VT-CM-19-197

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Automatic Generation of Contractual Requirements from MBSE Artifacts

6 September 2019

Dr. Alejandro Salado

Paul Wach

Virginia Tech

The research presented in this report was supported by the Acquisition Research Program of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

Executive Summary

This report describes recent research in support of acquisition programs that leverage requirements as contractual elements. Textual requirements form the backbone of contracting in acquisition programs. Requirements define the problem boundaries within which contractors try to find acceptable solutions (design systems). At the same time, requirements are the criteria by which a customer measures the extent to what their contract has been fulfilled by the contractor. However, current government reports and academic research show latent problems in acquisition programs stemming from poor practices in requirements engineering. In order to cope with such a challenge, academia and industry envision extending the application of MBSE beyond conceptual design, particularly addressing problem formulation. Two main paths to integrate requirements within a complete MBSE environment are currently pursued. In the first path, major modeling languages, such as SysML, incorporate elements called requirement models, which are intended to model the requirements the system is expected to fulfil. However, the only modeling value of this approach is to achieve traceability between requirements and architectural elements. In the second path, researchers propose to use behavioral models of the system of interest as problem definition elements (requirements). However, the proposition remains positional, since such work has not addressed how contracting in acquisition programs is affected, or needs to be adjusted, to incorporate behavioral models as a contractual mechanism instead of textual requirements. Hence, the near-term, practical feasibility of the approach is questionable.

In order to cope with these challenges, this research project addressed the main question of whether contractual requirements in textual form can be automatically generated from requirement models in an MBSE environment without loss of information or intent. In particular, this research had the following objectives: (1) Create requirement models that can capture all information and intent of textual requirements; and (2) Translate requirement models into textual requirements without loss of information or intent in terms of contractual needs in acquisition programs. The research employed a combination of theoretical foundations and tool development and

implementation. The hypotheses were tested on an Air Force Institute of Technology notional satellite.

By fulfilling the research objectives, the results of this research are anticipated to significantly improve the performance of acquisition programs, in particular with regards the generation of contractual requirements. Furthermore, the direct public benefit of this research is anticipated to be higher early efficacy of commercial products and public services. Finally, while we considered an application for the Air Force as a test case, we anticipate that the methodologies and insights provided in this work can be applicable to a broad range of systems that require careful definition of requirements: other defense systems, space systems, aeronautics, automotive systems, manufacturing systems, electronic products, civil infrastructure, public health systems, or transportation systems.

The research has already resulted in one published paper for the 2019 Acquisition Research Symposium, one published paper for the 2019 Conference on Systems Engineering Research (CSER), and one published paper in the Systems journal. Several other conference and journal papers resulting from this research are currently under preparation and will be submitted before the end of 2019.

VT-CM-19-197

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Automatic Generation of Contractual Requirements from MBSE Artifacts

6 September 2019

Dr. Alejandro Salado

Paul Wach

Virginia Tech

Disclaimer: This material is based upon work supported by the Naval Postgraduate School Acquisition Research Program under Grant No. HQ0034-18-1-0006. The views expressed in written materials or publications, and/or made by speakers, moderators, and presenters, do not necessarily reflect the official policies of the Naval Postgraduate School nor does mention of trade names, commercial practices, or organizations imply endorsement by the U.S. Government.

THIS PAGE LEFT INTENTIONALLY BLANK

Table of Contents

THIS PAGE LEFT INTENTIONALLY BLANK

List of Figures

THIS PAGE LEFT INTENTIONALLY BLANK

List of Tables

THIS PAGE LEFT INTENTIONALLY BLANK

Background

Textual requirements form the backbone of contracting in acquisition programs. Requirements define the problem boundaries within which contractors try to find acceptable solutions (design systems) (Alejandro Salado, Nilchiani, & Verma, 2017). At the same time, requirements are the criteria by which a customer measures the extent to what its contract has been fulfilled by the contractor, e.g. (INCOSE, 2015). Hence, it is not surprising that some authors consider requirements "the cornerstone of … systems engineering" (Buede, 2009). However, literature shows latent problems in acquisition programs stemming from poor practices in requirements engineering, e.g. (Dada, 2006; El Eman & Birk, 2000; McConnell, 2001; Yeo, 2002).

In order to cope with such a challenge, academia and industry envision extending the application of Model-Based Systems Engineering (MBSE) beyond conceptual design, particularly addressing problem formulation. Two main paths to integrate requirements within a complete MBSE environment are currently pursued. In the first path, major modeling languages, such as SysML, incorporate elements called *requirement models* (Friedenthal, Moore, & Steiner, 2015), which are intended to model the requirements the system is expected to fulfil. Some authors have attempted to demonstrate how those so-called *requirement models* can be used to move acquisition practice from document-centric (textual) requirements to model-based requirements, e.g. (J. Holt et al., 2015; Jon Holt, Perry, & Brownsword, 2011). However, this approach is based on defining specific model elements, called "requirements", which contain a text property that takes the textual requirement. The requirement element is then linked to a specific component in the system architecture. Hence, the only modeling value of this approach is to achieve traceability between requirements and architectural elements. Although this is valuable on its own merit, requirements remain textual; thus, model-based requirements are not achieved.

In the second path, researchers propose to use behavioral models of the system of interest as problem definition elements (requirements), e.g. (Miotto, 2014). Such work has been confined though to the technical challenges of modeling expected system behavior. Therefore, the proposition remains positional, since such work has not addressed how contracting in acquisition programs is affected, or needs to be adjusted, to incorporate behavioral models as a contractual mechanism instead of textual requirements. Hence, the near-term, practical feasibility of the approach is questionable.

In a third path, less extended, mathematical or formal structures are used to capture requirements, e.g. (Micouin, 2008). In these approaches, *shall* statements or similar natural language statements are not used in the formulation of the requirement. In the context of the research presented in this report, these representations may be considered examples of true model-based requirements. Their usage in the context of SysML is however not evident.

The research presented in this report overcomes those key problems by providing a translation mechanism that enables the engineering of true requirement models, while automatically generating corresponding textual requirements. The research provides two main contributions. First, it provides constructs to generate requirement models that can capture at least the same information and intent as textual requirements. The true model-based requirements are based on formal systems theoretic constructs (Wymore, 1993), an orthogonal requirements taxonomy (Alejandro Salado & Nilchiani, 2014), and an existing interface taxonomy (Kossiakoff, Sweet, Seymour, & Biemer, 2011). This is expected to extend MBSE capabilities in problem formulation, which are currently lacking. Second, it provides a requirement translation process that automatically generates textual requirements from the developed requirement models. The translation process leverages requirement templates (Alejandro Salado & Wach, 2019a) that fulfill guidelines for good requirements (INCOSE, 2012). This is expected to enable the technical team to transition to model-based requirements, while guaranteeing fulfilling the expectation of contractual departments and acquisition programs.

Literature Investigation

Note: This section has been slightly adapted from a publication by the authors prepared, submitted, and published during the period of performance of this research (Alejandro Salado & Wach, 2019b).

The majority of the literature in model-based requirements deals with aspects related to requirements management. Some examples include work on requirements traceability and allocation (e.g., (Badreddin, Sturm, & Lethbridge, 2014; Borgne, Belloir, Bruel, & Nguyen, 2016; Holder et al., 2017; J. Holt et al., 2012; Marschall & Schoemnakers, 2003; Mordecai & Dori, 2017; Ribeiro, 2018; Schmitz, Nissen, Jarke, & Rose, 2010)) and modeling requirements engineering and management processes (e.g., (J. Holt et al., 2012; J. Holt et al., 2015; S. P. J. Holt, M. Brownsword, D. Cancila, S. Hallerstede and F. O. Hansen, 2012; S. P. J. Holt, R. Payne, J. Bryans, S. Hallerstede and F. O. Hansen, 2015)). However, as stated in the previous section, such work does not address epistemological and structural aspects of model-based requirements. Therefore, this section is limited to prior literature specifically addressing defining and developing model-based requirements.

A common approach for modeling non-functional requirements is to capture them as properties or attributes of the system (e.g., (H. Reza, 2017; M. Saadatmand, 2012)). In particular for SysML, this becomes handy because the approach is easily implementable by defining *values* for the physical block that represents the system of interest (Fockel & Holtmann, 2014; J. Holt et al., 2015). However, this approach presents two weaknesses. The major one relates to the ambiguous interpretation of what a non-functional requirement is (Alejandro Salado & Nilchiani, 2014). It is discussed later in this report that a distinction between functional and non-functional requirements does not really exist. As a result, modeling requirements in this way can yield severe inaccuracies in capturing the real requirement. The second weakness, while minor, is of conceptual nature. Requirements ought to define the external boundaries of the system, the solution space. Therefore, defining a requirement as a property or attribute of the system is conceptually inconsistent.

A higher level of sophistication in creating model-based requirements can be found in the field of software. Work in this area is aimed at transforming requirements in natural language into models, as part of the need elicitation and requirements derivation activities. For example, the Model-driven Object-oriented Requirements Editor (MOR Editor) parses a requirement text into a set of properties or constraints associated to objects, called requirement elements (Lu, Chang, Chu, Cheng, & Chang, 2008). A similar approach is used to interpret user stories (F. Wanderley, A. Silva, Araujo, & Silveira, 2014). The structure of the requirement models in these cases is derived from a template used to capture user stories in natural language. However, a theoretical framework for the template is not prescribed in these cases. While such approaches provide great flexibility, the resulting model structures acquire the limitations inherent to the template in natural language.

Mathematical definitions alone cannot solve this problem. For example, in the Requirements Driven Design Automation framework (RDDA) a requirement model is defined as

$$M_R = \langle P, A, S, F, C, N_C, R \rangle$$

where:

P is the set of products described,

A is the set of applications (with $A \subset S$),

S is the set of subsystems,

F is the set of features,

C is the set of constraints,

N_C is the set of constraint numeric descriptors, and

R is the set of relationships on these sets describing the model (Cardei, Fonoage, & Shankar, 2008).

This structure enables the automation of certain types of analyses in ways that natural language templates cannot (Cardei et al., 2008). However, the model itself is internally inconsistent, at least with respect to good practices for requirements engineering and other theoretical developments. For example, subsystems are defined as part of the requirement model. While a relationship between a system requirement and subsystem is meaningful, the requirement should be free of implementation prescriptions (INCOSE, 2012; Alejandro Salado et al., 2017). Furthermore, as earlier identified, some elements in the model are not orthogonal. For example, formal definitions are not provided for the terms, *constraint* and *feature*. This lack of orthogonality may yield inconsistencies in the definition of requirements. Such problems could be overcome by using a requirement model that is grounded on an internally consistent theory. This idea is central to the research presented in this report.

In this regard, the notion of semi-lattice has been used to define a requirement. Specifically, a requirement is considered a combination of a condition (e.g., when flying), a carrier (e.g., the system), a property (e.g., power consumption), and a domain (e.g., less than 100 W) (Micouin, 2008). Similar patterns are found in the literature, such as *check <condition> after <condition> within <time>* (Borgne et al., 2016). While internally consistent, these definitions present two problems. The first one stems from defining requirements as properties of the system, as discussed earlier. The second one relates to the way in which *conditions* may be defined. In particular, the definition does not prescribe against defining any type of system *state* as a condition. In fact, using *states* and *transitions between states* seems to be prevalent as the fundamental model to capture requirements. Using this concept, a functional requirement can be modeled as a required transition from one state to another (D. Aceituna, Do, Walia, & Lee, 2011; Daniel Aceituna, Walia, Do, & Lee, 2014; S. Siegl, 2010). Fundamentally, this leads to modeling a functional requirement as a triplet, such as (S_c, T, S_n), where S_c is the current state, T is the action triggering the transition, and S_n is the next state following such action (D. Aceituna et al., 2011; Daniel Aceituna et al., 2014). Different required properties can then be linked to each one of those states and to the transition trigger. While the state-based model is valuable for modeling system behavior, it has a fundamental problem for modeling requirements broadly. Specifically, there is no formalism to define what the system must do in each state, or during the transition. This is critical, since the purpose of defining requirements is to understand the interaction on the system boundaries; as previously stated, the requirements define the solution space. In addition, there are multiple interpretations to the meaning of *state* and *transition* (Wach & Salado, 2019), which accentuates this problem.

