Safety Patterns for SysML: What Does OMG Specify?

Nan Niu[1(✉)], Logan Johnson[1], and Christopher Diltz[2]

[1] University of Cincinnati, Cincinnati, OH 45221, USA
nan.niu@uc.edu, johns6lo@mail.uc.edu
[2] Edaptive Computing, Inc., Dayton, OH 45458, USA
c.diltz@edaptive.com

Abstract. The Systems Modeling Language (SysML) represents a significant and increasing segment of industrial support for building critical systems. The Object Management Group (OMG) has been releasing and revising the formal specification of SysML since 2007, with version 1.6 recently formalized in November 2019. However, little is known about what OMG specifies and how the official specification influences model-driven engineering (MDE). To fill the gap, we present a new way of analyzing the OMG SysML specification (version 1.6) to uncover reusable guidelines and constraints for safe MDE practice. We illustrate our approach with the discovery of the recurring "Asset Leakage" safety pattern and the development of a semantic-role-based theory to support practitioners' identification, formulation, and verification of critical properties in their modeling contexts.

Keywords: Systems Modeling Language (SysML) · Systems reuse · Specification patterns · Temporal constraints · Semantic roles · Grounded theory

1 Introduction

The Systems Modeling Language (SysML), first adopted by the Object Management Group (OMG) in 2006, is a general-purpose, visual modeling language for systems engineering [32]. It builds on UML as its foundation and provides additional extensions to facilitate the communication and collaboration among various stakeholders who participate in the model-driven engineering (MDE) activities. SysML is designed to equip MDE practitioners with simple but powerful constructs for modeling a wide range of problems in different application domains, including aerospace, automotive, energy, healthcare, manufacturing, and telecommunications.

In safety-critical domains, crucial properties—such as "a vehicle's revolutions per minute (RPM) shall never exceed 4,000"—must be checked. The MDE

S. Ben Sassi et al. (Eds.): ICSR 2020, LNCS 12541, pp. 19–34, 2020.
https://doi.org/10.1007/978-3-030-64694-3_2

literature distinguishes two modes of checking: offline verification and online monitoring. In offline settings, techniques like model checking [11] are employed to formally examine whether the models satisfy a given property, e.g., the LTL formula "[]!(RPM > 4000)" expresses the aforementioned safety requirement. In contrast, online monitoring deploys techniques like observer automata [31] to detect property violations for models at runtime [6]. A key difference is that runtime monitoring focuses on the current run of the deployed models, whereas model checking analyzes all the runs [8].

Researchers have addressed reusability and scalability issues in order for these property-checking mechanisms to be readily adopted in practice. For instance, Dou *et al.* [18] proposed a model-driven approach to generating diagnostic information for temporal properties, and further used synthesized data with millions of events to show that the approach grew linearly with respect to the size of the events logging the execution of a system. Besnard and colleagues [8] developed the observer automata in the same language as the models (i.e., by using UML), and their simulation on a STM32 discovery board showed that the runtime overhead of the monitoring based on observer automata was 6.5% for the embedded target.

Despite these advances, an inherent challenge facing MDE practitioners is to specify the critical properties, e.g., those related to safety, security, and dependability [39]. Not only is expertise required to identify important concerns in a specific domain, but the concerns also have to be formulated in ways amenable to the particular machinery (e.g., TemPsy temporal formulas [18] or UML state invariants [8]). The engineers are left with many questions: which properties to begin with, how to assess the validity of the properties, and realistically speaking, how others specify properties in their work.

In their seminal work, Dwyer and his colleagues [19] surveyed 555 temporal logic specifications and showed that an overwhelming majority (92%) fell into eight highly reusable patterns: Response, Universality, Absence, etc. While we review Dwyer *et al.*'s work in more detail in the next section, their patterns have had extensive influence in software and systems engineering: querying model histories [17], developing a UML interpreter [7], debugging declarative models [26], discovering latent behavior of UML models [20], model-based testing from UML/OCL [15], to name a few. However, like Dwyer *et al.*'s patterns, the MDE extensions stay mainly at a syntactic level. Take "[]!(RPM > 4000)" as an example, although it is an instance of the globally-scoped Absence pattern ([](!P)) [19], the modeler has to semantically map P without much guidance.

If the RPM case seems too straightforward, consider the requirement of a cruise control system: "When the system is engaged, the cruise speed should be defined [8]." Here, should P in [](!P) be instantiated with "systemEngaged & unknownCruiseSpeed", or with "systemEngaged → unknownCruiseSpeed"? Or should a completely different pattern, namely Response ([](Q →<>R)) [19], be applied where Q ="systemEngaged" and R="! unknownCruiseSpeed"? Unfortunately, syntactic patterns offer little help.

In this paper, we propose to reduce the semantic gap by taking a fresh look at what OMG specifies. In particular, we manually analyze the OMG specification [33] to search for recurring and reusable patterns that guide SysML-based software and systems engineering practices. We pay special attention to the parts of the specification where the integrity of SysML is discussed. We therefore call our results *safety patterns* to indicate the risk or danger of violating them. We further codify the *semantic roles* of each pattern to ease the mapping of syntactic structures in the modeler's MDE contexts.

