same elements as [**What**].
- **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 types.
- **Rationale** refers to the underlying reason why a requirement exists so is specified by a SysML dependency, a type of relationship, such as «*DeriveReqt*».

Now, with «*Structured Requirement*» defined, a 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 Test 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.

This is a disciplined approach for capturing requirements as well as specific aspects of a system 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 [11].

[1]In the current implementation, only existing SysML classifiers could be utilized for this typing activity. Future work will support customized classifiers based on combining different types and limiting types based on stereotypes.

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 stereotype, as shown in Fig. 2. Some specific included organization specific attributes, defined with classifier types and optional multiplicities, are:

- **Version** indicates a text or serial number typed by a literal string.
- **Assessment** refers to description of a requirement's quality specified by an enumerated list of quality properties that follows specific guidance, and defined by a criteria.
- **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, non-compliant, or TBR.
- **Implementer** refers to the stakeholder responsible for implementing a requirement to deliver a capability or achieve a system characteristic 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.
- **Priority** indicates the importance of a requirement on the basis of stakeholder interest, typed by an enumeration list.

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 Test Organizational Requirement, is shown in Fig. 2 with more detailed examples provided in Sec. III.

These attributes can be customized to fit a particular organization's needs. Additionally, by further breaking down a requirement statement into organization-specific attributes,

ID	Name	Requirement Type	Text	Rationale
1.1	Initial Time to Deploy	Performance	TRAS time to deploy shall be less than 2 seconds	Decelerate Aircraft Rapidly and Reliably
1.2	Overall Reliability	Non-Functional	TRAS shall achieve overall system reliability no less than 0.9 under normal landing condition.	Decelerate Aircraft Rapidly and Reliably
1.3	TRAS Probability of Failure	Non-Functional	TRAS composite probability of failure shall be no more than 1E-9	Decelerate Aircraft Rapidly and Reliably
1.4	TRAS Average Power Consumption	Operational	During Thrust reverser deploy operation, from ECU/DCR opening to fully extended actuator position, TRAS average power consumption shall be lower than 1TBDkW.	Decelerate Aircraft Rapidly and Reliably
1.5	Unlocking for TR Deployment	Functional	The TRAS actuators shall include overstow function to ensure unlocking function is performed without any loading during TR deployment.	Improve Efficiency, Safety and Sustainability
1.6	Total Mass	Physical	System total mass shall be less than 320 pounds	Differentiate with More Electric Aircraft
1.7	Tertiary Lock Relocking	Design Constraint	Each tertiary lock shall be designed to not relock during T/R deploy and stow translation.	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	T2	T2 Overall TRAS Requirements	Overall System Requirement Specification					
2	T2.1	T2.1 Initial Time to Deploy	TRAS time to deploy shall be less than 2 seconds	Decelerate Aircraft Rapidly and Reliably	3.1 Normal Landing	Initial Deploy Time : time[second]	Initial Deploy Time	Verify Time to Deploy
3	T2.2	T2.2 Overall Reliability	TRAS shall achieve overall system reliability no less than 0.9	Decelerate Aircraft Rapidly and Reliably	3.6 TR Probability of Failure	Overall Reliability : Real	Overall Reliability	Verify Overall Reliability
4	T2.3	T2.3 TRAS Probability of Failure	TRAS composite probability of failure shall be no more than 1E-9	Decelerate Aircraft Rapidly and Reliably	3.6 TR Probability of Failure	Probability of Failure : Real	Probability of Failure	Verify Probability of Failure
5	T2.4	T2.4 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 1TBDkW.	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
6	T2.5	T2.5 Unlocking for TR Deployment	The TRAS actuators shall include overstow function to ensure unlocking function is performed without any loading during TR deployment.	Improve Efficiency, Safety and Sustainability	3.2 Power Dissipation	Safely Dissipate Regenerative Energy	Without Loading	Verify Unlocking at No Load
7	T2.6	T2.6 Total Mass	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
8	T2.7	T2.7 Tertiary Lock Relocking	Each tertiary lock shall be designed to not relock during T/R deploy and stow translation.	Decelerate Aircraft Rapidly and Reliably	3.3 Safety Against IAS	Control Lock	Prevent Relocking	Verify TL Relocking