A different approach to model-based requirements leverages what the system must do, as opposed to what characteristics it must exhibit or in what states it must transition. This conceptualization is also central to the research presented in this report. Prior work has attempted to model requirements as data exchanges and semantics associated with those exchanges (S. Teufl, 2013). However, the authors claimed that their approach was not broadly generalizable and that it had to be complemented with textual-based requirements (S. Teufl, 2013). Other authors suggest that requirements may be broadly defined as actions that the system must execute, specifically *The [Actor] shall [Action] [Object of Action] [to] [Recipient Actor]* (Miotto, 2014). This template was used as the basis to extract textual-based requirements from existing SysML diagrams, specifically from activity diagrams, state machine diagrams, and block diagrams. As such, formal model-based requirements were not defined, but standard SysML directly used as requirements. The weaknesses of this approach have been discussed earlier in this report. The textual formalism has some parallels with the theoretical framework that is presented in the next section, but with two key differences. First, defining the recipient actor for a system requirement is unnecessary, or at least should be abstracted. This is because the purpose of requirements is to define the boundaries of the system. Therefore, by definition, other external actors should not be included in the requirement, but their abstracted interaction should. This is in fact a key aspect in distinguishing stakeholder needs from system requirements. Incorporating external actors in a requirement statement introduces significant risks in the system development become it fails to decouple elements out of the control of the system development. Second, all requirements may be modeled by a minimal set of actions, as will be discussed later in this report, and, hence, they can be prescribed.

Model-Based Requirements

Note: This section has been slightly adapted from a publication by the authors prepared, submitted, and published during the period of performance of this research (Alejandro Salado & Wach, 2019b).

Theoretical Framework

The purpose of defining system requirements is to allow for distinguishing systems that are acceptable from systems that are not. In this research, system requirements are explicitly distinguished from stakeholder needs (Alejandro Salado et al., 2017), and only the former have been addressed. In this sense, a stakeholder need refers to a desire on an interaction between the system and an external system or actor. A system requirement takes the form of "an objective or criterion [that] a system is expected to fulfill" and could actually fulfill on its own (Alejandro Salado et al., 2017). In essence, system requirements define the conditions that a system needs to meet in order to enable the desires of the interaction captured in the form of stakeholder needs. For example, a stakeholder need would indicate the desired level of convenience while transporting something from A to B. Derived requirements would indicate the necessary mechanical vibration profiles provided to an abstraction of that *something*, which would enable such *convenience* once the system would be put in the operational context.

Wymore's conceptualization of *system* is central to the theoretical framework in this research. The starting point is considering that any system can be modeled as a transformation of input trajectories into output trajectories (Wymore, 1993). Since a set of requirements yields a solution space (that is, a set of systems that fulfill those requirements) (Alejandro Salado et al., 2017), it follows that a solution space can be modeled as a set of transformations of input trajectories into output trajectories. Consequently, **the central proposition of the theoretical framework employed in this research is that every requirement can be modeled as an input/output transformation**. In such models, the inputs, outputs, and transformations may be multidimensional.

Prior work in defining taxonomies for requirements supports this proposition. Requirement types can be reduced to four (Alejandro Salado & Nilchiani, 2014; A. Salado & Nilchiani, 2017), which are listed in Table 1.

Table 1. Requirement taxonomy [adapted from (Alejandro Salado & Nilchiani, 2014)]

Req Type	**Description**	**Examples**
Functional	What the system must do	The system shall image the Earth surface in UV spectral range. The system shall transmit image data according to Interface XYZ.
Performance	How well the system must perform its functions	The system shall have a resolution better than 1 m. The system shall have a field of view larger than 2 degrees.
Resource	What the system may consume to perform its functions at the required performance	The system shall consume less than 200 W. The system shall have a mass lower than 900 kg.
Environment	Settings or contexts in which the system must perform its functions	The system shall operate in vacuum. The system shall withstand shock levels higher than ABC.

These four types of requirements can be described as transformations of inputs into outputs:

> *Functional requirements* inherently describe input/output transformations. Mathematically, a function is necessarily defined as a mapping between a domain and codomain. From a General Systems Theory perspective, engineered systems are necessarily open (von Bertalanffy, 1969).
>
> *Performance requirements* are, as defined, necessary characteristics, properties, or attributes associated with the inputs and outputs of the transformations that the system shall perform. In fact, this condition is necessary because any attribute transparent to the interaction between the system and external systems should not be considered a requirement due to unnecessarily constraining the solution space (INCOSE, 2012; Alejandro Salado et al., 2017).

> *Resource requirements* define limits on resources that the system may consume. It is obvious that a resource must therefore be inputted to the system and that it is consumed for producing something. Hence, any limitation imposed on resource consumption is in fact part of a functional exchange and can be modeled in such a way.
>
> *An environment for the system* is an abstraction of boundaries between the system and external systems. The environment provides certain conditions under which the system must operate and imposes certain limitations on how the system may affect the environment. In other words, the environment provides certain *inputs* under which the system must operate and imposes certain limitations on the *outputs* the system may yield to the environment.

The feasibility of this idea in practice is further supported by Kossiakoff's taxonomy for external interfaces. According to the author, systems operate in three types of media: information, material, and energy, which become inputs to and/or outputs from the system (Kossiakoff et al., 2011). Hence, it is recognized that transformations are not limited to the logical domain (information) but can be performed on material and energy as well.

Finally, the requirement, as a required input/output transformation, is completed by a required interface through which the system can accept the necessary inputs and provide the desired outputs.

Basic Model of a Requirement

The basic model of a requirement consists of a logical component and a physical component. The logical component describes the required transformation. The physical component describes the interface(s) through which the transformation occurs. SysML's Sequence Diagram is used and extended to capture the logical component; SysML's Internal Block Diagram is used to capture the physical component.

The Sequence Diagram was chosen over other SysML model structures because it provides the necessary elements to model inputs and outputs without prescribing any internal behavior of the system. As shown in Figure 1, the basic Sequence Diagram consists of input signals, output signals, and system boundaries. System boundaries are represented by the line that models the system (called *lifeline*). This visualization forces the modeler to consider only the boundaries of the system (as opposed, for example, to using an activity diagram). We have chosen to use *Signals* in SysML notation to model the inputs and outputs of the system. The definition of signal allows for capturing all logical properties of required system inputs and outputs. In this way, the required transformation is captured as a property of the output signal, since it is defined as a function of system inputs. In addition, using signals allows for capturing specific types of messages, even though they may have the same logical meaning.

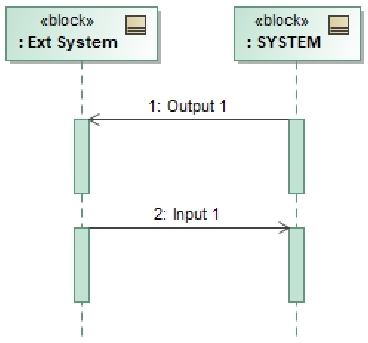


Figure 1. Basic Sequence Diagram components

Certainly, the Sequence Diagram is already part of SysML. Its use for modeling requirements demands two new aspects, which have developed in this research. The first one is related to intent. Specifically, the model does not capture the expected behavior of a system, but its required behavior. Therefore, the difference between a Sequence Diagram that captures the behavior of a system and one that captures its required behavior will not necessarily differ in their level of abstraction or accuracy of values. Yet, they are different in nature; the former is an abstraction of a system and the latter is an abstraction of a solution space. This difference in intention, although subtle, is captured through the values associated with the different elements of the model. This aspect will become more apparent in the next section.

The second new aspect is how *signals* are treated. Traditionally in SysML, a signal in a sequence diagram represents a logical message. Such formalism has been extended to let a signal represent any type of logical exchange between two systems, as described in Kossiakoff's taxonomy and described earlier in this report. In this way, a signal can represent an energy or material exchange, for example. Furthermore, the *signal* definition is extended to capture not only discrete messages, but also signals of continuous nature. Certainly, such continuity can still be modeled in SysML by using an infinite *loop* element that repeats at infinitely small periods. However, the extension allows for more efficient modeling. Instead of capturing continuous properties of the signal in the sequence diagram as a property of the overall exchange, they are directly captured as a property of the *signal* element.

Attributes of the elements *signal* are used to model the required characteristics of the inputs and outputs. Figure 2 shows an example. Attributes of input signals capture the conditions that the system is required to accept. Attributes of the output signals capture the characteristics that those outputs must exhibit. There is no prescription for how those properties may be defined; they may take the form of value ranges, images, or functions (such as the required transformation that is allocated to output signals). For this purpose, other SysML elements or diagrams may be used, such as a parametric diagram to capture a required transformation. But it should be noted, as described in the meta-model presented later, that the key elements that define the requirement are the signals; other modeling elements flow from them.

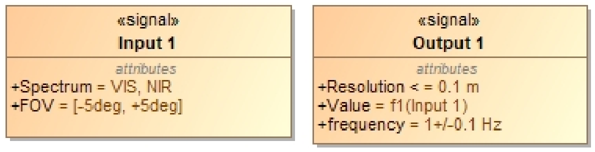


Figure 2. Signal attributes to capture required input and output characteristics

Physical interfaces are modeled as ports of a block representing the system. Using standard SysML practice, signals are then allocated to corresponding connections between ports, as shown in Figure 3. Contrary to most of the practice in MBSE, the requirement for the system is defined at port level, and not at part level. This makes the model consistent with the theoretical framework presented in the previous section: requirements define external transformations, not internal behavior. Ports are defined by *InterfaceBlocks*. Properties of these blocks capture requirements associated with the physical interface, as shown in Figure 4. As was the case for signals, there is no prescription for how those properties may be defined.

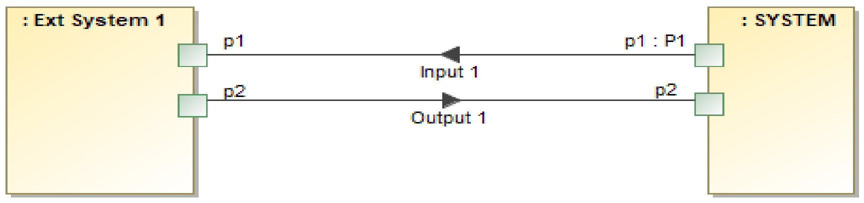


Figure 3. Basic Internal Block Diagram components

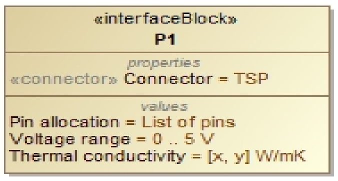

«interfaceBlock»
P1
properties
«connector» Connector = TSP
values
Pin allocation = List of pins
Voltage range = 0 .. 5 V
Thermal conductivity = [x, y] W/mK

Figure 4. Properties of physical interfaces

In addition, there is no prescription for what attributes and properties need to be defined for signals and ports. This is left open for each project. Although, to guarantee consistency with the theoretical framework presented in the previous section, the following rules apply:

- Properties that are related to the meaning of the exchange (e.g., packet configuration of a message, performance of a signal, etc.) must be modeled in the logical domain (i.e., signals). As previously stated, one such property is the transformation function, which is allocated to output signals.
- Properties that are related to the specific vehicle through which meaning is conveyed (e.g., electrical properties of a signal through which a message is sent) must be modeled in the physical domain (i.e., ports). These properties include aspects related to transport layer (such as data structure) and physical layer (such as voltage levels).

Adding Richness to the Requirement Model

Certain requirements may impose or define logical and time dependencies between inputs and outputs. In textual form, these may take forms such as *The system shall do C once conditions A and B are fulfilled* or *The system shall do B in less than X s after having done A*. The formal specification of SysML's Sequence Diagram provides modeling features that can enhanced the richness of the requirement model by capturing these dependencies. It is also noted in this case that such formalisms are used as models of the requirements (that is, of a solution space) and not as a model of the actual behavior of a system.