The contributions of our work lie in the analysis of the OMG SysML specification as a new way to guide MDE practices. Our work is particularly valuable for the practitioners who are required or recommended to adhere to OMG specifications, and our results on safety patterns offer concrete insights into the kind of critical properties subject to be verified in all SysML models. The remainder of the paper is organized as follows: Sect. 2 provides background information and discusses related work, Sect. 3 presents our research methodology for analyzing the OMG SysML specification, Sect. 4 describes our results by detailing the "Asset Leakage" pattern, Sect. 5 elaborates our vision about semantics-enriched support for MDE and systems reuse, and finally, Sect. 6 concludes the paper.

2 Background and Related Work

2.1 Property Patterns in MDE

The seminal work by Dwyer *et al.* [19] tackled the challenge concerning practitioners' unfamiliarity with temporal logic notations. They developed a pattern-based approach to ease the specification of informal requirements into temporal logic formulas. Eight reusable patterns were reported in [19]: Absence, Bounded Existence, Chain Precedence, Chain Response, Existence, Precedence, Response, and Universality. Figure 1 organizes these patterns based on occurrence and order. A survey of 555 specifications from over 35 sources showed that 92% were instances of the eight patterns shown in Fig. 1, with the top three accounted for 80% of the sample: Response ($\frac{245}{555} = 44\%$), Universality ($\frac{119}{555} = 21\%$), and Absence ($\frac{85}{555} = 15\%$) [19]. The results provided empirical evidence of human specifiers' use and reuse of common formalisms to express behavioral aspects of their subject systems.

Inspired by these patterns, MDE extensions are made. An important task of offline checking is trace diagnostics, i.e., providing the modeler with the relevant information in case a property fails to hold. When checking the logged events of a system's execution, some tools pinpoint the last log entry (i.e., the last event) read before detecting the property violation. However, the usefulness of the traces truncated in this manner is limited, because a property can be violated in different ways and the last read event may not necessarily be the event responsible for the violation. To provide relevant diagnostic information, Dou *et al.* [18] developed algorithms and their development was guided directly by Dwyer *et al.*'s work [19]. As shown in the left column of Table 1, six patterns were presented to classify the violation of a property: two on occurrence and four on

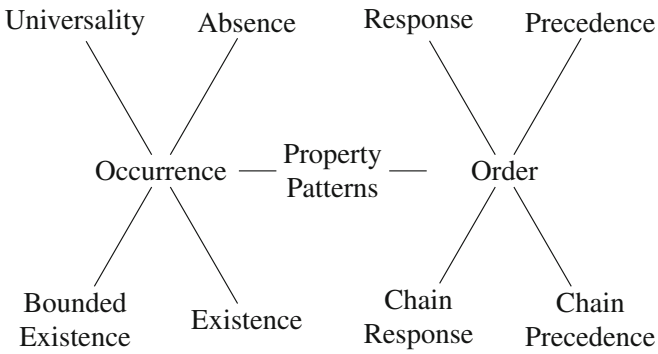

Fig. 1. Temporal property patterns by Dwyer *et al.* [19] (*figure adopted from* [17]).

Table 1. Illustrating the MDE Extensions of Dwyer *et al.*'s Reusable Patterns [19]

Property violations [18]	Model histories [17] (# of OCL operations)
(1) unexpected occurrence	(i) basic version traversal (7)
(2) no-show occurrence	(ii) temporal assertions (5)
(3) no-show order	(iii) predicate-based version scoping (3)
(4) wrong temporal order	
(5) wrong temporal chain	(iv) context-based version scoping (2)
(6) wrong temporal order and chain	(v) version unscoping (1)

order [18]. The trace diagnostics algorithms were then informed by the patterns, two of which (diagnostic information of Existence and that of Precedence) were explained in detail in [18].

Another extension was made by García-Domínguez *et al.* [17] to allow a model's histories to be queried online. Five groups of querying [17] are listed in the right column of Table 1. In each group, one or more OCL operations are defined to instrument the temporal assertions. For example, to check model x's property p within predicate-based version scoping, three operations are added: one for the versions since p ($x.since(v \mid p)$, $x.after(v \mid p)$), the second until p ($x.until(v \mid p)$, $x.before(v \mid p)$), and the third with the matching of p ($x.when(v \mid p)$). García-Domínguez *et al.* [17] showed that all the order patterns and all the five scopes (i.e., globally, before Q, after R, between R and Q, and after R until Q) by Dwyer *et al.* [19] were mapped to one or more of the 18 OCL operations.

The patterns of property violation [18] and time-aware querying [17] illustrate the MDE extensions of Dwyer *et al.*'s work [19]. These patterns offer much syntactic help, as the chief intent is to assist practitioners in understanding the scope of temporal logic modalities encapsulated in each pattern's generalized description of permissible state/event sequences [19]. Having only syntactic sup-

port can be limited, e.g., the main query of the remote data mirroring case study in [17] was a composition of ten OCL primitives and two OCL operations involving four syntactic structures, i.e., (i), (iii), (iv), and (v) of Table 1. Semantic support bridging the syntactic structures and the practical modeling concerns can be complementary and lead to enhanced benefit. Next we describe the major source from which we derive the semantic knowledge.

