

**INCOSE MBSE Patterns Working Group Report:
Semantic Technologies for Systems Engineering (ST4SE) Project**

Version 1.10.6

Oct 31, 2022



INCOSE MBSE Patterns Working Group Report--ST4SE Project

Contents

Project Team and Acknowledgements	3
<u>In a Nutshell: ST4SE--What Problem Are We Solving, and How? What Value to Me?</u>	4
1 Report Purpose, Scope, Intended Readership	5
2 Background and Pre-Requisite Knowledge	5
2.1 INCOSE MBSE Patterns Working Group and the Value of Model-Based Patterns	5
2.2 Basics of Semantic Technologies, Ecosystem Models, S*Models, and S*Patterns	6
2.3 Beyond the Basics: How to Learn More	13
3 Summary of the Project	16
3.1 Consistency Management as a Paradigm for Engineering and Life Cycle Management	16
3.2 The Interface Pattern; Use in the Traveler Power Converter Pattern	18
3.3 Enhanced Generation of a Model Consistent with a Trusted Pattern	18
3.4 Teams, Scale, and Trust: Implications in the Larger Innovation Ecosystem Environment	26
4 Tooling and Technologies Utilized	27
4.1 Semantic Technologies Applied, Placed in the Larger Information Ecosystem	27
4.2 Modeling Languages and Representations Utilized	28
4.3 Modeling, Semantic, and Pattern Configuring Tools Utilized	30
5 Project Results Demonstrated	34
5.1 Enhanced Generation of a Model from a Trusted Pattern	34
5.2 Enhanced Checking of a Model Against the Same Trusted Pattern	37
5.3 Gaining Access to the Project's Tooling and Information	41
6 Observations, Conclusions, and Implications for Action	42
6.1 Observations During the Project	42
6.2 Project Conclusions	45
6.3 Explore and Gain from this Project: Suggested Incremental Actions You Can Take	45
6.4 Additional Questions for Future Work	45
7 Engaging with the S*Patterns Community	46
8 Definitions	46
9 References	48
10 Document Change History	49

Semantic Technologies for Systems Engineering (ST4SE):

A Project of the INCOSE MBSE Patterns Working Group

Project Execution, Report Authoring, and Advisory Team:

Mark Blackburn, Stevens Institute of Technology

Ken Cureton, U. of Southern California

Hans Peter de Koning, European Space Agency (ESA)

Steve Jenkins, NASA JPL

Stephen Lewis, ICTT System Sciences

Bill Schindel, ICTT System Sciences

Acknowledgements

Report Review Special Thanks To:

Joe Marvin, Prime Solutions Group

Kathryn Trase, Ball Aerospace

Dariusz Walter, BAE Systems Australia

Previous Phase Interface Pattern Project Team:

Frank Salvatore, SAIC

Jason Sherey, ICTT System Sciences

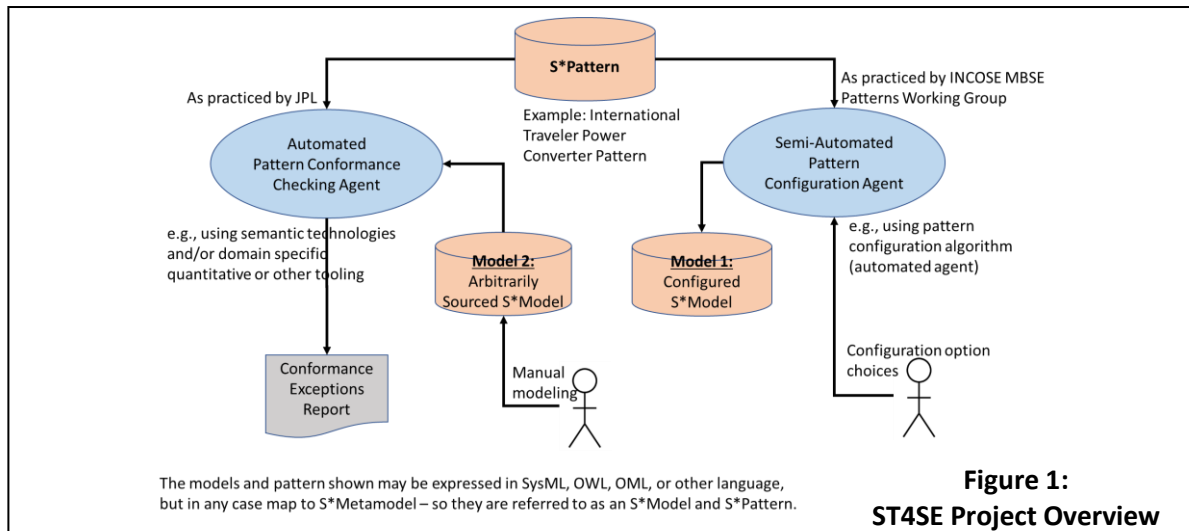
Jonathan Torok, US Navy

(Affiliations listed above were at the time of activities referenced.)