Conditional dependencies are captured using *interaction operators*. Note that these model elements become part of the requirement model as well. For example:

> *Alternative behavior*: This interaction operator can be used to capture a requirement to the system to exhibit different behaviors depending on certain conditions. No limitations are imposed in the conditions. They may refer to operational modes, historical actions, or external conditions (e.g., outside temperature). Figure 5.(a) shows an example of a requirement model, where the system is required to react to external commands only if it is in *On* mode; if in *Off* mode, the system is required to accept the command but to not react to it (which is a different requirement from stating that the system will not receive commands when in *Off* mode).
>
> *Parallel behavior*: This interaction operator can be used to capture a requirement to perform two or more transformations in parallel. No limitations are imposed on the type of elements that need to be executed in parallel. For example, they may refer to completely independent transformations that are performed in parallel, to the provision of several outputs simultaneously, or to the reception of inputs simultaneously. Figure 5.(b) shows an example of a requirement model, where the system must perform a background health monitoring exchange while also responding to other operational requests.
>
> *Loop*: This interaction operator can be used to capture a requirement to execute a transformation repeatedly until a condition is met.

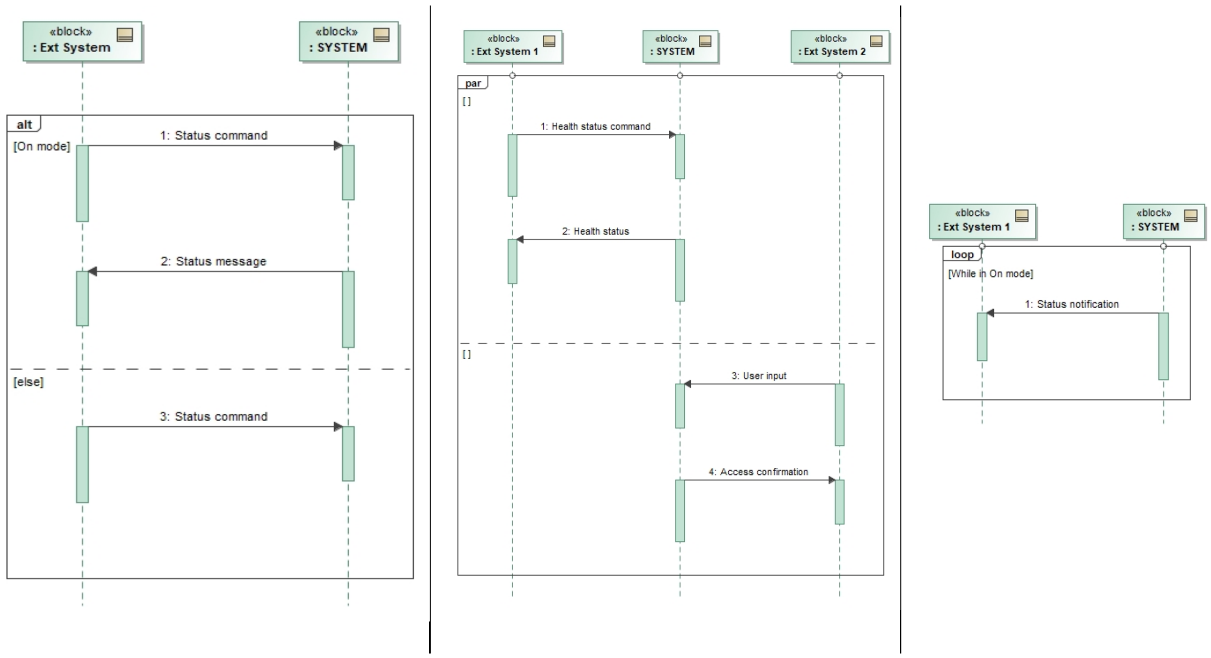


Figure 5. Examples of model-based requirements capturing various dependencies between inputs and outputs

(Left: alternative required exchange based on conditions; Center: exchanges that need to be executed in parallel; Right: Continuous exchange until a condition is met).

It should be noted that the example models in Figure 5 may be (and likely are) incomplete. For example, restrictions on time dependencies between inputs and outputs are not defined. From a problem formulation standpoint, Figure 5.(a) would imply that as long as the system would provide the *Status message* output after receiving the *Status command*, the system would be acceptable, regardless of the time that it would take it to do so. If this would not be the case (and it is generally not the case), requirements on time dependencies need to be captured in the model. This is done by adding *duration constraints* between the different inputs and outputs, as shown in Figure 6. It should be noted that requirements on timing dependencies are not limited to time lapses between inputs and outputs. They may also impose time constraints between outputs and define timing relationships between inputs. As a reminder, these *duration constraints* do not define the expected behavior of the system in this case, but instead its desired behavior.

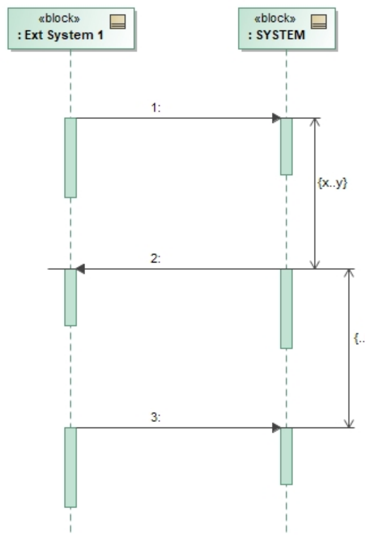


Figure 6. Example of time dependencies formalism

Modeling Simultaneity of Requirements Applicability

Given a set of requirements, different subsets may need to be fulfilled simultaneously. Simultaneity can be modeled using *interaction objects*, as described in the previous section. In fact, it is possible to depict the model of an entire set of requirements in a single sequence diagram. However, such an approach would not be convenient due to the resulting modeling and interpretation complexities. The SysML's state machine diagram has been extended for capturing simultaneity of requirements applicability.

State machine diagrams have been widely used to capture patterns of how a system behaves against certain sequences of input trajectories (Wymore, 1993). Using states allows for simplifying an otherwise infinite model of exchanges, by finding repetitions in the way in which inputs arrive to the system. This concept has been leveraged to create state-based requirements, or more precisely, mode-based requirements. In essence, a *mode requirement* is defined as a collection of requirements that do not have conflicting requirements and that must be fulfilled simultaneously. Conflicting requirements cannot be fulfilled simultaneously. Therefore, a mode requirement must necessarily be free of them. Otherwise, the mode requirement would likely lead to an empty solution space (Alejandro Salado et al., 2017). Simultaneity is also necessary, since it captures the notion that the system needs to fulfill several requirements at the same time. Furthermore, two mode requirements are distinct if and only if their collections of requirements are not identical. Using traditional jargon, this could be understood as modeling operational scenarios in which certain conditions apply simultaneously.

Specifically, the *state machine diagram* in SysML is extended to capture mode requirements as follows (ref. Figure 7). Each *state* element captures a mode. Each requirement (that is, the model of each requirement) that is applicable in that mode is linked to the corresponding *state* element. Note that, since the functional aspects of the requirement model are linked to its physical aspects, it is sufficient to link the *Sequence Diagram* to the *state* element. A final model element is needed to capture how each subset of the requirements becomes applicable. This is captured by one or more *sequence diagrams*, which model the required conditions for transitioning between mode requirements. These sequence diagrams link then to the states for which the transition occurs, as well as to the transition itself.

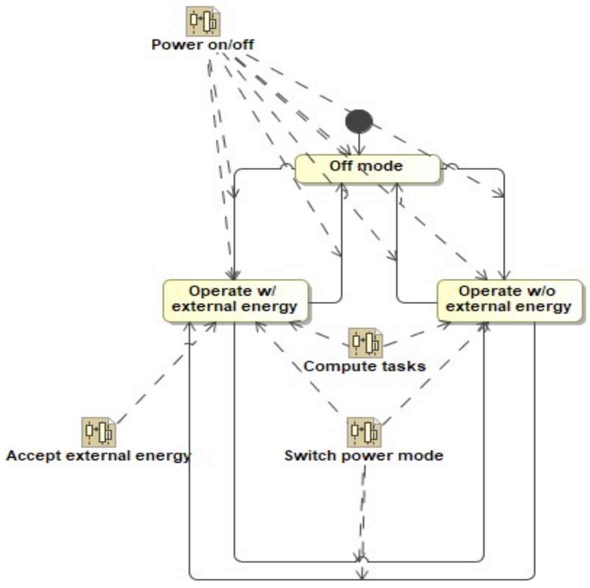


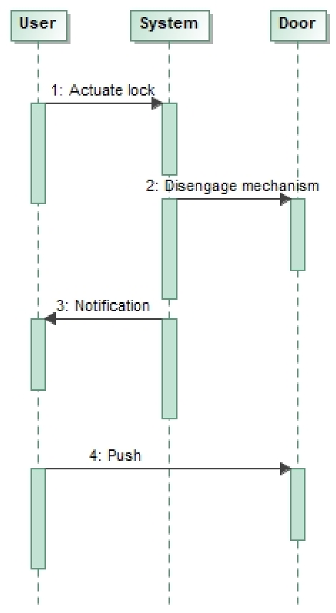
Figure 7. Example of mode requirements

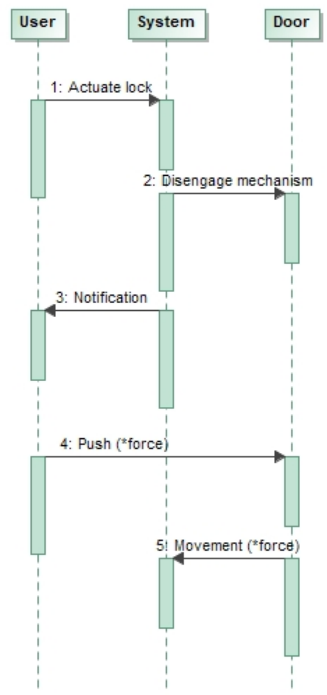
.

Non-Functional Requirements are Always Related to Functional Requirements

Considering that every requirement can be modeled as an input/output transformation makes distinguishing between functional and non-functional requirements unnecessary. This is a departure from most literature in requirements engineering. Its merit has been previously justified in this report. This section shows three specific instantiations of how traditional non-functional requirements can be better described as transformations.

The key paradigm change is to recognize that functional transformations are not limited only to operational or commandability aspects of a system's behavior. Instead, they also exist in physical interactions of the system with other external systems. Consider for example the requirements for a locking system on a door. With the traditional conceptualization, one would come up with requirements associated with the operation of the locking mechanism, as depicted in Figure 8 (a). Those requirements would be complemented with a set of non-functional requirements, such as vibration levels that the locking system needs to withstand, derived from the mechanical forces injected into the locking system. With the conceptualization presented in this report, those mechanical forces are also modeled as inputs to and outputs from the system, as shown in Figure 8 (b). As can be seen, pushing the door to open it injects a force into the door, which is transmitted as a force to the lock. Vibration levels are hence properties of those input signals to the system, not a property of the environment as an external system.





(a) (b)

Figure 8. (a) Traditional conceptualization of using a locking system. (b) Proposed conceptualization of using a locking system

These physical properties can eventually be linked to functional transformations, even if not apparent initially. For example, consider a requirement on the maximum mass of a satellite. As we discussed earlier, requirements ought to be defined on external boundaries of the systems, not as attributes of the system. Where does mass fit then? The first step is to question why there is a mass requirement in the first place. Let us keep it simple and state that the restriction on the satellite mass derives from the need to launch the satellite on a certain rocket. The satellite is attracted by Earth's gravity, pulling the rocket in an opposing direction from where it needs to go, which puts more energy demands on the rocket to leave the Earth. In fact, if the satellite could be designed so that it would float inside the rocket (that is, it would compensate gravity somehow), there would be no need for a mass requirement. Therefore, in essence, a restriction for the satellite mass exists because the satellite is providing an output to the rocket (force in one direction) at a particular joint between the satellite and the rocket (physical interface). Consequently, the mass requirement can be easily modeled as a property of a required logical output of the system (force) through a physical interface.

In addition, capturing non-functional requirements as models increases the level of precision over using natural language. Consider as an example the requirement *the system shall exhibit ABC color*. The three models in Figure 9 capture such a requirement. However, as can be seen, they describe a different solution space. The model in (a) indicates that the system has to exhibit such a color inherently, without using any external source. The model in (b) indicates that the system has electrical power available for use, when generating the color. The model in (c) indicates that the system can use the sunlight to provide the required color. It should be noted that in the three cases, the apparent non-functional requirement of *color* can be modeled as a functional exchange (e.g., as in (c), a paint is no more than a reflection -output- of incoming light -input). Certainly, all these model-based requirements can also be described in natural language. However, conceptualizing any requirement as a transformation pushes the requirement analyst to be explicit about his/her mental models with the requirement, increasing consistency and completeness.