2.2 OMG SysML Specification

The OMG SysML specification, like all other OMG specifications (e.g., UML and CORBA), addresses vertical (application domain independent) practices to promote interoperability, portability, and reusability. Because SysML reuses a subset of UML, the specification facilitates systems engineers modeling with SysML and software engineers modeling with UML to be able to collaborate on models of software-intensive systems [3]. In addition, OMG specifies the language so that modeling tool vendors can implement and support SysML [5, 40].

Being a standards consortium, OMG has an open process allowing all the specifications to undergo continuous review and improvement. There is no exception for the SysML specification [33]. Version 1.0 of the formal specification was adopted in September 2007. Since then, OMG has made six revisions, each taking an average of 24.3 months to be released. The most recent version—released in November 2019—is the formal SysML specification version 1.6 [34] which we use in our study. Throughout this paper, we simply refer to [34] as the "OMG SysML specification".

This specification is a 398-page PDF document containing 17 sections and 7 annexes [34]. The main contents can be divided into general principles (e.g., conformance and language formalisms), model elements (covering both structural and behavioral constructs), and crosscutting topics (e.g., extending metamodel via stereotypes). While the contents of such an international standard are expected to evolve in an incremental and stable fashion, we next present the strategies for analyzing the OMG SysML specification.

3 Research Methodology

Our goal is to discover useful and reusable pattern-oriented knowledge from the OMG SysML specification so as to ease the MDE practitioners' identification and formulation of critical properties. To that end, we develop a research methodology based on grounded theory [37]. Figure 2 overviews the process of our approach. Our underlying research question is to explore commonly occurring guidelines and constraints from the OMG SysML specification. We rely on the specification descriptions, but also go beyond the syntactic layer to uncover semantic patterns to inform the MDE practices. While our approach of Fig. 2 takes the OMG SysML specification as the input, the output is a theory providing new ways of formulating critical properties in the modeling contexts that

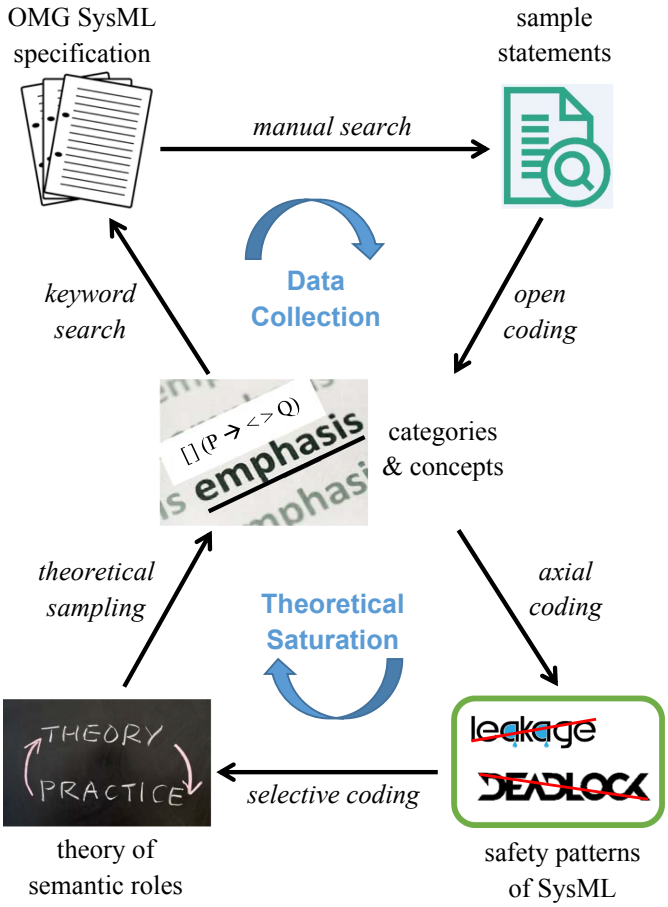

Fig. 2. Analyzing the OMG SysML specification informed by grounded theory.

can be reused across different domains (e.g., automotive, energy, manufacturing, etc.).

The process of Fig. 2 consists of two cycles. The first cycle is concerned with data collection. Initially, we manually search the OMG SysML specification for the statements that express rules of thumb or impose limiting powers in modeling. For example, one of the statement samples is:

> "*. . . control can only enable actions to start. SysML extends control to support disabling of actions that are already executing. This is accomplished by providing a model library with a type for control values that are treated like data*" (§11.1.1 "Control as Data" *page 155* [34])

We then apply open coding [37] to explicitly annotate the categories and concepts from the sample statements. For the above example, the category "Activities"

(as a diagram element) and the concepts "disabling" and "executing" are coded. The open coding results are fed back to the specification to enrich the data collection. Here, keyword search based on "disabling", "executing", and their lexical variants like "disable" and "disables" is performed, and more focused manual search of §11 on "Activities" is conducted. As a result, the following statement, among others, is selected and coded:

> "*. . . when an activity invokes other activities, they can be associated by a composition association, with the invoking activity on the whole end, and the invoked activity on the part end. If an execution of an activity on the whole end is terminated, then the executions of the activities on the part end are also terminated*" (§11.3.1 "Diagram Extensions" *page 164* [34])