(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	Rationale	Who	What	Condition	How Well	Satisfied By	Refined By	Verification Means
1	M2	M2 Overall TRAS Requirements		Overall System Requirement Specification								
2	M2.1	M2.1 Initial Time to Deploy	Performance	TRAS time to deploy shall be less than 2 seconds	DeriveReqt[Initial Time to Deploy -> Normal Landing]	TRAS	Operate TRAS	Normal Landing Scenario	Initial Deploy Time	Initial Deploy Time : time[second]	Initial Deploy Time	Verify Time to Deploy
3	M2.2	M2.2 Overall Reliability	Non-Functional	TRAS shall achieve overall system reliability no less than 0.9 under normal landing condition.	DeriveReqt[Overall Reliability -> TR Probability of Failure]	TRAS	Operate TRAS	Normal Landing Scenario	Overall Reliability	Overall Reliability : Real	Overall Reliability	Verify Overall Reliability
4	M2.3	M2.3 Probability of Failure	Non-Functional	TRAS composite probability of failure shall be no more than 1E-9	DeriveReqt[TRAS Probability of Failure -> TR Probability of Failure]	TRAS	Operate TRAS	Normal Landing Scenario	Probability of Failure	Probability of Failure : Real	Probability of Failure	Verify Probability of Failure
5	M2.4	M2.4 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 1TBDkW.	DeriveReqt[TRAS Average Power Consumption During Deploy Operation -> Energy Efficiency Gain]	Power Interface	Control Energy	Normal Landing Scenario	TRAS Average Power Consumption During Deploy Operation	Average Power Consumption : power[kilowatt]	TRAS Average Power Consumption During Deploy Operation	Verify Average Power Consumption
6	M2.5	M2.5 Unlocking for TR Deployment	Functional	The TRAS actuators shall include overstow function to ensure unlocking function is performed with loading during TR deployment.	DeriveReqt[Unlocking for TR Deployment -> Power Dissipation]	Actuators	Perfrom Overstow	TR Deployment Scenario	With Loading	Safely Dissipate Regenerative Energy	Without Loading	Verify Unlocking at No Load
7	M2.6	M2.6 Total Mass	Physical	System total mass shall be less than 320 pounds	DeriveReqt[Total Mass -> TR Level Weight Constraint]	TRAS	Operate TRAS	Normal Testing Condition	Total Mass	Total Mass : mass[pound]	Total Mass	Verify Total Mass
8	M2.7	M2.7 Tertiary Lock Relocking	Design Constraint	Each Tertiary lock shall be designed to not relock during T/R deploy and stow translation.	DeriveReqt[Tertiary Lock Relocking -> Safety Against IAS]	TL	Control Lock	TR Deployment Scenario	Prevent Relocking	Control Lock	Prevent Relocking	Verify TL Relocking

(c) Table summarizing the requirements captured as structured organizational requirements (another view of the model elements in Fig. 4).

Fig. 3: Several ways of implementing structured requirements.

both engineers and managers can more readily quantify and visualize several indicators of requirement quality and identify gaps/missing information.

III. Examples

In this section, we present several more detailed examples of requirements implemented as MBSRs, as well as pure textual structured and classical SysML requirements. The Cameo System Modeler models for these examples are available at [12]. All three are summarized in Fig. 3. 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 or slippery runways by reversing the flow of fan bypass air [13]. TRAS also reduces brake wear and provide control under emergency scenarios [14].

A. Textual Requirements

As we will see, much of the necessary information is captured in the pure spreadsheet version in Fig. 3a 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

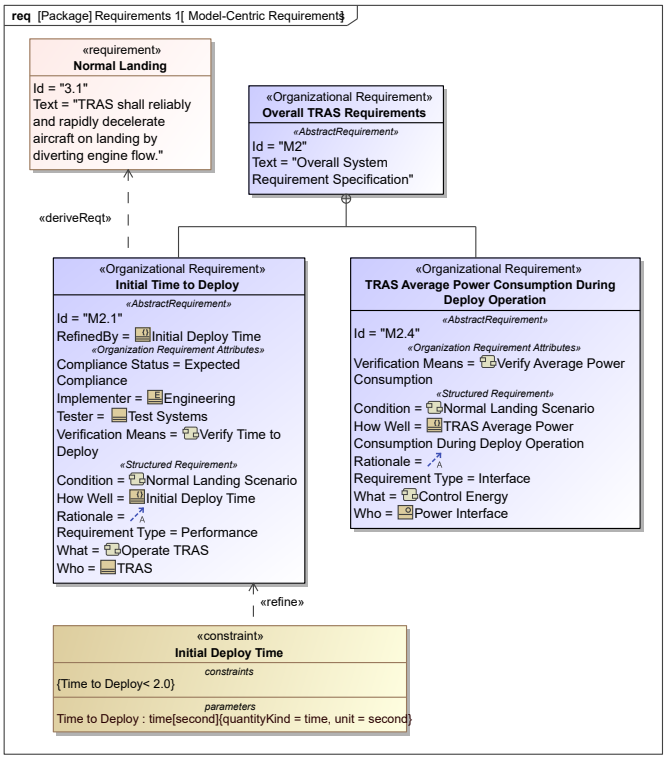

Fig. 4: Model-centric SysML requirements utilizing the structured organizational requirement stereotype defined in Fig. 2 (and is an alternative representation of the requirements first presented in Figs. 1 and 3b).

the system of interest, so requirements cannot be readily developed in a manner consistent with 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 analysis 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. 3b. 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 (view through this table) is always up to date and linked. We also start to see that some model elements are reused in this 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 text-based statement, which is not generally linked to the system model.

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. 3c, two of the requirements are visualized in the requirement diagram in Fig. 4.

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. Now, if the normal landing scenario definition changed, the requirement has a direct link to those changes (and many tools can support interactive activities when you want to understand what is impacted by such a change in the model, i.e., suspect links [15]).