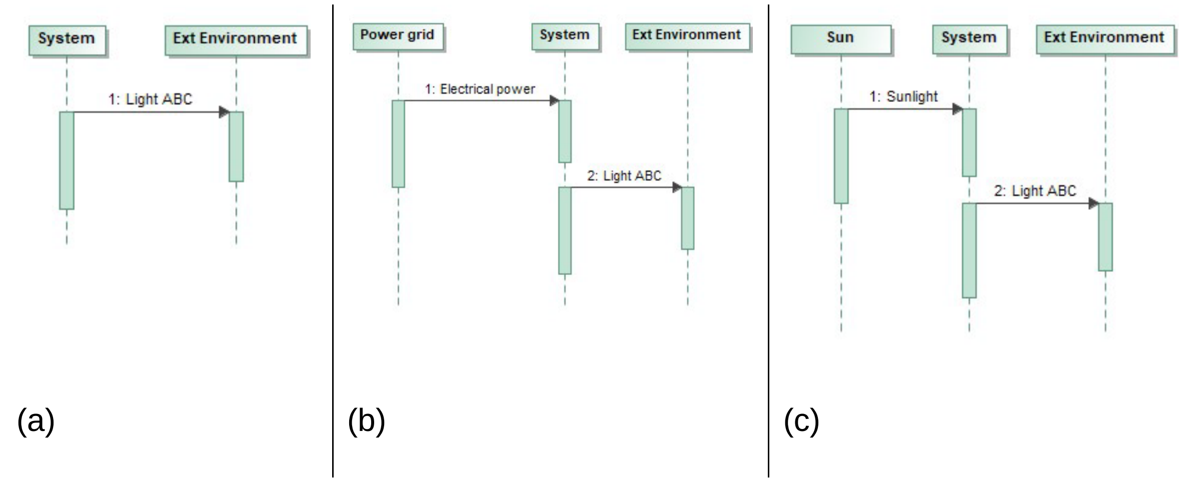


Figure 9. Three different models of a color requirement, capturing in fact different requirements

Meta-Model

Figure 10 shows a partial meta-model of the model-based requirements framework developed in this research. Its purpose is to serve as a general guide for the reader. Future work will address formalizing the meta-model. It should be noted that all elements and diagrams in Figure 10 extend the original elements and diagram in SysML, as described previously in this report.

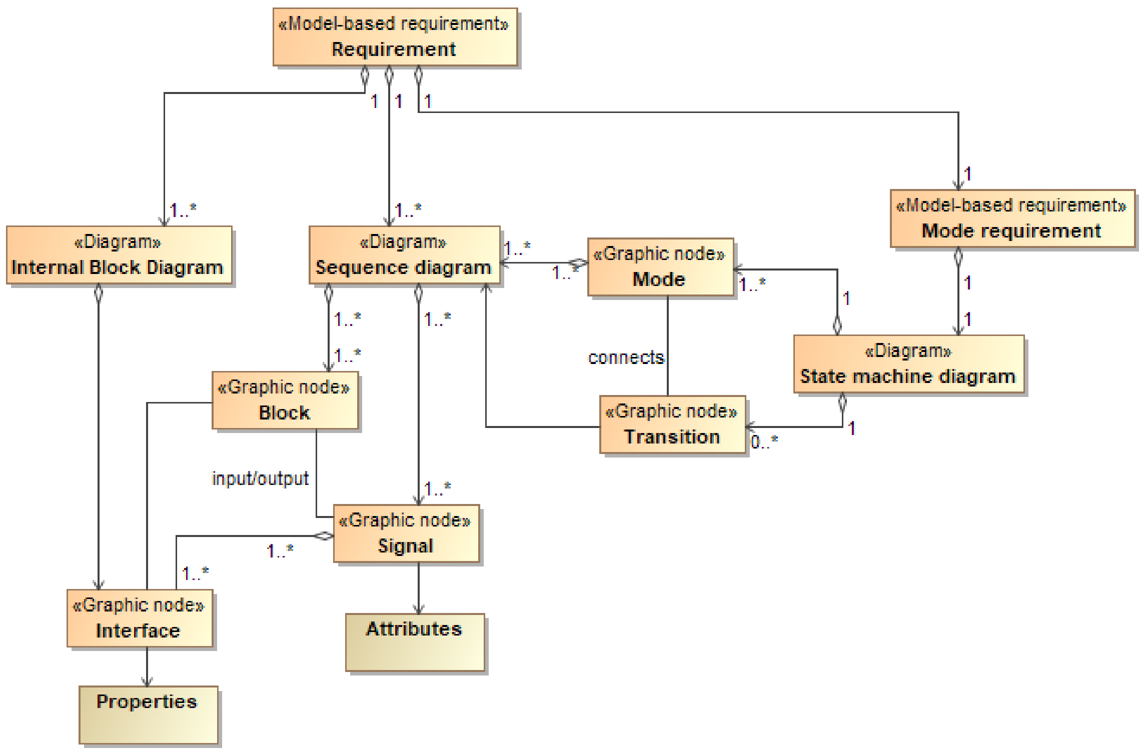


Figure 10. Meta-model of the requirements approach developed in this research.

The meta-model in Figure 10 explains how the different elements that form a model-based requirement relate to each other. These relationships are fundamental to guarantee that the definition of the model-based requirement is complete. Two aspects are worth noting in the meta-model. First, the meta-model shows that the model-based requirement is formed by three main pillars, as previously discussed: a sequence diagram (which captures the input/output exchanges), an internal block diagram (which captures the physical interfaces through which the inputs and outputs are conveyed by the system to external actors), and mode requirements (which describe the sets of requirements that apply together). Furthermore, the meta-model indicates that a model-based requirement can only be fully described if at least one element of each type (sequence, block, and mode) is defined and linked to each other.

Second, the three pillars that define a requirement are not independent. Instead, there is a complete linkage between all elements that are necessary to capture a requirement completely. Specifically:

1. Signals are linked to Interfaces, which makes the Sequence Diagram elements be connected to the Internal Block Diagram elements. This means that the logical and physical domains aspects of the requirement are connected.
2. Modes are linked to Sequence Diagrams, which makes the Sequence Diagram elements be linked to the Mode requirement elements. This means that that every required input/output exchange is contextualized within the overall requirement set for the system.
3. As a result of the previous two points, the Internal Block Diagram elements are also linked to mode requirements. This means that every required external interface is also contextualized within the overall requirement set for the system.

THIS PAGE LEFT INTENTIONALLY BLANK

Transforming Model-Based Requirements to Contractual Requirements

Process

As discussed previously, textual elements are the fundamental form by which contracts are established. Therefore, this project has assumed that transforming model-based requirements into textual requirements is essential to facilitate adoption of model-based requirements in real acquisition projects.

The process to transform the model-based requirements presented earlier to contractual requirements in natural language consists of four steps:

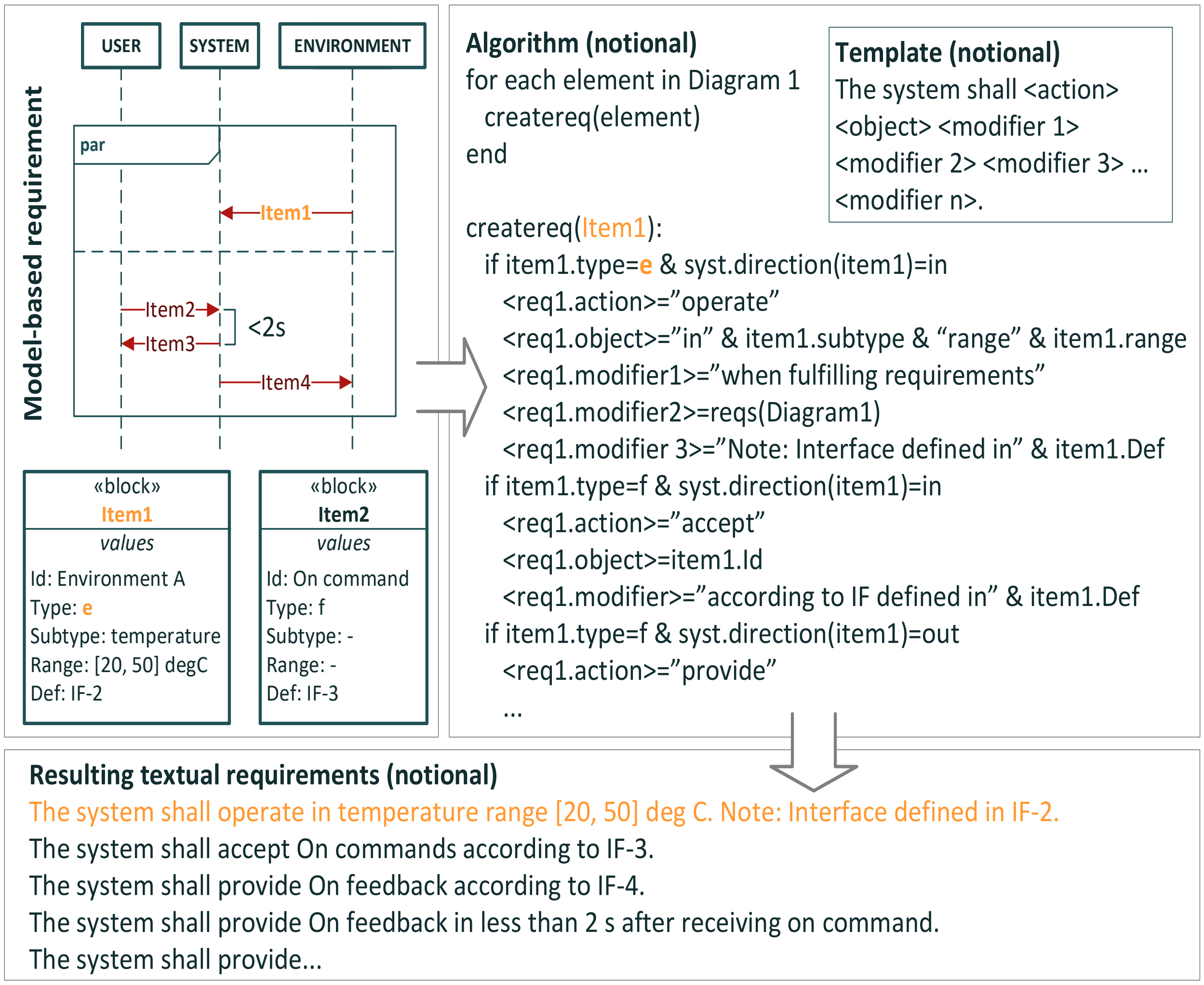
Step 1. For each port, generate corresponding textual requirements. This step generates a list of physical interfaces that are characterized by a set of required properties, which will be pointed at by the requirements resulting from the sequence diagrams.

Step 2. For each mode, generate simultaneity modifier. This step assigns tags to each sequence diagram associated to a particular mode. These tags are used later to associate a modifier to the textual requirements resulting from such sequence diagram that indicates the need to fulfill such requirement in the context of all other requirements with the same modifier.

Step 3. For each sequence diagram, generate corresponding textual requirements. This step generates a list that contains requirements associated with the need to accept inputs and provide outputs, the characteristics of those inputs and outputs, and the logical or temporal conditions for the acceptance of those inputs and provision of those outputs. In addition, for each requirement referring to the required inputs and outputs, a modifier referring to the physical interface through which such input or output is conveyed is added. Furthermore, the simultaneity modifiers in Step 2 are used to identify the subset of requirements that need to be fulfilled simultaneously.

Step 4. Remove repetitions, if any. Because inputs and outputs may be used in several sequence diagrams, this step will consolidate the list of requirements to avoid repetitions. It should be noted that this step can be executed after all textual requirements have been generated or as they are being generated, for efficiency purposes.

The basic concept for generating textual requirements leverages a predefined template of natural language requirements that maps to the different elements in the meta-model depicted in Figure 10. A simplified view of this concept is shown for illustrative purposes in Figure 11.