When no new categories or concepts are generated, our data collection comes to a fixed point and we proceed to the second cycle of Fig. 2. This cycle first involves axial coding [37] where the categories and concepts are related and possibly combined. This allows us to synthesize the core variable. In our study, for instance, an essential dependency emerges from the codes of the above two statements: "data depends on control" and "part depends on whole". Any SysML modeling that violates this class of dependencies is then unsafe. Such dependencies form what we call a safety pattern of SysML, which we further apply selective coding [37] to build a theory. In selective coding where the tentative core has already been established, we deliberately code data without bothering about concepts with little importance to the safety patterns. If necessary, selective coding leads us to go back to the OMG specification, further improving our data collection. Once our codified semantic roles of each safety pattern are able to incorporate most selective coding results, theoretical saturation is reached and our theory shall be put into use.

4 Results and Analysis

We share the analysis results by first describing our effort level, followed by a detailed account of the top safety pattern from our work ("Asset Leakage"). We then demonstrate our application of the "Asset Leakage" pattern to a SysML model, and conclude this section by discussing the threats to validity affecting our results.

Effort. Three researchers spent a total of about 40 h in analyzing the OMG SysML specification [34]. The data collection phase of our process, as shown in the top cycle of Fig. 2, involved two researchers working independently to identify sample statements and to code categories and concepts. These individual sessions totaled approximately 30 h.

A two-hour meeting was held among the three researchers to merge the collected data, perform axial coding, formulate safety patterns, and build a theory to offer semantic support for SysML-based MDE. This was followed by selective coding done individually, occasionally collecting more data from the

OMG specification. The entire theory building cycle of Fig. 2 took around 10 h in total. Although our manual analysis effort was not negligible, we expect this cost would be amortized over the application of our theory to all SysML models and potential reuse of our results in other MDE tasks (e.g., trace diagnostics [18], requirements discovery [9,21], visual analytics [29,35], obstacle and mistake analysis [2,4], and so on).

Asset Leakage. Our most prominent result is what we call the "Asset Leakage" pattern. Table 2 provides the details of this safety pattern. We express the pattern in LTL as: "[] $((p \rightarrow q)$ U $!q)$", which is interpreted as: "it is always the case that if p is running/executing, then (it is because that) q is running/executing, and this will hold until q stops running/executing". In this pattern, p can be thought of as an asset which is guarded by q, and if the specified property fails to hold, then asset leakage occurs. Although the pattern positively prevents the asset from leaking, we negatively name the pattern to alert what will go wrong if SysML is unsafely practiced. Our naming convention is in line with such terms as "segmentation fault" and "buffer overflow" [42].

We were able to identify ten instances of "Asset Leakage" from the OMG specification. Table 2 lists the ten statements resulted from our axial coding, as well as the categories and concepts of each statement. The keywords underlying the concepts show strong connections with p and/or q's creation (e.g., "start") and termination (e.g., "destroying"). In addition, the dependency between the two are important (e.g., "invokes"). Some keywords like "dependency" return many automatically searched results, which suggests weighting the keywords may be more effective to automate data collection. Some statements, such as #9 and #10 in Table 2, do not contain any keyword, which implies that manual search is indispensable.

Table 2 sorts the ten statements based on their page numbers, and the category column of the table shows that six "Asset Leakage" instances appear in the OMG specification's behavioral constructs (§11—§14), one in the structural constructs (§7—§10), one in the crosscutting constructs (§15—§17), and two in the annexes. Unsurprisingly, more than half of the statements tie directly to the behavioral aspects of SysML as "Asset Leakage" concerns more about functions and responses; however, structural integrity like #1 shall not be ignored. Although §C defines SysML elements that are deprecated, statements #9 and #10 are of relevance when modelers or tool vendors face backward compatibility issues.

As our overarching goal is to support MDE practice, we are building a theory that not only raises the modelers' awareness of SysML safety patterns, but also shields them from the complexity of the formal notations like "[] $((p \rightarrow q)$ U $!q)$". We thus develop a theory of semantic roles to guide practice and promote reuse in systems engineering. Our idea is inspired by *frame semantics* [16] arguing that one cannot understand a word's meaning without access to all the essential knowledge that relates to that word. Better understanding is gained once a frame of semantic knowledge which the word evokes is teased out.

Table 2. "Asset Leakage" safety pattern grounded in the statements of the OMG SysML specification [34]