In a Nutshell: The ST4SE Project: What Problem Are We Solving, and How? What Value to Me?

The primary purpose of the Semantic Technologies for Systems Engineering (ST4SE) Project is to demonstrate:

1. the integrated capability to perform two Model-Based Systems Engineering (MBSE) tasks of value;
2. with both tasks making use of the same reusable, configurable MBSE “Pattern”;
3. Task 1: generate a specific MBSE model (Model 1) that is a configuration of that MBSE Pattern;
4. Task 2: inspect a separately provided MBSE model (Model 2) for conformance to that MBSE Pattern;
5. with both tasks performed by a systems engineer aided by contemporary modeling languages and tooling, bridging “semantic technologies” and MBSE. See Figure 1.



Why is this of value?

- A. Generating models that conform to a pre-existing pattern (for a product line, an architectural framework, an ontology, a standard, etc.) can improve the efficiency, quality, consistency, and completeness of new models, and provide a pattern-based point of accumulation (through incremental updates) of organizational learning to impact future models.
- B. Checking models for consistency against a pre-existing pattern addresses those same values, for models that were not generated by method (A), or which are modified versions of models generated by method (A).
- C. If the same pattern is used for both tasks, the resulting “loop closure” further increases value.

This project can be of value to your understanding of how to accomplish similar tasks, advancing your technical skills, your team’s performance, and reducing time model usability while improving quality.

This project has been performed under a Technical Project Plan of the International Council on Systems Engineering (INCOSE) MBSE Patterns Working Group. INCOSE has a long history of providing its membership with technical publications for advancing the practice of systems engineering, but these publications have ordinarily been in the form of traditional “document-style” publications. A secondary purpose of this project has been for INCOSE to gain experience in the publication and distribution of model-based information, using contemporary systems and methods that apply to models. A portion of this project involved configurable patterns for system interfaces, based on an earlier Patterns Working Group Interface Patterns Project.

The following sections explain this project, including necessary background, in greater detail.

1. Report Purpose, Scope, Intended Readership

In support of the above project purposes, this report explains the INCOSE Patterns Working Group ST4SE Project conducted during 2018-2021. That includes the motivation for the project, its technical approach, resulting observations and conclusions, how to access the technical deliverables of the project to allow it to be repeated with variation, and implied future work.

This report is meant to be used along with the technical project deliverables package described in the references.

This report is not a “primer” on the underlying subject matter of Semantic Technologies such as OWL DL language, the Pellet reasoner and other Semantic Technologies, Model-Based S*Patterns, Semantic Technologies, the Innovation Ecosystem, the OMG SysML modeling language, or other technologies and methodologies used by the project. However, it does briefly introduce each of those subjects and provide references that may be consulted to learn enough to understand this project.

The intended readership for this report includes:

- (a) Practicing systems engineers who would like to learn more about, and possibly practice, the tasks, methods, and technology applications described;
- (b) Business and functional leaders, entrepreneurs, or other members of the systems community who are not hands-on practitioners but who need to gain perspective on the intended uses, values, and achievable gains, using the related technologies and approaches.

2. Background and Pre-Requisite Knowledge

This section introduces the background subject matter and pre-requisite knowledge areas necessary to understand the other sections of this report, for the purposes indicated. Each of the following background sections very briefly introduces the purpose and nature of each subject area, then cites items in the References which may be used to learn more.

2.1. INCOSE MBSE Patterns Working Group and the Value of Model-Based Patterns

The INCOSE MBSE Patterns Working Group [Reference <https://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:patterns>] is a component of the INCOSE/OMG Model-Based Systems Engineering (MBSE) Initiative [Reference <http://www.omgwiki.org/MBSE/doku.php>].