«block»
Item1
values
Id: Environment A
Type: e
Subtype: temperature
Range: [20, 50] degC
Def: IF-2

«block»
Item2
values
Id: On command
Type: f
Subtype: -
Range: -
Def: IF-3

Algorithm (notional)

```
for each element in Diagram 1
  createreq(element)
end

createreq(Item1):
  if item1.type=e & syst.direction(item1)=in
    <req1.action>="operate"
    <req1.object>="in" & item1.subtype & "range" & item1.range
    <req1.modifier1>="when fulfilling requirements"
    <req1.modifier2>=reqs(Diagram1)
    <req1.modifier 3>="Note: Interface defined in" & item1.Def
  if item1.type=f & syst.direction(item1)=in
    <req1.action>="accept"
    <req1.object>=item1.Id
    <req1.modifier>="according to IF defined in" & item1.Def
  if item1.type=f & syst.direction(item1)=out
    <req1.action>="provide"
    …
```

Template (notional)
The system shall <action> <object> <modifier 1> <modifier 2> <modifier 3> … <modifier n>.

Resulting textual requirements (notional)
The system shall operate in temperature range [20, 50] deg C. Note: Interface defined in IF-2.
The system shall accept On commands according to IF-3.
The system shall provide On feedback according to IF-4.
The system shall provide On feedback in less than 2 s after receiving on command.
The system shall provide…

Figure 11. A representation of the concept to generate textual requirements out of model-based requirements

Template for Contractual Requirements in Natural Language

The basic template for a requirement takes the form of *The system shall <action> through <interface>*. This form is refined to capture the richness of requirements offered by the model-based requirements described earlier in the report. The resulting forms are shown below.

Consider the basic model provided by the sequence diagram in Figure 1 and the internal block diagram in Figure 2. Table 2 shows the template for the requirement in natural language and describes how each element of those model-based requirements is mapped to an element of such template.

Table 2. Mapping of model elements to textual template

Template of textual requirement	**Model element**
The <object> shall <accept> <Input> according to <Interface>. *Note 1*: <Input> is defined in <Source 1>. *Note 2*: <Interface> is defined in <Source 2>.	<object>: Block in diagrams referred to as *System*. <accept>: Captured as an input directional port on the system in the Sequence Diagram (incoming arrow in the sequence diagram). <Input>: Name of the *Signal* connected to the input directional port in the Sequence Diagram. <Interface>: Connection between *System* block and external block in the Internal Block Diagram, to which *Signal* is allocated. This is described as a physical port in the System block. <Source 1>: Properties of the *Signal*, directly described in the properties of the element. <Source 2>: Properties of the physical interface, directly described in the properties of the *Port* element.
The <object> shall <provide> <Output> according to <Interface>. *Note 1*: <Output> is defined in <Source 1>. *Note 2*: <Interface> is defined in <Source 2>.	<object>: Block in diagrams referred to as *System*. <provide>: Captured as an output directional port on the system in the Sequence Diagram (outgoing arrow in the sequence diagram). <Output>: Name of the *Signal* connected to the output directional port in the Sequence Diagram. <Interface>: Connection between *System* block and external block in the Internal Block Diagram, to which *Signal* is allocated. This is described as a physical port in the System block. <Source 1>: Properties of the *Signal*, directly described in the properties of the element. <Source 2>: Properties of the physical interface, directly described in the properties of the *Port* element.

Consider now the model-based requirements in Figure 5, which capture required dependencies between the inputs and outputs. As previously described, these are not exhaustive but suffice for the purpose of the report. Table 3 shows the templates for the requirement in natural language and describes how each element of model-based requirements is mapped to an element of such templates.

Table 3. Mapping of functional dependencies model elements to textual template

Template of textual requirement	**Model element**
The <object> shall <action> <when> in <condition>.	<object>: Block in diagrams referred to as *System*. <action>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside one of the branches of the conditional element is an input or an output, respectively to the block *System*. <when>: This value is used when the diagram element is *alt*. <condition>: As described in the condition property of the *alt* element.
The <object> shall <action 1> <while> <action 2>.	<object>: Block in diagrams referred to as *System*. <action 1>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside one of the branches of the conditional element is an input or an output, respectively to the block *System*. <while>: This value is used when the diagram element is *par*. <action 2>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside another branch of the conditional element is an input or an output, respectively to the block *System*.
The <object> shall <action> <while/for> <condition>.	<object>: Block in diagrams referred to as *System*. <action>: It takes the value of *accept* or *provide* depending on whether the *Signal* element inside the conditional element is an input or an output, respectively to the block *System*. <while/for>: This value is used when the diagram element is *loop*. <condition>: As described in the condition property of the *alt* element.

The model elements of required time dependencies or restrictions between inputs and outputs as modeled in Figure 6 can be mapped to elements of a requirement in natural language as shown in Table 4.

Table 4. Mapping of timing dependencies model elements to textual template

Template of textual requirement	**Model element**
The <object> shall <action 1> in <time dependency> <after> <action 2>.	<object>: Block in diagrams referred to as *System*. <action 1>: It takes the value of *accept* or *provide* depending on whether the *Signal* element is an input or an output, respectively to the block *System*. <time dependency>: This is formally defined as a range of [Min, Max], which refer to dependencies such as: less than, more than, within. <after>: This is implied by the temporal dependency given by the *duration constraint*. <action 2>: It takes the value of *receiving* or *providing* depending on whether the *Signal* element is an input or an output, respectively to the block *System*.

Two options are offered for the template for capturing simultaneity of requirement applicability in natural language (as modeled for example in Figure 7). The first one is shown in Table 5, together with a description of how each element of model-based requirements is mapped to an element of such template. The second one consists in simply creating separate sections of the requirement document for each mode requirement, with a statement that reads *All requirements in this section shall be fulfilled simultaneously*.

Table 5. Mapping of applicability simultaneity model elements to textual template

Template of textual requirement	**Model element**
<Req X>. The system shall… Note: This requirement must be fulfilled simultaneously with [<Req Y>].	<Req X> is a requirement originating from a *Sequence Diagram* linked to a *state* element. [<Req Y>] is a list of all requirements originating from all *Sequence Diagrams* linked to the *state* element to which *Sequence Diagram* from which <Req X> originates is also connected.

No template is prescribed for capturing the characteristics of inputs, outputs, and interfaces in textual form. In general, they may be listed as columns containing the property and the required values for each property. For physical interfaces, properties may be organized, for example, following a layered approach, such as identifying a transport layer and a physical layer.

Tool

A Java script connects to SysML models in Cameo Systems Modeler™ through its Application Protocol Interface (API). The script is designed to generate plain text tables from a SysML model file. Primarily, the script reads the model data and creates maps to store its information. After this step, it separates and organizes data according to the previous template in Java objects. Then, leveraging the organized data, the script creates the tables of requirements and by substituting the different objects in the template with the information stored in the Java objects. These tables are then exported to a Microsoft Word file.

Application Case

Methodology

The suitability of the model-based requirements methodology and the effectiveness of the approach to translate them into contractual requirements in natural language presented in this report have been evaluated by applying them to a case. The work consisted of four sequential activities:

Activity 1. An existing set of requirements in natural language was identified and adjusted to be used as the benchmark.

Activity 2. A set of model-based requirements that is equivalent to the benchmark requirements in natural language was created.

Activity 3. The requirements conversion tool was applied to the requirements model, yielding a set of requirements in natural language.

Activity 4. A comparative analysis between the requirements yielded by the tool and those of the benchmark was performed.

Problem Statement

Note: This section has been slightly adapted from a publication by the authors prepared, submitted, and published during the period of performance of this research (Alejandro Salado & Wach, 2019b)

An optical space instrument is considered for this case study. The purpose of the instrument is to take images of the Earth and send them to the satellite platform under command by the platform. In parallel, the instrument provides health status data *continuously* to the satellite platform for monitoring purposes. The case study is limited to capture in model-based form the requirement set listed in Table 6 in natural language. The requirement set has been adapted from (Alejandro Salado & Nilchiani, 2014) and includes new requirements that have been added to make the set coherent and with a flavor of completeness. Although the set of requirements is very limited subset of what an actual set of requirements may be, the “[a]cceptability and suitability of the sample requirements [were] validated by deriving and contrasting them against requirements of actual operational and scientific optical space systems developed by different manufacturers for different customers and with a similar level of complexity, which is represented by an instrument mass of around 1 ton" (Alejandro Salado & Nilchiani, 2014).

Table 6. Requirements for the optical instrument in natural language [adapted from (Alejandro Salado & Nilchiani, 2014)]

ID	Description
R1	The instrument shall image a target at 600 km – 650 km according to IF-1.
R2	The instrument shall image a target with spectral radiance of ABC (*plot) according to IF-1.
R3	The instrument shall accept Command A according to IF-2.
R4	The instrument shall transmit image data according to IF-2 in less than 0.2 s after receiving Command A.
R5	The instrument shall have a resolution better than 1 unit.
R6	The instrument shall have a FOV greater than 2°.
R7	The instrument shall provide telemetry data every 1 s according to IF-2.
R8	The instrument shall accept power according to IF-3.
R9	The instrument shall consume less than 600 W of electrical power.
R10	The instrument shall withstand a mechanical load of 5 g in any direction on IF-4.
R11	The instrument shall fulfill its performance when subjected to a temperature between -10 deg C and +45 deg C at IF-4.
R12	The instrument shall have a lifetime of at least 7 years.
Note 1	R10 only applies during launch. All other requirements only apply once the instrument is powered on through IF-3.

*These elements are not shown for convenience.

Tables 7 through 10 capture the interface details referred to by the requirements in Table 6. An incomplete sample list of parameters is used for convenience, but it is sufficient to showcase how such properties are captured in model-based form.

Table 7. IF-1 description

Property	**Value**
Physical layer	
Pressure	$< 3 \times 10^{-15}$

Table 8. IF-2 description

Property	**Value**
Transport layer	
Protocol	MIL-STD-1553B[1]
Data structure	**Complex definition of packet structures, etc.*
Data map: Command A	ID: 11 Messages: 110011: Send current image data 110101: Resend last image data
Data map: Image data	ID: 01 Messages: xxyyyy: xx is a time stamp and yyyy parts that form the image data
Data map: Telemetry	ID: 00 Messages: 01xxxx: No error found, followed by xxxx detailed status data 11xxxx: Critical error found, followed by xxxx detailed status data
Physical layer	
Electrical Properties	
Voltage range	[0, 5] V
Impedance	78 ohm +/2%
Conducted emissions	**A plot*
Conducted susceptibility	**A plot*
Thermal properties	
Conductivity	<= 200 W/K
Connector	
Type	D9F
Pin allocation	1: GND 2: Data+ 3: Data- …

[1]This is used for illustrative purposes. The data in the table may not be in line with the actual standard.

*These elements are not shown for convenience.

Table 9. IF-3 description

Property	**Value**
Physical layer	
Electrical Properties	
Voltage range	[22, 28] V, nominal 24V
Impedance	> 1 Mohm
Conducted emissions	**A plot*
Conducted susceptibility	**A plot*
Thermal properties	
Conductivity	<= 200 W/K
Connector	
Type	D9M
Pin allocation	1: GND 2: GND 3: Power …

*These elements are not shown for convenience.

Table 10. IF-4 description

Property	**Value**
Physical layer	
Thermal properties	
Conductivity	<= 5 W/K
Contact surface	[2.0, 2.5] cm^2
Mechanical properties	
Footprint	**Mechanical drawing*

*These elements are not shown for convenience.

Formulation Strategy

Note: This section has been slightly adapted from a publication by the authors prepared, submitted, and published during the period of performance of this research (Alejandro Salado & Wach, 2019b)

Table 11 describes the strategies or conceptualizations to capture the requirements in Table 6 in model-based form. Interface properties in Tables 7 through 10 will be captured as properties of the physical port through which logical signals are conveyed, as has been explained previously.