No.	**Statement** (concepts are underline)	**Category** (enclosing section & page #)	**Instantiation of** [ ] (($p \rightarrow q$) U !q)
1	"*If the general ports had binding connectors to internal parts, then the full specialization would be invalid.*"	"Proxy and Full Ports" (§9.4.4 & page 139)	[ ] ((fullSpecialization → ! bindingConnector) U bindingConnector)
2	"*… control can only enable actions to start. SysML extends control to support disabling of actions that are already executing. This is accomplished by providing a model library with a type for control values that are treated like data*"	"Control as Data" (§11.1.1 & page 155)	[ ] ((dataActive → controlActive) U ! controlActive)
3	"*Destroying an instance of an activity terminates the corresponding execution …*"	"Diagram Extensions" (§11.3.1.1.1 & page 164)	[ ] ((activity → execution) U ! execution)
4	"*Terminating an execution also terminates the execution of any other activities that it invoked synchronously …*"	"Diagram Extensions" (§11.3.1.1.1 & page 164)	[ ] ((sub-exec → main -exec) U ! main-exec)
5	"*Composition means that destroying an instance at the whole end destroys instances at the part end.*"	"Diagram Extensions" (§11.3.1.1.1 & page 164)	[ ] ((composedPart → whole) U ! whole)
6	"*… when an activity invokes other activities, they can be associated by a composition association, with the invoking activity on the whole end, and the invoked activity on the part end. If an execution of an activity on the whole end is terminated, then the executions of the activities on the part end are also terminated*"	"Diagram Extensions" (§11.3.1.1.1 & page 164)	[ ] ((composedPart → whole) U ! whole)
7	"*if an instance of Operating Car is destroyed, terminating the execution, the executions it owns are also terminated.*"	"Usage Examples" (§11.4 & page 176)	[ ] ((sub-exec → main -exec) U ! main-exec)
8	"*When a Copy dependency exists between two requirements, the requirement text of the client requirement is a read-only copy of the requirement text of the requirement at the supplier end of the dependency.*"	"Stereotypes" (§16.3.2.2 & page 217)	[ ] ((clientReqReadOnly → supplierReqCopied) U ! supplierReqCopied)
9	"*The isConjugated attribute inherited from UML port is interpreted in the following way: … if the direction of every flow property specified in the flow specification is reversed (IN becomes OUT and vice versa). If set to True, then all the directions of the flow properties specified by the flow specification that types a nonatomic flow port are relayed in the opposite direction …*"	"FlowPort" (§C.3.2.2 & page 260)	[ ] ((nonatomicFlow-Reversed → isConjugated) U ! isConjugated)
10	"*If a flow port is not connected to an internal part, then isBehavior shall be set to true.*"	"Semantic Variation Point" (§C.3.2.3 & page 261)	[ ] ((! isBehavior → flowPortConnected) U ! flowPortConnected)

We made an initial attempt to build a semantic frame for the "Asset Leakage" pattern. Table 3 shows our results where the three "semantic roles" (or three pairs of roles) are what we believe to best guide reusable MDE practice. To understand and apply each role-pair, "key relationship" and "action trigger" provide further hints, as they are evoked by the given roles. Table 3 also links each role-pair to the OMG specification's statements listed in Table 2. We elaborate the semantic roles as follows.

Table 3. Semantic Roles of "Asset Leakage"

Semantic Roles (p–q)	Key Relationship	Action Trigger	Examples (cf. Table 2)
delegate-constituency	invoked execution	terminate	#2, #3, #4, #7
part–whole	composite binding	destruction	#5, #6, #8
value–condition	special configuration	setup	#1, #9, #10

delegate–constituency captures a dynamic relationship driven by constituency's invocation of delegate's execution, e.g., main invokes sub, or control enables data. The safety concern here is triggered by constituency's termination of the delegate, and if delegate's execution is not properly halted, then the asset leaks.
part–whole binds a composition or a client-supplier relationship. Once the whole or the supplier is destructed, the part or the provided service must follow the same course; otherwise, the binding is breached.
value–condition allows special setup and configuration to be defined. This pair of roles can be used to enforce prohibitions (#1), behaviors (#9), and default values (#10).

Demo. We illustrate how our theory of semantic roles might be applied in practice via a state machine model adopted from the cruise control system (CCS) study presented in [8]. As SysML reuses UML's state machine diagram, the model shown in Fig. 3 is syntactically sound in SysML. Figure 3 models how the cruise speed manager (CCM) and the pedals manager (PM) set, reset, increase, or decrease the cruise speed (CS).

In this safety-critical scenario, our "Asset Leakage" pattern is readily applicable. The SysML modeler can be prompted with the three pairs of semantic roles of Table 3 to identify whether any would apply to the state machine diagram. Suppose "Lock" is recognized by the modeler as the "delegate" to stabilize the CS. Then, the formulation and the model checking of the property, [] (("Lock" → "Engaged") U ! "Engaged"), can be done automatically without the modeler's input. The result—in this case, a failed model checking and a counterexample—can be presented to the modeler for further investigation. With the assistance of our semantic roles in Table 3, the modeler notices an important value-condition constraint (i.e., "Lock" is enabled upon "Pause") and adds it to the delegate-constituency relationship. The property is automatically updated to, [](("Lock" → ("Engaged" | "Pause") U ! ("Engaged" | "Pause")), and this time, model checking successfully verifies that the state machine of Fig. 3 has no "Asset Leakage".

Threats to Validity. We followed grounded theory [37] to design and execute our research, driven by the question seeking for recurring themes from the OMG SysML specification to guide safe MDE practice. A threat to construct validity hinges on our integrity-focused, and admittedly temporal-property-biased, interpretation of safety, exemplified by the "Asset Leakage" pattern. Safe SysML practice may also be explained from a more structural point of view; however, our study focuses more on the model behaviors.