As used here, “System Patterns” are configurable, re-usable System Models that would otherwise be like those expected and found in the practice of MBSE (not limited to, but including, OMG SysML models). Through the availability and use of System Patterns, the outcomes targeted by MBSE models are made more accessible, in terms of ease (and skill) of generation and use, associated modeling cost, schedule, risk, completeness, consistency, and quality. Over time, updated System Patterns become points of accumulation of organizational learning and expertise. Because they are configurable and re-usable models of families or classes of systems, model-based System Patterns involve some additional methods and disciplines that extend the ideas of MBSE (e.g., Pattern Management, Configuration Rules, model minimality, etc.).

This model-based PBSE approach has been in use for a number of years, applied across enterprises and domains that include mil/aerospace, communications, automotive, medical/health care, advanced manufacturing, consumer products, along with business processes including sales, engineering, production, and general innovation. The first INCOSE PBSE tutorial was provided at IS2005.

For detailed background on the INCOSE Patterns Working Group and the value of model-based patterns, refer to Section 2.5.

2.2. Basics of Semantic Technologies, Ecosystem Models, S*Models, and S*Patterns

2.2.1. Semantic Technologies Basics

Semantics refers to the meaning of information, within or relative to some domain of discourse (that is, for the range of valid statements within or about that domain). While data may be collected for a given domain for many business or engineering purposes, it is only after such data is interpreted and given meaning (by humans or automated agents) that decision-making can be made enhanced by that data. Thus, associating our data with the intended meaning or semantics facilitates interpretation and decision-making consistency and precision. In the case of semantic technology, that “meaning” is strictly limited to the formal descriptions provided to the technology, and it is only within that formal description that the notion of “meaning” applies.

An early example of information technology based on semantics would be the mid-Twentieth Century automated generation and subsequent processing of airline flight and seat reservations. Semantics in this domain include the meaning of concepts of Air Carrier, Flight Number, Flight Origin, Flight Destination, Departure and Arrival Time and Date, Passenger Name, Ticket Price, Available and Reserved Seats, along with meaningful relationships between these, as well as values of related parameters. A “closed” view of such a formally encoded domain is that the meaning of the “things” in this domain are fully described by the valid set of relationships between those things. Additionally, nothing else can be “said” about those things that are relevant to the domain of interest except through those relationships. It is possible to “say” (assert) and “ask” (inquire, query) many things within that domain of discourse in order to share information or support decision making. However, it is not possible to describe a recipe for cooking a chocolate cake using those same semantics, because such descriptions would come from a different domain with different semantics. Formally defined relational database technologies such as ANSI Standard SQL, Entity-Relationship Modeling, and Relational Database Management System (RDBMS) platforms were early semantic technologies used in various domains to automate a range of business, technical, and personal domain specific information interactions across enterprises, government, institutions, and personal life—within the limits of those technologies.

With the rise of the World Wide Web, linked information across a vastly greater range of domains became more universally accessible—initially for interactions with humans. Initially, those links were not given formal semantic meaning, with reasoning left to the humans navigating the web. To increase capabilities for using information that included automated reasoning agents interacting with the data of the World Wide Web, the standards-making work of the World Wide Web Consortium (W3C) established standards for an extended vision for information enabling more automated reasoning. Called the Semantic Web, this well-specified vision included more structured semantic relationships linking data according to formalized domain descriptions. This included formalization of Web Ontology Language (OWL) for the description of the semantics of ontologies characterizing various domains. It also included the notion of “Triple Stores” of three-part “information atoms,” which describe the nature of the relationship between some subject relative to another

object. For some, reference to “semantic technologies” particularly refers to the technologies of the Semantic Web, including OWL and other languages, triple-stores, graph databases, model authoring tools, and automated query and reasoning agents.

Efforts to establish formal ontologies, semantic frameworks, model-based patterns, and schema have also grown up in other environments, so that Semantic Technologies may be understood to include but extend beyond those technologies associated with the Semantic Web. To learn more about Semantic Technologies, consult the references cited in Section 2.3.

2.2.2. S*Model Basics

This project involved use of S*Models, expressed in OMG SysML and W3C OWL modeling languages. (Refer also to Section 8 Definitions.)

S*Models are models (may be MBSE models, but not required to be) in any modeling language and tooling (may be SysML, but not required to be) which satisfy a minimum model content that is described by the S*Metamodel. That reference framework sets forth a “smallest possible model” lower bound on semantics that has emerged over generations of practice as a minimal content required for the purposes of engineering or science, performed over the life cycle of a system. “Small” here also implies more general (abstract, simple).