Table 11. Strategies to model-based requirements

Req ID	Strategy
R1	This requirement defines an input that the system must accept. It will be modeled directly as an input signal.
R2	This requirement defines a characteristic of the required input defined in R1. It will be modeled as a property of the signal modeled to capture R1.
R3	This requirement defines an input that the system must accept. It will be modeled directly as an input signal.
R4	The requirement defines an output that the system must provide, as well as the conditions under which the output must be provided. It will be modeled as an output signal and a time dependency with the signal modeled to capture R3.
R5	This requirement defines a characteristic of the required output defined in R4. It will be modeled as a property of the signal modeled to capture R4.
R6	This requirement defines a characteristic of the required output defined in R4. It will be modeled as a property of the signal modeled to capture R4.
R7	This requirement defines an output that the system must provide, as well as the conditions under which the output must be provided. It will be modeled as an output signal occurring in parallel to the exchanges required by R1 through R4.
R8	This requirement defines an input that the system must accept. It will be modeled directly as an input signal. In addition, it will be modeled as a starting event that needs to occur before the exchanges required by R1, R2, R3, R4, and R7 can be executed (since the instrument must be powered in order to fulfill those requirements), and which remains active in parallel with the other exchanges defined in the corresponding mode of operation.
R9	This requirement defines a resource limitation that the system must fulfill, in relation to the required input defined in R8. It will be modeled as a property of the signal modeled to capture R8.
R10	This requirement defines an external environment in which the system needs to operate. It will be modeled as an input signal (mechanical energy) to the system.
R11	This requirement defines an external environment in which the system needs to operate. It will be modeled as an input signal (thermal energy) to the system.
R12	This requirement defines a constraint on how long the system needs to fulfill its requirements. It will be modeled as a duration constraint that describes for how long each transformation needs to be executed.
Note 1	This note defines modes of operation for the system, for which different sets of requirements apply. It leads to define a specific mode (launch) in which R10 applies and another set of modes in which the rest of the requirements apply, as well as the transitions between the modes.

Model-Based Requirements

Note: This section has been slightly adapted from a publication by the authors prepared, submitted, and published during the period of performance of this research (Alejandro Salado & Wach, 2019b)

Collectively, the models represented in Figures 12 through 19 capture all requirements listed in Table 6 without using *shall* statements. The sets of requirements that need to be fulfilled simultaneously can be modeled as shown in Figure 12. Following the strategy in Table 11, two modes are defined (sets of model-based requirements), *Launch* and *Nominal operations*. The model-based requirements that need to be fulfilled in each mode are linked to each corresponding mode. Furthermore, the model-based requirement *Mode transition* is created to indicate the conditions under which each set of requirements (i.e., mode) becomes applicable. These conditions are depicted in Figure 13. As can be seen, a strict order is defined, meaning that the *Nominal operations* requirements set will only need to be fulfilled after the *Launch* requirements set has been fulfilled. Such transition in applicability of requirements is determined by a condition on the state of the acceleration inputted into the system. In this regard, it is important to note that Figure 13 should not be understood as the system transitioning from one state to another and acceleration acting as a trigger. Instead, the model-based requirement *Mode Transition* captures the conditions that make one set of requirements (mode) applicable over the other one.

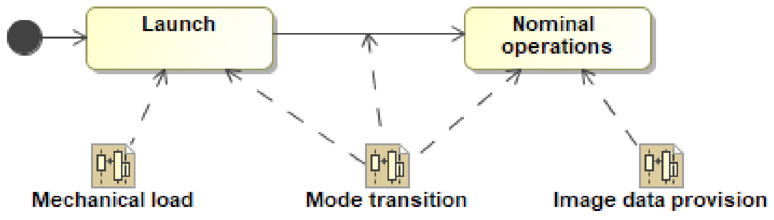


Figure 12. Mode requirements

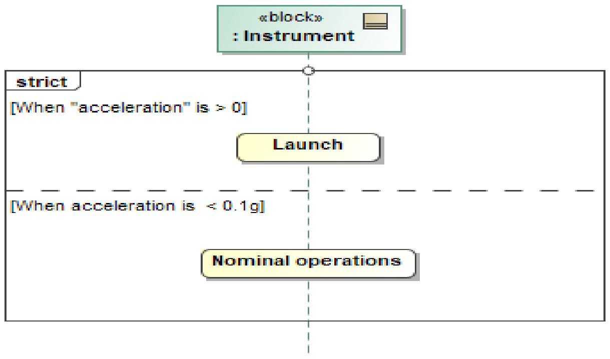


Figure 13. Conditions for applicability of each subset of requirements
(Mode transition in Figure 12).

The set of requirements applicable in *Launch* is modeled as shown in Figure 14. The set of requirements applicable in *Nominal operations* is modeled as shown in Figure 15. Note that these two refer only to the required exchanges, which are complemented by Figures 16 to 19, depicting required signal properties, required allocation to interfaces, and required interface properties.

As described in Table 11, the mechanical load requirement is modeled as an energy input to the instrument. The input is characterized as a continuous flow (not a one shot) with the value of the minimum acceleration that the instrument will receive (ref. Figure 16). The diagram in Figure 14 does not show any output. However, this does not imply that the system is not executing any transformation. The full set of requirements needs to be evaluated when making such judgement. In this case, the transformation occurs. It is captured by including Figure 13 and the requirements modeled in Figure 15. The system will execute several outputs (as modeled in Figure 15, which will be described later) after it has received the input in Figure 14, as indicated by the strict sequence in Figure 13.

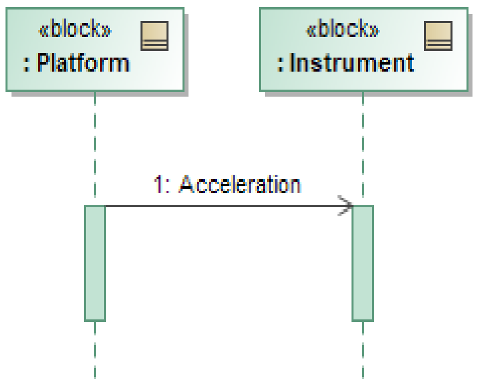


Figure 14. Exchange related to the mechanical load requirement

Several aspects in Figure 15 are worth discussing with respect to the *Nominal operations* requirement set. First, there is a need to leverage the *parallel* interaction operator because the requirements in the set need to be fulfilled simultaneously. Hence, the diagram represents that the instrument will:

1. continuously be exposed to the incoming light from the Earth's surface, during which
2. the instrument will receive a continuous stream of electrical power and thermal energy, and
3. will receive commands and be expected to provide image data, and
4. is expected to provide telemetry data periodically.

Second, the *loop* operation operator is used to capture the lifetime requirements. It indicates that the modeled exchange needs to be executed for 7 years at least, as stated in Table 11. Third, the temporal nature of different inputs and outputs are captured through properties in the *signal* elements. Figure 16 shows how they can be used to capture one-off signals (such a trigger), continuous flows (such as energy flux), and periodic signals. In addition, it should be reminded that signal properties can also be defined by linking to *parametric diagrams*.

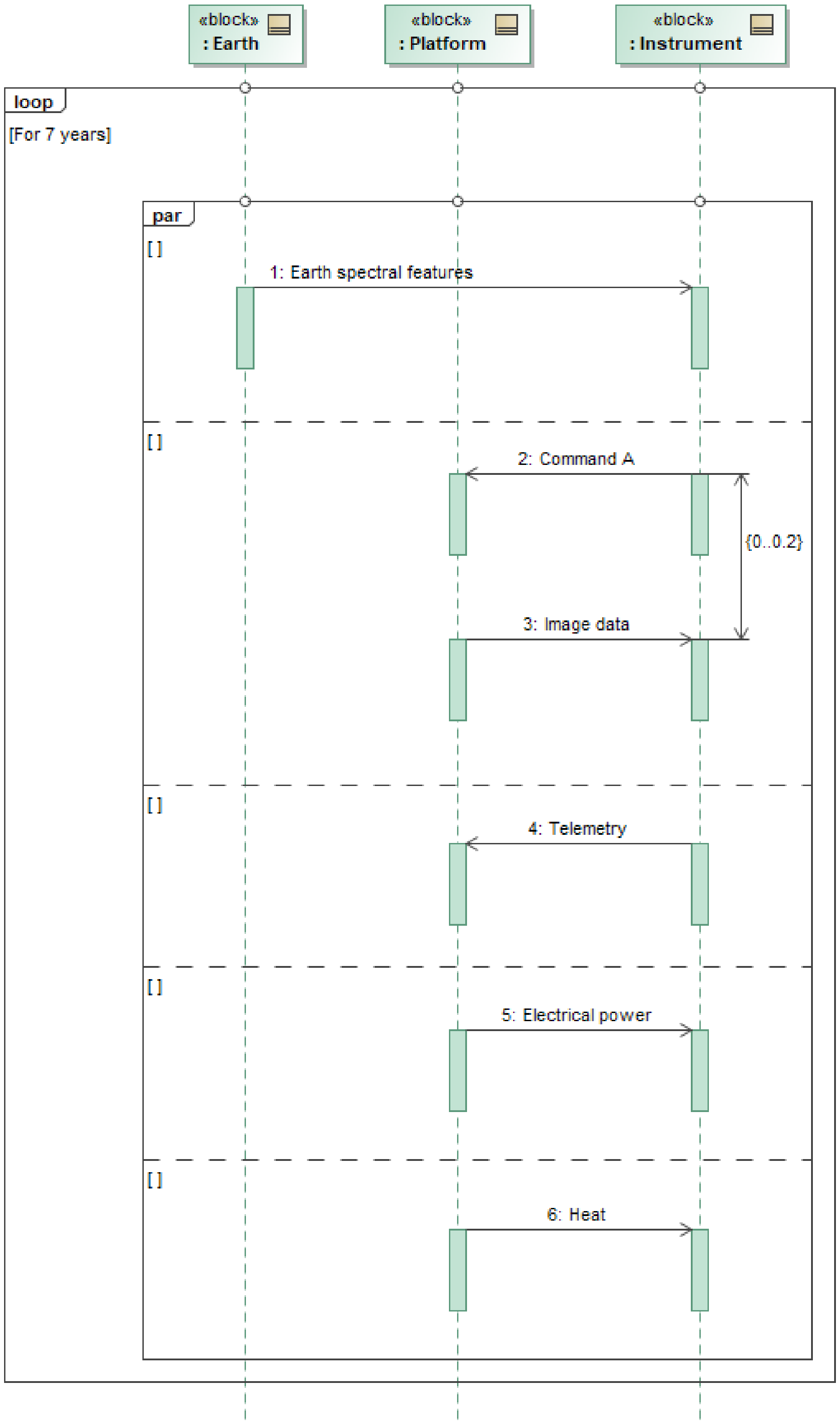


Figure 15. Required exchanges in nominal operations

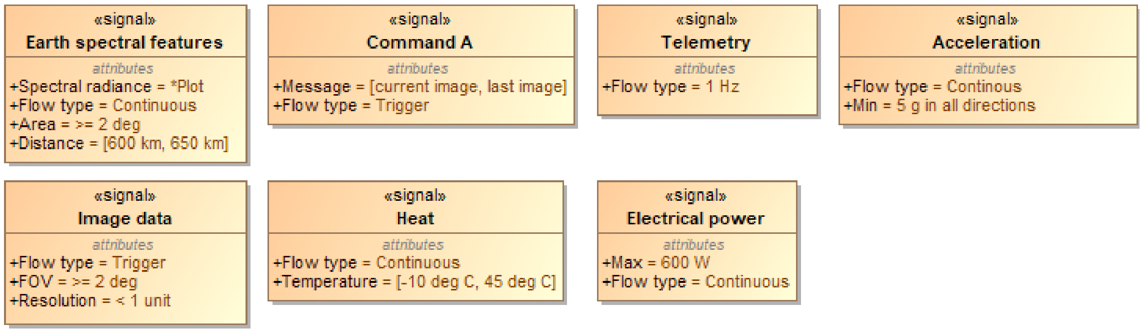


Figure 16. Required characteristics of the required inputs and outputs

The allocation of inputs and outputs into physical interfaces in Figure 17 captures the four different interfaces. Direction of arrows is consistent with the signals being inputted to or outputted by the instrument, as indicated by the sequence diagrams in Figures 14 and 15.