We mitigate threats to internal validity by explicitly defining our overall process (cf. Fig. 2) as well as protocols to the concrete qualitative data analysis: open, axial, and selective coding. We also made sure that each phase and activity of our study were carried out jointly rather than by a single researcher. Neither of these steps removes researcher bias entirely; only replications (including the theoretical ones [23,28]) can address this issue. An important threat to external

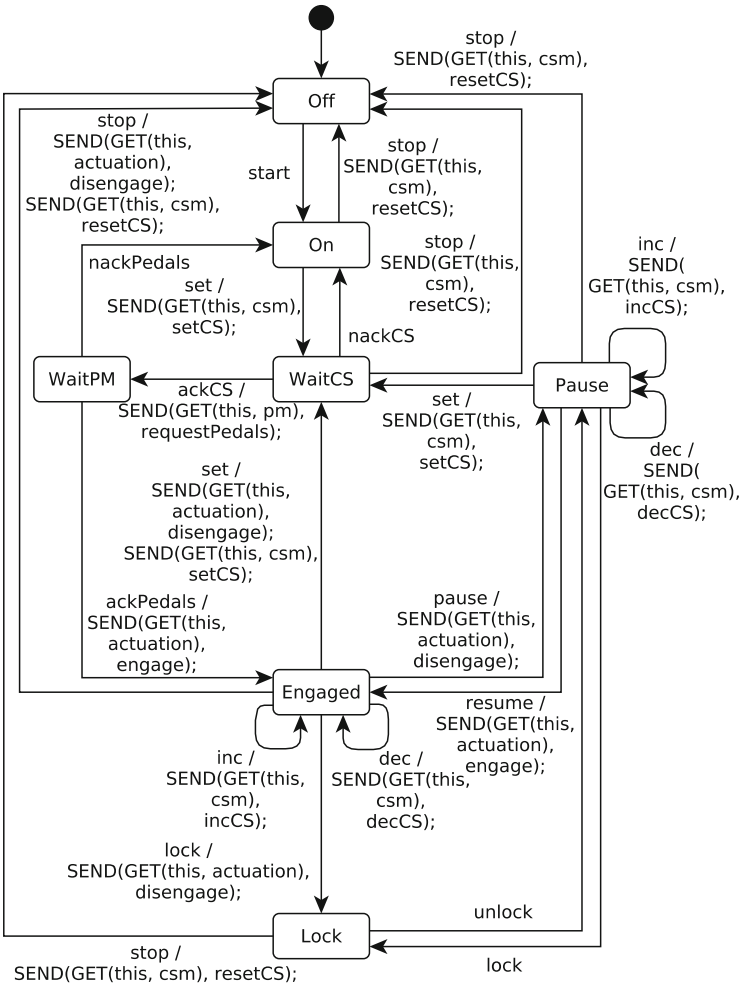

Fig. 3. State machine of the cruise control system (CCS) example (*figure adopted from* [8]).

validity is that our results may not generalize beyond the OMG SysML specification. While this is a comprehensive document, it is the only data source of our study. Therefore, it is not clear how generalizable our safety patterns and semantic roles are if additional sources are considered (e.g., the OMG UML specification).

5 Discussion

Inadequacies in the State-of-the-Art. Our work addresses two major gaps in the literature. For systems engineers, SysML has become a *de facto* choice [1, 36]. As more industries and organizations adopt SysML, rigorous MDE that is capable of handling safety, security, and other dependability concerns will be of crucial significance. As pointed out by Dou *et al.* [18]: "*. . . our industrial partner, which uses a software development methodology that requires all solutions to adhere to OMG specifications.*" To our surprise, researchers have not attempted to analyze arguably the most authoritative documentation: the OMG SysML specification [33]. Our work therefore fills a much-needed gap.

Secondly, the contemporary support for MDE practitioners to formulate critical properties mainly stays at a syntactic level. While syntactic patterns like those pioneered by Dwyer *et al.* [19] offer insights into how often the modal operators have been used, we take a step toward codifying what p and q mean in the syntactic structures. With our semantic layer of support, not only can the complexity of syntactic notations be hidden from the modelers, but the support can better relate to their modeling concerns. We thus hypothesize that semantic-enriched support like ours could shorten practitioner's cognitive distance toward critical requirements formalization thereby improving reusability. We posit this closer distance is manifested in both term acquaintance (e.g., "Asset Leakage" sounds more familiar than "Universality") and application closeness (e.g., "Asset Leakage" tends to encapsulate domain characteristics at a proper level of abstraction). We plan to build upon the recent literature [12–14] to further test our hypothesis empirically.

Pertinence and Correctness. Three strands of work help establish the pertinence of our development of a semantic-role-based theory. Liaskos *et al.* [24] applied frame semantics to identify variability from natural language documents and then to incorporate the semantically framed variability as OR-decompositions into requirements goal modeling. Niu and Easterbrook [27] used semantic frames to characterize the functional requirements in software product lines [38]. Breaux *et al.* [10] reported their experience of deriving generalizable natural language patterns to aid engineers in mapping frequently recurring requirements phrases from policy and regulatory descriptions, and showed that these patterns' reuse level was above 50% when Internet privacy policies and the U.S. HIPAA Privacy Rules were analyzed.