The other relationships from the classical SysML should certainly be included. For example, «*deriveReqt*» is used to understand the source of requirement, Initial Time to Deploy as shown in Fig. 4. 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, in Fig. 4, we see that Compliance Status as been filled in.

While the summarizing table in Fig. 3c is complete (i.e., no missing cells), this generally will not be the case as the system is being developed and requirements are being modified. It would be quite easy to pin point requirements that are missing certain attributes. Furthermore, because the definition of the structured requirement attributes are based on model elements, it is much easier to determine if a 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**] for several requirements. This is automatically visualized in a relation map that includes some of the key «*Structured Requirement*» attributes. Such a map 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 Lock are only associated with a single requirement (in this view). This is to say that by including more specific relationships between the requirements and model elements, we have enabled new mechanisms for defining requirements in a model-based context.

[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.

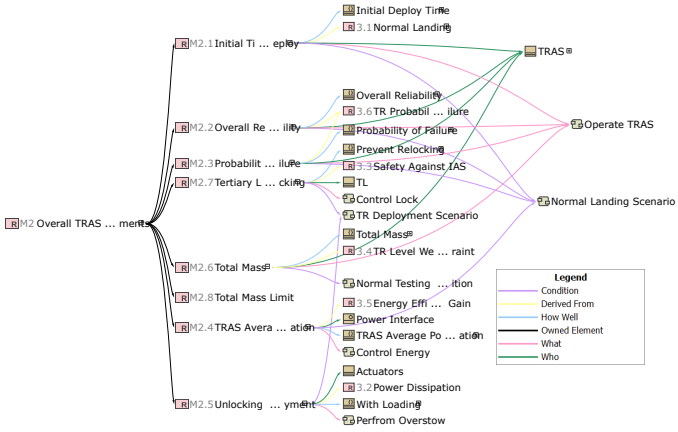

Fig. 5: Model-driven relation map between the requirements attributes and model elements.

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 rigor when developing the system model (and requirements) that support the wide range of SE activities.

As an organization, we could follow and mandate conformance to various standards and guidelines for requirements, such as [5], but this does not generally force engineers to adhere to them (and certain requirement tools certainly do provide some means to address this already). The MBSR *restricts* us to create and define the right elements and relationships (or readily see that they are missing). Either they must spend significant resources adhering to them or others must spend time fixing them during reviews or worse, only identifying issues late in the development cycle. Having such a structured approach, based on the system model that is also being developed and reviewed, certainly has the potential to help produce requirements that are complete, correct, verifiable, and implementable.

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 exists between elements. Basic attributes such as a element names are kept in sync as well as the digital artifacts (such as a diagrams and tables) that use them. Leveraging the power of SysML to create and maintain interrelationships between elements and artifacts within a model, the impact of a change can be assessed more readily by reviewing the suspicions raised by the SysML tool, hence the analysis is more rigorous and can be automated. Most MBSE tools support some form of suspect links [15], which indicate if something should be reviewed for validity is a related element is modified and are only possible through leveraging modeled relationships. 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 available of the SysML tools, it might be possible to automatically generate the text statement in Fig. 3c only using the structured attributes. After all, the structured requirement template is simply filling in the placeholders with the natural language names for the who, what, how well, and condition attributes in Eq. (1). Furthermore, metrics on the requirements can be readily computed, such as current percentage that are complete, individual attribute complete percentages, rankings of elements with the most relationships to the requirements, etc.

Supporting the mission of automating the requirements analysis from the perspectives of completeness, correctness, and consistency, there is an opportunity to develop customized validation rules. Many such rules are often already built into MBSE tools, but with the specified MBSR structure, we could readily assess the severity of different 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 is supposed only to be a block. Thus, the inclusion of 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 concepts 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 included defining who, what, how well, and condition attributes to construct a structured requirement statement in natural language, but is flexible to other structures. Additionally, the expansion of organization attributes was demonstrated, which included adding structuring of the requirements compliant to organization's policies and rules. MBSRs were demonstrated with a notional thrust reverser actuation system (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] *OMG System Modeling Language*, Object Management Group Std. 1.6, Dec. 2019. [Online]. Available: https://www.omg.org/spec/SysML/1.6

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

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

[5] *ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering*, IEEE Std. 29 148-2018, Rev. ISO/IEC/IEEE 29148:2011(E), Nov. 2018, doi: 10.1109/IEEESTD.2018.8559686

[6] K. Pohl, *Requirements Engineering - Fundamentals, Principles, and Techniques*. Springer, 2010.

[7] "Guide for writing requirements," INCOSE Requirements Working Group, Tech. Rep. INCOSE-TP-2010-006-03, Jul. 2019.

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

[9] *IEEE Guide for Developing System Requirements Specifications*, IEEE Std. 1233-1998, Rev. IEEE Std 1233-1996, Dec. 1998, doi: 10.1109/IEEESTD.1998.88826

[10] 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

[11] 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

[12] Model-based structured requirements examples. [Online]. Available: https://github.com/danielrherber/available-at-final-submission

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

[14] 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.

[15] Suspect links. No Magic, Inc. [Online]. Available: https://docs.nomagic.com/display/MD190/Suspectlinks