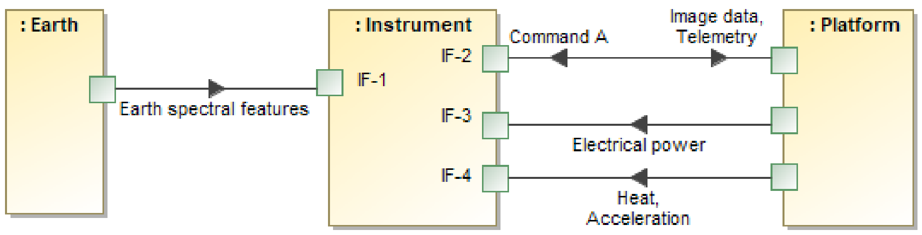


Figure 17. Requirements on the allocation of logical inputs and outputs to physical interfaces through which they must be conveyed

The properties of those interfaces are shown in Figures 18 and 19. Pinout allocation and some aspects of the transport layer have been modeled using proxy ports to avoiding model complexity. In this example, specific meanings of data messages are captured separately and the linked to the physical interfaces (ref. Figure 19). Finally, as it was the case for signals, properties could also be linked to *parametric diagrams*.

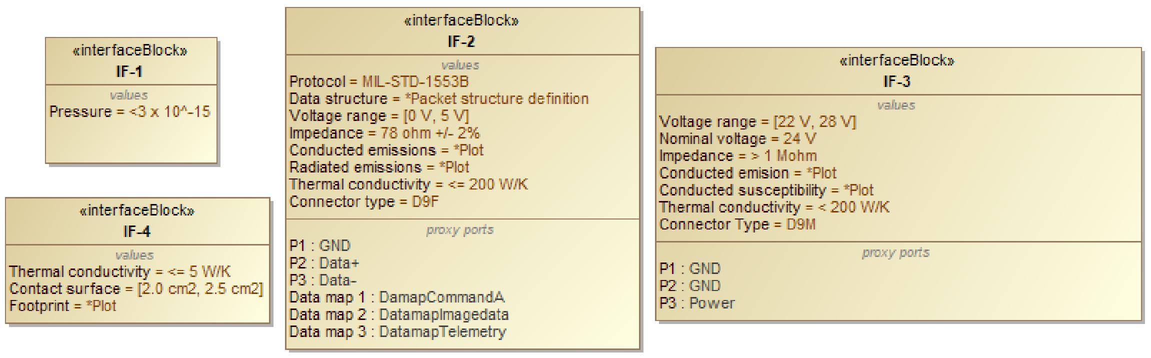

«interfaceBlock»
IF-1
values
Pressure = <3 x 10^-15
«interfaceBlock»
IF-4
values
Thermal conductivity = <= 5 W/K
Contact surface = [2.0 cm2, 2.5 cm2]
Footprint = *Plot
«interfaceBlock»
IF-2
values
Protocol = MIL-STD-1553B
Data structure = *Packet structure definition
Voltage range = [0 V, 5 V]
Impedance = 78 ohm +/- 2%
Conducted emissions = *Plot
Radiated emissions = *Plot
Thermal conductivity = <= 200 W/K
Connector type = D9F
proxy ports
P1 : GND
P2 : Data+
P3 : Data-
Data map 1 : DamapCommandA
Data map 2 : DatamapImagedata
Data map 3 : DatamapTelemetry
«interfaceBlock»
IF-3
values
Voltage range = [22 V, 28 V]
Nominal voltage = 24 V
Impedance = > 1 Mohm
Conducted emision = *Plot
Conducted susceptibility = *Plot
Thermal conductivity = < 200 W/K
Connector Type = D9M
proxy ports
P1 : GND
P2 : GND
P3 : Power

Figure 18. Required characteristics of the physical interfaces through which inputs and outputs must be conveyed

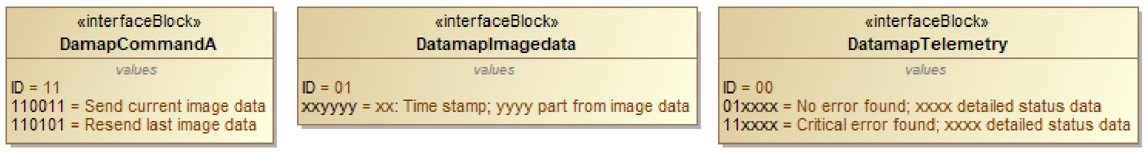

«interfaceBlock»
DamapCommandA
values
ID = 11
110011 = Send current image data
110101 = Resend last image data
«interfaceBlock»
DatamapImagedata
values
ID = 01
xxyyyy = xx: Time stamp; yyyy part from image data
«interfaceBlock»
DatamapTelemetry
values
ID = 00
01xxxx = No error found; xxxx detailed status data
11xxxx = Critical error found; xxxx detailed status data

Figure 19. Modeling of transport layer aspects as proxy ports for leveraging model complexity

Automatically Generated Textual Requirements

The tool described previously was applied to the model presented in the previous section. Tables 12 to 14 list the resulting requirements in natural language.

Table 12. Requirements extracted form the requirements models

ID	Requirement
R1	The system shall provide Acceleration according to IF-4. Note 1: Acceleration is defined in Table 13. Note 2: IF-4 is defined in Table 14.
R2	The system shall provide Earth spectral features according to IF-1. Note 1: Earth spectral features is defined in Table 13. Note 2: IF-1 is defined in Table 14.
R3	The system shall provide Command A according to IF-2. Note 1: Command A is defined in Table 13. Note 2: IF-2 is defined in Table 14.
R4	The system shall accept Image data according to IF-2. Note 1: Image data is defined in Table 13. Note 2: IF-2 is defined in Table 14.
R5	The system shall accept Telemetry according to IF-2. Note 1: Telemetry is defined in Table 13. Note 2: IF-2 is defined in Table 14.
R6	The system shall provide Electrical power according to IF-3. Note 1: Electrical power is defined in Table 13. Note 2: IF-3 is defined in Table 14.
R7	The system shall provide Heat according to IF-4. Note 1: Heat is defined in Table 13. Note 2: IF-4 is defined in Table 14.
R8	The system shall provide Image data in less than 0.2s after having received Command A.
R9	The system shall <all actions> For 7 years.
R10	The system shall accept [Earth spectral features, Command A, Electrical power, Heat] while providing [Image data, Telemetry].

Table 13. Required characteristics of required inputs and outputs

Property	**Value**
S1: Earth spectral features	
Spectral radiance	*Plot
Flow type	Continuous
Area	>= 2 deg
Distance	[600 km, 650 km]
S2: Command A	
Message	[current image, last image]
Flow type	Trigger
S3: Image data	
Flow type	Trigger
FOV	>= 2 deg
Resolution	< 1 unit
S4: Telemetry	
Flow type	1 Hz
S5: Heat	
Flow type	Continuous
Temperature	[-10 deg C, 45 deg C]
S6: Electrical power	
Max	600 W
Flow type	Continuous
S7: Acceleration	
Flow type	Continous
Min	5 g in all directions

Table 14. Required properties of required interfaces

Property	Value
IF-1	
Pressure	<3 x 10^-15
IF-2	
Protocol	MIL-STD-1553B
Data structure	*Packet structure definition
Voltage range	[0 V, 5 V]
Impedance	78 ohm +/- 2%
Conducted emissions	*Plot
Radiated emissions	*Plot
Thermal conductivity	<= 200 W/K
Connector type	D9F
P1	GND
P2	Data+
P3	Data-
Data map 1	DamapCommandA
Data map 2	Datamapimagedata
Data map 3	DatamapTelemetry
IF-3	
Voltage range	[22 V, 28 V]
Nominal voltage	24 V
Impedance	> 1 Mohm
Conducted emission	*Plot
Conducted susceptibility	*Plot
Thermal conductivity	< 200 W/K
Connector Type	D9M
P1	GND
P2	GND
P3	Power
IF-4	
Thermal conductivity	<= 5 W/K
Contact surface	[2.0 cm2, 2.5 cm2]
Footprint	*Plot

Comparative Analysis

The resulting textual requirements for the required properties of the physical interfaces captured in Figures 18 and 19, and given in Table 14, are identical to those in the benchmark given in Tables 7 to 10. Therefore, no additional discussion is necessary.

The requirements in Tables 12 and 13, corresponding to the models in Figures 12 to 17, differ at first sight from those of the benchmark, listed in Table 6. In fact, there are some differences in terms of defined solution space between the natural language requirements resulting from the model-based requirements and the benchmark requirements. Both of these differences are discussed below.

In terms of visual differences, a different approach is taken for describing the different modes. However, this is purely a stylistic matter and of no real concern for the definition of the solution space. In addition, the benchmark employed a single requirement for each required property of the required system inputs and outputs, whereas the resulting set in this paper employs a table form for the properties linked to a single requirement for each input and output. We believe that both options have pros and cons with respect to requirement management. For example, the benchmark option may be easier to manage in terms of traceability in requirements management tools. However, it does not present any structure to facilitate consistency during requirement derivation. Certainly, there may be ways to overcome both problems with both approaches. Hence, these differences remain aesthetic and with no impact on the definition of the solution space. Therefore, both the benchmark and the contractual requirements derived from the model- based requirements can be considered equivalent.

Wording employed in the textual statements is also different. The free form employed in the benchmark yields the use of verbs that provide a description of the intent or purpose expected to be fulfilled by the system, whereas the proposed template uses only on accepting/providing statements. It is suggested that the proposed approach in this report is actually more effective. This assertion is based on two aspects. First, the purpose of deriving stakeholder needs into system requirements is to devoid the requirements of context, so that only what the system has to do is defined; not what an external actor will do with the actions of the system. In this sense, and using systems theory, a system can be fully characterized by the inputs it accepts from the environment and external systems and the output it provides to them. No other type of action verb is necessary. Second, natural language lends itself towards diversity of interpretation. This difference can cause a difference in the content of the solution space, as different engineers work towards finding an acceptable solution. Therefore, limiting the types of actions that the system can take, as proposed in this paper, may be beneficial to cope with such limitation of natural language.

In terms of effects on the solution space beyond wording interpretation, the only apparent difference is that the benchmark did not explicitly refer to the need to execute certain actions in parallel, while the models did. This difference may just be an artifact of the limitations of the case study. However, some aspects like parallelization are sometimes not explicitly captured in the requirements but interpreted by the engineer from the set of requirements. The former does not have any impact in this analysis. The latter would actually be a potential risk for requirements derivation, since the interpretation of the engineer may not be consistent with the actual need. Therefore, in the best case, both sets of requirements can be considered equivalent from this perspective. In the worst-case, the proposed approach yields a more accurate and complete definition of the solution space.

Finally, it should be noted that the transition requirements captured in Figure 13 have not been transformed to textual requirements. The reason is that the model-based requirements were incomplete, as per the scope of the research project, and did not capture the external conditions for the different mode requirements as external inputs (particularly, pressure conditions), but just as operational conditions of the transitions. However, it is important to note that such requirements were also not part of the benchmark requirements in Table 6. Therefore, the same discussion about incompleteness of the benchmark and potential interpretation of the engineer made in the previous paragraph applies to this case as well.

Recommendations and Future Directions

This research has provided two main results. First, it is possible to construct models of requirements that are consistent with systems theory by extending existing SysML constructs. This contribution is critical to advance practice towards a model-based acquisition paradigm. Underlying model constructs with theory guarantees consistency in the definition of solution spaces using models. Furthermore, SysML has been widely adopted by the engineering community, which allows for an easier entrance and adoption of the proposed model-based requirements.

Second, the models of the requirements can be used to derive shall statements, which are useful for contracting activities in acquisition. Although a future where models become contractually binding can be envisaged, a transition mechanism needs to be implemented. The mathematical structure of the proposed model-based requirements guarantees the necessary level of precision and consistency to derive requirements in natural language that accurately represent the intended problem scoping captured with model-based requirements. By automating the conversion process directly in the MBSE tool, an engineer can produce and deliver a complete set of requirements in natural language without ever having to write a shall statement.

Therefore, the research presented in this report bridges modeling and contracting through models and automated conversion processes. This research has shown strong indication that the resulting requirements are at least as good as the current approaches to write requirements in natural language. Future work is necessary to more strongly evaluate the contractual safety of model-based requirements by gathering evidence of their precision, accuracy, and potential for completeness compared to those achieved by textual requirements.