Correctness can be backed up by the negative results. For example, we were tempted to create semantic roles for what we call the "Deadlock" safety pattern. We had already formalized the LTL expression as: "$[]\ (\ (p\ \&\ q) \rightarrow <>\ r)$", and found a couple of instances from the OMG SysML specification. Due to this, "Deadlock" was part of our axial coding results as illustrated in Fig. 2. However, we were unable to find more instances during selective coding, leading us to put a hold on this particular tentative core at the moment. Because our theory, especially the development of our theory, is refutable, correctness of our results can be evaluated on more solid grounds.

Potential Impacts. We discuss our work's impacts from three angles. For MDE practitioners, the theory of semantic roles, exemplified by "Asset Leakage", offers practical and reusable guidance to uncover important modeling concerns without them being bogged down in the syntactic complexity or formal methods unfamiliarity. For SysML tool builders, recognizing the semantic roles and associated frames allows more effective and robust configurations to be set up, and more pertinent properties to be verified offline or monitored online. For researchers, our study illustrates grounded theory in action, encouraging more effort to understand and analyze MDE assets like the OMG specifications in a principled way.

6 Conclusions

In MDE, critical concerns such as safety and security must be ensured. SysML represents a significant and increasing segment of industrial support for building critical systems that are interdisciplinary, complex, and constantly evolving. We have presented in this paper a new way of analyzing the OMG SysML specification in order to support the identification, formulation, and verification of critical properties, which are codified in reusable safety patterns like "Asset Leakage" and further encapsulated via semantic roles, such as delegate–constituency, part–whole, and value–condition.

Our future work includes expanding the data sources to include the relevant documentation like the OMG UML specification or industry-specific standards, e.g., functional safety for road vehicles (ISO 26262 [22]). Our current safety patterns and their semantic roles are likely to be updated by the vertical or horizontal expansions. In light of the possible expansions, we are also interested in keyword weighting mechanisms for potentially speeding up the search and data collection over natural language documents [25,30,41]. Finally, we want to explore synergies of our patterns and the syntactic ones like "Bounded Existence" by Dwyer *et al.* [19]. We anticipate such synergies as: "access to the asset shall be bounded to n times without any leakage" would offer a wider range of guidance and assistance to the MDE practitioners.

Acknowledgments. We thank Raj Desai and Mounifah Alenazi from the University of Cincinnati for their assistances during data collection and analysis.

References

1. Alenazi, M., Niu, N., Savolainen, J.: A novel approach to tracing safety requirements and state-based design models. In: International Conference on Software Engineering, pp. 848–860 (2020)
2. Alenazi, M., Niu, N., Savolainen, J.: SysML modeling mistakes and their impacts on requirements. In: International Model-Driven Requirements Engineering Workshop, pp. 14–23 (2019)
3. Alenazi, M., Niu, N., Wang, W., Gupta, A.: Traceability for automated production systems: a position paper. In: International Model-Driven Requirements Engineering Workshop, pp. 51–55 (2017)

4. Alenazi, M., Niu, N., Wang, W., Savolainen, J.: Using obstacle analysis to support SysML-based model testing for cyber physical systems. In: International Model-Driven Requirements Engineering Workshop, pp. 46–55 (2018)
5. Alenazi, M., Reddy, D., Niu, N.: Assuring virtual PLC in the context of SysML models. In: International Conference on Software Reuse, pp. 121–136 (2018)
6. Bencomo, N., Götz, S., Song, H.: Models@run.time: a guided tour of the state of the art and research challenges. Software Syst. Model. **18**(5), 3049–3082 (2019)
7. Besnard, V., Brun, M., Jouault, F., Teodorov, C., Dhauss, P.: Unified LTL verification and embedded execution of UML models. In: International Conference on Model Driven Engineering Languages and Systems, pp. 112–122 (2018)
8. Besnard, V., Teodorov, C., Jouault, F., Brun, M., Dhaussy, P.: Verifying and monitoring UML models with observer automata: a transformation-free approach. In: International Conference on Model Driven Engineering Languages and Systems, pp. 161–171 (2019)
9. Bhowmik, T., Niu, N., Savolainen, J., Mahmoud, A.: Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering. Requirements Eng. **20**(3), 253–280 (2015). https://doi.org/10.1007/s00766-015-0226-2
10. Breaux, T.D., Antón, A.I., Doyle, J.: Semantic parameterization: a process for modeling domain descriptions. ACM Trans. Software Eng. Methodol. **18**(2), 5:1–5:27 (2008)
11. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2001)
12. Czepa, C., Amiri, A., Ntentos, E., Zdun, U.: Modeling compliance specifications in linear temporal logic, event processing language and property specification patterns: a controlled experiment on understandability. Software Syst. Model. **18**(6), 3331–3371 (2019)
13. Czepa, C., Zdun, U.: How understandable are pattern-based behavioral constraints for novice software designers? ACM Trans. Softw. Eng. Methodol. **28**(2), 11:1–11:38 (2019)
14. Czepa, C., Zdun, U.: On the understandability of temporal properties formalized in linear temporal logic, property specification patterns and event processing language. IEEE Trans. Software Eng. **46**(1), 100–112 (2020)
15. Dadeau, F., Fourneret, E., Bouchelaghem, A.: Temporal property patterns for model-based testing from UML/OCL. Softw. Syst. Model. **18**(2), 865–888 (2017). https://doi.org/10.1007/s10270-017-0635-4
16. Fillmore, C.J., Baker, C.F.: Frame semantics for text understanding. In: WordNet and Other Lexical Resources Workshop (2001)
17. García-Domínguez, A., Bencomo, N., Ullauri, J.M.P., Paucar, L.H.G.: Querying and annotating model histories with time-aware patterns. In: International Conference on Model Driven Engineering Languages and Systems, pp. 194–204 (2019)
18. Dou, W., Bianculli, D., Briand, L.C.: Model-driven trace diagnostics for pattern-based temporal specifications. In: International Conference on Model Driven Engineering Languages and Systems, pp. 278–288 (2018)
19. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: International Conference on Software Engineering, pp. 411–420 (1999)
20. Goldsby, H., Cheng, B.: Automatically discovering properties that specify the latent behavior of UML models. In: International Conference on Model Driven Engineering Languages and Systems, pp. 316–330 (2010)