Mapped to whatever modeling language and tooling are in use, S*Models focus on modeled content that includes: Functional Interactions; the Functional Roles engaged in those interactions; the selectable Stakeholder Features that express selectable stakeholder value, measures of effectiveness or optimality, trade space, impacts of risk, or configurability; the system States; Input-Outputs and Interfaces; system Requirement Statements; Design Components; Failure Modes; Attribute Couplings (parametric couplings) and other key modeled elements.

An informal representation of a key subset of the formal S*Metamodel (Section 2.4) is shown in Figure 2.

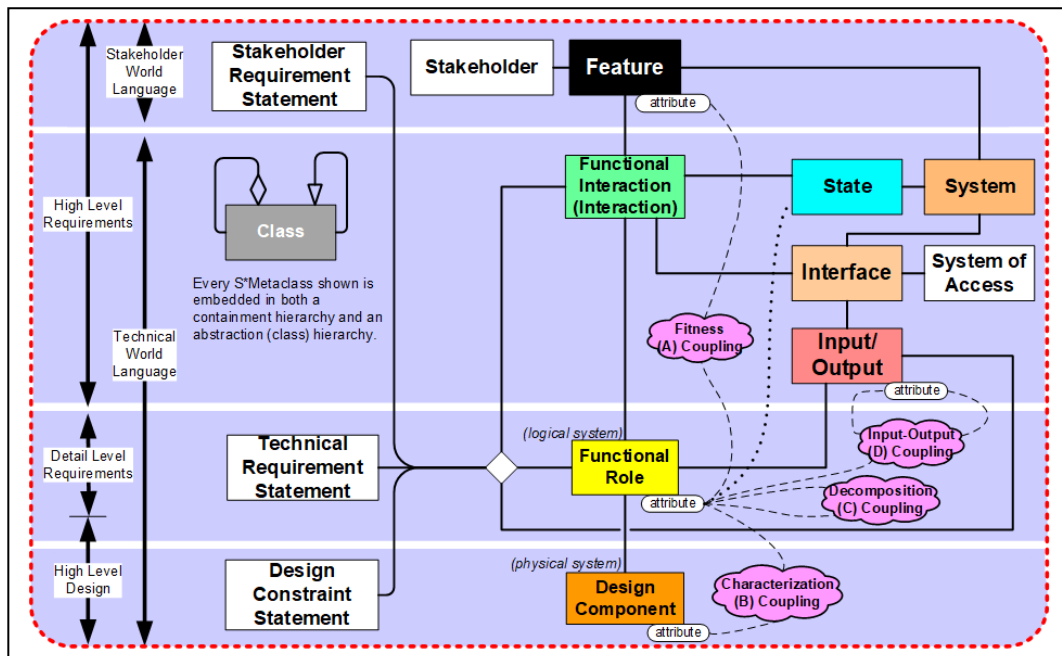


Figure 2: Informal Representation of a Subset of the S*Metamodel

(See Section 2.4 for Formal S*Metamodel)

The example S*Model illustrated in this project is an example MBSE model of an international traveler’s electrical power converter, used by the traveler in different countries with different public utility electrical standards, and the traveler’s personal electrically-powered devices. See Figure 3 and Table 1.