A note of caution is also necessary. Current research is exploring the use of hybrid approaches for requirements, where textual requirements are augmented with or supported by images of system models. This research shows that such approaches, at least as currently explored, should be discouraged in front of the proposed approach. Existing hybrid approaches present two potential risks. First, two different theoretical frameworks and requirement constructs are joined, without an understanding of the inconsistencies and gaps they may yield. Second, and most importantly, most (if not all) existing SysML models are aimed to model a system, not a solution space. However, requirements should yield solution spaces and not systems. Consequently, hybrid approaches may unconsciously limit the solution space unnecessarily, potentially discarding acceptable solutions.

Conclusions

This report has presented an approach to capture requirements in the form of models and derive contractual requirements in natural language directly from the model, without the engineer ever having to write shall statements. This approach can enable a technical team to transition to model-based requirements, while guaranteeing fulfilling the expectation of contractual departments and acquisition programs. The former can work directly in developing models, while the latter can still provide *shall* statements to vendors and suppliers.

The approach builds on an internally consistent theoretical framework, which guarantees avoiding formal flaws in terms of bounding solution space. The central proposition of the approach is that every requirement can be modeled as an input/output transformation. As a result, the approach itself forces an engineer to define requirements on the boundaries of the system and inherently avoids over-constraining the solution space unnecessarily. Specifically, SysML has been leveraged and extended as follows:

(1) An extended sequence diagram captures the required logical transformation.

(2) Signals capture logical inputs and outputs with their required attributes.

(3) Ports in block elements capture the physical interfaces and their required properties through which inputs and outputs are conveyed.

(4) An extended state machine diagram is used to capture mode requirements, which capture the simultaneity aspects of requirements applicability.

In addition, modeling requirements as transformations of inputs into outputs implies that it is not meaningful to distinguish between functional and non-functional requirements. This conceptualization, which departs from existing literature, facilitates consistency. Examples have been provided to show how attributes traditionally allocated to a system and thought of as non-functional (e.g., mass or color) are actually attributes of the inputs and outputs of the system. Furthermore, this work has shown that the proposed approach to model requirements as transformations can improve precision over using natural language.

This precision is maintained when converting the models into requirements in natural language by leveraging a predefined template. The template follows existing guidelines for writing good requirements. The basic structure of a requirement follows the pattern, *The system shall accept/provide <signal> through <interface> based on <conditions>*. This structure limits the type of actions that the system may use, guaranteeing an efficient usage of terms, improving precision and accuracy of the resulting requirement set.

The approach has been implemented as a Java script application that leverages SysML models in Cameo System Modeler™. The application has been used to test the proposed approach on a case study. The case of development of an optical instrument for a satellite available in literature has been employed. Results indicate that the proposed approach is at least as effective in defining solution spaces (capturing requirements) as traditional requirement writing approaches based on natural language. No significant differences were found between the textual requirements of the benchmark and those derived from the requirements models. Future work is however necessary to measure precisely the level of completeness, accuracy, and precision that model-based requirements can yield.

The research has already resulted in one published paper for the 2019 Acquisition Research Symposium, one published paper for the 2019 Conference on Systems Engineering Research (CSER), and one published paper in the Systems journal. Several other conference and journal papers resulting from this research are currently under preparation and will be submitted before the end of 2019.

References

Aceituna, D., Do, H., Walia, G. S., & Lee, S. (2011). *Evaluating the use of model-based requirements verification method: A feasibility study*. Paper presented at the Workshop on Empirical Requirements Engineering (EmpiRE 2011), Trento.

Aceituna, D., Walia, G., Do, H., & Lee, S.-W. (2014). Model-based requirements verification method: Conclusions from two controlled experiments. *Information and Software Technology, 56*(3), 321-334. doi:https://doi.org/10.1016/j.infsof.2013.11.004

Badreddin, O., Sturm, A., & Lethbridge, T. C. (2014). *Requirement traceability: A model-based approach*. Paper presented at the 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), Karlskrona. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6890829&isnumber=6890816

Borgne, A. L., Belloir, N., Bruel, J., & Nguyen, T. (2016). *Formal Requirements Engineering for Smart Industries: Toward a Model-Based Graphical Language*. Paper presented at the 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), Toulouse, France. http://ieeexplore.ieee.org.ezproxy.lib.vt.edu/stamp/stamp.jsp?tp=&arnumber=7816956&isnumber=7816796

Buede, D. M. (2009). *The engineering design of systems: Models and methods*: Wiley.

Cardei, I., Fonoage, M., & Shankar, R. (2008). *Model Based Requirements Specification and Validation for Component Architectures*. Paper presented at the 2008 2nd Annual IEEE Systems Conference, Montreal, Que. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4519001&isnumber=4518971

Dada, D. (2006). The failure of E-government in developing countries: A literature review. *Electroninc Journal of Information Systems in Developing Countries, 26*(7), 1-10.

El Eman, K., & Birk, A. (2000). Validating the ISO/IEC 15504 measure of software requirements analysis process capability. *IEEE Transactions in Software Engineering, 26*(6), 541-566.

F. Wanderley, A. Silva, Araujo, J., & Silveira, D. S. (2014). *SnapMind: A framework to support consistency and validation of model-based requirements in agile development*. Paper presented at the 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), Karlskrona.

Fockel, M., & Holtmann, J. (2014). *A requirements engineering methodology combining models and controlled natural language*. Paper presented at the 2014 IEEE 4th International Model-Driven Requirements Engineering Workshop (MoDRE), Karlskrona. http://ieeexplore.ieee.org.ezproxy.lib.vt.edu/stamp/stamp.jsp?tp=&arnumber=6890827&isnumber=6890816

Friedenthal, S., Moore, A., & Steiner, R. (2015). *A Practical Guide to SysML - The Systems Modeling Language* (3rd ed.). Waltham, MA, USA: Morgan Kaufman.

H. Reza, R. S., J. Straub and N. Alexander. (2017). *Toward model-based requirement engineering tool support*. Paper presented at the 2017 IEEE Aerospace Conference, Big Sky, MT, USA. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7943647&isnumber=7943554

Holder, K., Zech, A., Ramsaier, M., Stetter, R., Niedermeier, H.-P., Rudolph, S., & Till, M. (2017). Model-Based Requirements Management in Gear Systems Design Based On Graph-Based Design Languages. *Applied Sciences, 7*(11), 1112. doi:10.3390/app7111112

Holt, J., Perry, S., Brownsword, M., Cancila, D., Hallerstede, S., & Hansen, F. O. (2012). *Model-based requirements engineering for system of systems*. Paper presented at the 2012 7th International Conference on System of Systems Engineering (SoSE), Genova, Italy. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6384145&isnumber=6355890

Holt, J., Perry, S., Payne, R., Bryans, J., Hallerstede, S., & Hansen, F. O. (2015). A Model-Based Approach for Requirements Engineering for Systems of Systems. *IEEE Systems Journal, 9*(1), 252-262. doi:10.1109/JSYST.2014.2312051

Holt, J., Perry, S. A., & Brownsword, M. (2011). *Model-Based Requirements Engineering*: IET.

INCOSE. (2012). *Guide for writing requirements*. Retrieved from

INCOSE. (2015). *Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities* (version 4.0 ed.). Hoboken, NJ, USA: John Wiley and Sons, Inc.

J. Holt, S. P., M. Brownsword, D. Cancila, S. Hallerstede and F. O. Hansen. (2012). *Model-based requirements engineering for system of systems*. Paper presented at the 2012 7th International Conference on System of Systems Engineering (SoSE), Genova, Italy. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6384145&isnumber=6355890

J. Holt, S. P., R. Payne, J. Bryans, S. Hallerstede and F. O. Hansen. (2015). A Model-Based Approach for Requirements Engineering for Systems of Systems. *IEEE Systems Journal, 9*(1), 252-262. doi:10.1109/JSYST.2014.2312051

Kossiakoff, A., Sweet, W. N., Seymour, S. J., & Biemer, S. M. (2011). *Systems Engineering Principles and Practice* (2nd ed.). Hoboken, NJ, USA: John Wiley & Sons, Inc.

Lu, C., Chang, C., Chu, W. C., Cheng, Y., & Chang, H. (2008). *A Requirement Tool to Support Model-Based Requirement Engineering*. Paper presented at the 2008 32nd Annual IEEE International Computer Software and Applications Conference, Turku. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4591654&isnumber=4591503

M. Saadatmand, A. C. a. M. S. (2012). *Toward Model-Based Trade-off Analysis of Non-functional Requirements, software engineering; Unified Modeling Language; tradeoff analysis; nonfunctional requirements; generic approach; NFR; embedded systems; real-time system; UML profiling method; software development process; Unified modeling language; Analytical models; Software;Security; Batteries; Real-time systems; Time factors; Non-Functional Requirements; Trade-off Analysis; Non-Functional Properties; UML; MBD*. Paper presented at the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications, Cesme, Izmir. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6328142&isnumber=6328119

Marschall, F., & Schoemnakers, M. (2003). *Towards model-based requirements engineering for Web-enabled B2B applications*. Paper presented at the 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2003. Proceedings, Huntsville, Alabama, USA. http://ieeexplore.ieee.org.ezproxy.lib.vt.edu/stamp/stamp.jsp?tp=&arnumber=1194813&isnumber=26873

McConnell, S. (2001). From the editor - An ounce of prevention. *IEEE Software, 18*(3), 5-7.

Micouin, P. (2008). Toward a property based requirements theory: System requirements structured as a semilattice. *Systems Engineering, 11*(2).

Miotto, B. L. a. P. (2014). *Model-based requirement generation*. Paper presented at the 2014 IEEE Aerospace Conference, Big Sky, MT.

Mordecai, Y., & Dori, D. (2017). *Model-based requirements engineering: Architecting for system requirements with stakeholders in mind*. Paper presented at the 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, Austria. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8088273&isnumber=8088244

Ribeiro, F. G. C., Pereira, C.E., Rettberg, A. et al. (2018). Model-based requirements specification of real-time systems with UML, SysML and MARTE. *Software & Systems Modeling, 17*(1), 343-361.

S. Siegl, K. H. a. R. G. (2010). *Model Based Requirements Analysis and Testing of Automotive Systems with Timed Usage Models*. Paper presented at the 2010 18th IEEE International Requirements Engineering Conference, Sydney, NSW. http://ieeexplore.ieee.org.ezproxy.lib.vt.edu/stamp/stamp.jsp?tp=&arnumber=5636645&isnumber=5636534

S. Teufl, D. M. a. D. R. (2013). *MIRA: A Tooling-Framework to Experiment with Model-Based Requirements Engineering*. Paper presented at the 2013 21st IEEE International Requirements Engineering Conference (RE), Rio de Janeiro.

Salado, A., & Nilchiani, R. (2014). A Categorization Model of Requirements Based on Max-Neef's Model of Human Needs. *Systems Engineering, 17*(3), 348-360. doi:10.1002/sys.21274

Salado, A., & Nilchiani, R. (2017). Reducing Excess Requirements Through Orthogonal Categorizations During Problem Formulation: Results of a Factorial Experiment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 47*(3), 405-415. doi:10.1109/TSMC.2015.2502953

Salado, A., Nilchiani, R., & Verma, D. (2017). A contribution to the scientific foundations of systems engineering: Solution spaces and requirements. *Journal of Systems Science and Systems Engineering, 26*(5), 549-589. doi:10.1007/s11518-016-5315-3

Salado, A., & Wach, P. (2019a). *Automatic Generation of Contractual Requirements from MBSE Artifacts.* Paper presented at the Sixteenth Annual Acquisition Research Symposium, Monterey, CA, USA.

Salado, A., & Wach, P. (2019b). Constructing True Model-Based Requirements in SysML. *Systems, 7*(2), 19.

Schmitz, D., Nissen, H. W., Jarke, M., & Rose, T. (2010). *Relating domain model based requirements management and situational method engineering*. Paper presented at the 2010 Third International Workshop on Managing Requirements Knowledge, Sydney, NSW.

von Bertalanffy, L. (1969). *General Systems Theory - Foundations, development, applications*. New York, NY: George Braziller, Inc.

Wach, P., & Salado, A. (2019). *Can Wymore's Mathematical Framework Underspin SysML? An Initial Investigation of State Machines*. Paper presented at the Conference on Systems Engineering Research (CSER), Washington, DC, USA.

Wymore, A. W. (1993). *Model-based systems engineering*. Boca Raton, FL: CRC Press.

Yeo, K. T. (2002). Critical failure factors in information system projects. *International Journal of Project Management, 20*(3), 241-246.