21. Guo, J., Wang, Y., Zhang, Z., Nummenmaa, J., Niu, N.: Model-driven approach to developing domain functional requirements in software product lines. IET Software **6**(4), 391–401 (2012)
22. International Organization for Standardization. Road Vehicles – Functional Safety (ISO 26262). https://www.iso.org/standard/68383.html. Accessed October 2020
23. Khatwani, C., Jin, X., Niu, N., Koshoffer, A., Newman, L., Savolainen, J.: Advancing viewpoint merging in requirements engineering: A theoretical replication and explanatory study. Requirements Eng. **22**(3), 317–338 (2017)
24. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., Mylopoulos, J.: On goal-based variability acquisition and analysis. In: International Requirements Engineering Conference, pp. 76–85 (2006)
25. Mahmoud, A., Niu, N.: Supporting requirements to code traceability through refactoring. Requirements Eng. **19**(3), 309–329 (2013). https://doi.org/10.1007/s00766-013-0197-0
26. Montaghami, V., Rayside, D.: Pattern-based debugging of declarative models. In: International Conference on Model Driven Engineering Languages and Systems, pp. 322–327 (2015)
27. Niu, N., Easterbrook, S.: Extracting and modeling product line functional requirements. In: International Requirements Engineering Conference, pp. 155–164 (2008)
28. Niu, N., Koshoffer, A., Newman, L., Khatwani, C., Samarasinghe, C., Savolainen, J.: Advancing repeated research in requirements engineering: a theoretical replication of viewpoint merging. In: International Requirements Engineering Conference, pp. 186–195 (2016)
29. Niu, N., Reddivari, S., Chen, Z.: Keeping requirements on track via visual analytics. In: International Requirements Engineering Conference, pp. 205–214 (2013)
30. Niu, N., Wang, W., Gupta, A.: Gray links in the use of requirements traceability. In: International Symposium on Foundations of Software Engineering, pp. 384–395 (2016)
31. Ober, I., Graf, S., Ober, I.: Validating timed UML models by simulation and verification. Int. J. Softw. Tools Technol. Transfer **8**(2), 128–145 (2006)
32. Object Management Group. Systems Modeling Language (SysML). http://www.omgsysml.org. Accessed Oct 2020
33. Object Management Group. Systems Modeling Language (SysML) Specification. https://www.omg.org/spec/SysML/. Accessed Oct 2020
34. Object Management Group. Systems Modeling Language (SysML) Specification (Version 1.6). https://www.omg.org/spec/SysML/1.6/PDF. Accessed Oct 2020
35. Reddivari, S., Chen, Z., Niu, N.: ReCVisu: a tool for clustering-based visual exploration of requirements. In: International Requirements Engineering Conference, pp. 327–328 (2012)
36. Schäfer, W., Wehrheim, H.: The challenges of building advanced mechatronic systems. In: International Conference on the Future of Software Engineering, pp. 72–84 (2007)
37. Strauss, A.L., Corbin, J.: Grounded Theory in Practice. Sage Publications, Thousand Oaks (1997)
38. Vale, T., Santana de Almeida, E., Alves, V., Kulesza, U., Niu, N., de Lima, R.: Software product lines traceability: a systematic mapping study. Inf. Softw. Technol. **84**, 1–18 (2017)
39. Wang, W., Gupta, A., Niu, N., Xu, L.D., Cheng, J.-R.C., Niu, Z.: Automatically tracing dependability requirements via term-based relevance feedback. IEEE Trans. Industr. Inf. **14**(1), 342–349 (2018)

40. Wang, W., Niu, N., Alenazi, M., Xu, L.D.: In-place traceability for automated production systems: a survey of PLC and SysML tools. IEEE Trans. Industr. Inf. **15**(6), 3155–3162 (2019)
41. Wang, W., Niu, N., Liu, H., Niu, Z.: Enhancing automated requirements traceability by resolving polysemy. In: International Requirements Engineering Conference, pp. 40–51 (2018)
42. Westland, T., Niu, N., Jha, R., Kapp, D., Kebede, T.: Relating the empirical foundations of attack generation and vulnerability discovery. In: International Conference on Information Reuse and Integration, pp. 37–44 (2020)