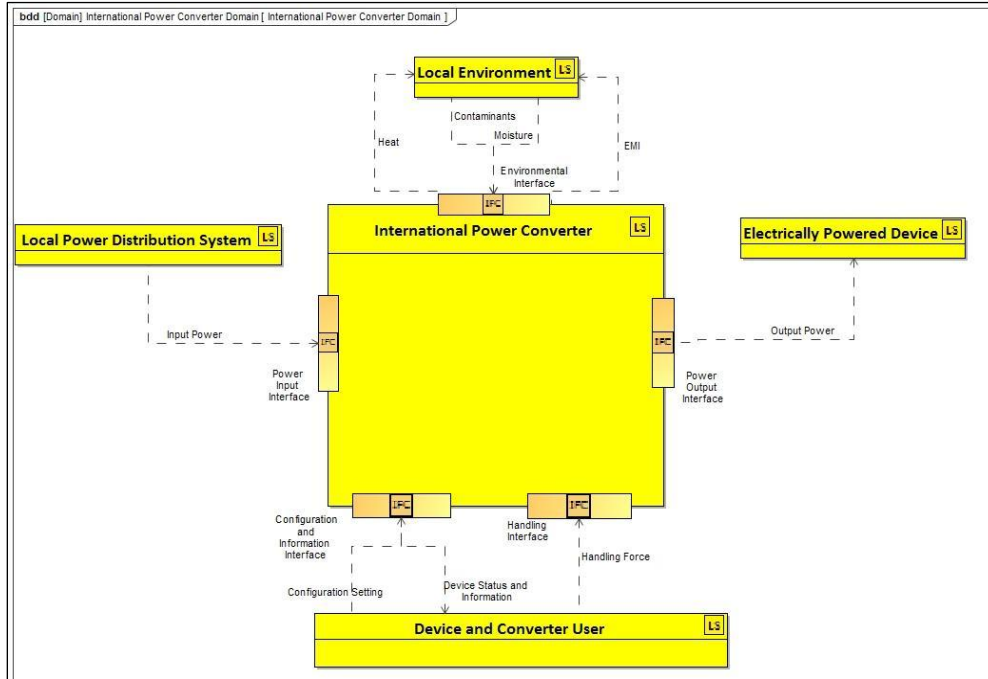


Figure 3: International Power Converter Domain Diagram / System Context Diagram

Table 1: Sampling of S*Model Elements of International Power Converter Model

S*Metamodel Element Type	S*Model Element	S*Metamodel Element Type	S*Model Element
Features	Power Mains Compatibility	Systems, Functional Roles	International Power Converter
	Powered Devices Compatibility		Local Power Distribution System
	Ease of Use		Electrically Powered Device
	Portability		Device and Converter User
	Reliability and Durability		Local Environment
	Safety		Input Power
Functional Interactions	Consume Electrical Power	Output Power	
	Convert Electrical Power	Heat	
	Set Configuration	EMI	
	Resist Contamination	Contaminants	
	Tolerate Moisture	Moisture	
	Display Status and Information	Configuration Setting	
Interfaces	Power Input Interface	Device Status and Information	
	Power Output Interface	Handling Force	
	Handling Interface		
	Configuration and Information Interface		
	Environmental Interface		

To learn more about S*Models, refer to the resources listed in Section 2.4.

2.2.3. S*Pattern Basics

This project involved use of S*Patterns, expressed in OMG SysML and W3C OWL modeling languages.

S*Patterns are reusable, configurable S*Models (see the above section) that describe system families or classes of systems. S*Patterns are therefore parameterized, general S*Models that include variability or degrees of freedom; those variable aspects include the ability to populate or depopulate (including multiply populate) various components of the S*Pattern, subject to configuration rules expressed as part of the pattern, and the ability to set the value of Attributes (parameters, properties, variables) provided as part of the S*Pattern, subject to configuration rules of the S*Pattern.

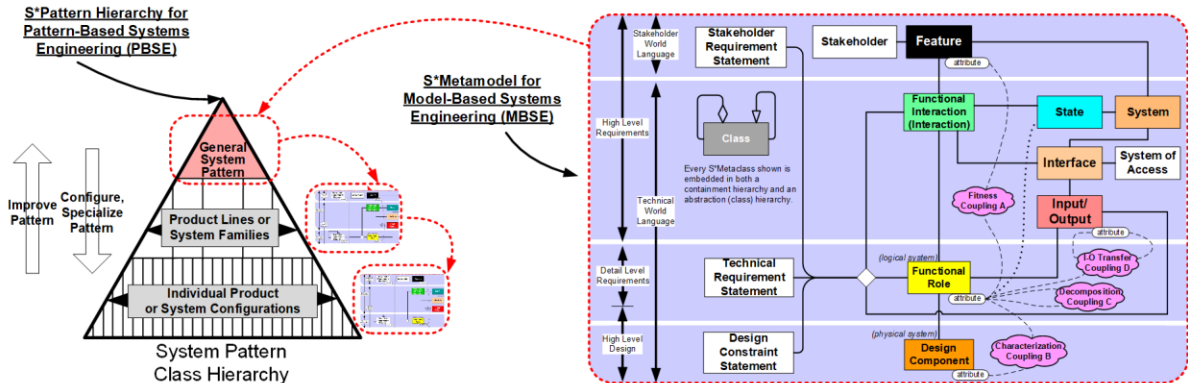


Figure 4: S*Patterns Permit Generation of Configured S*Models

The use of S*Patterns in an engineering or other environment of innovation includes the practice of accumulating learned experience in the form of S*Patterns. Those patterns may then be “configured” (populated) for specific applications or projects, rapidly yielding a first draft configured model instead of requiring “from scratch” modeling. For example, a generalized lawnmower system pattern may be configured to rapidly produce a specific case model of requirements, design, or other aspects of specific riding lawnmower, walk-behind lawnmower, or autonomous lawnmower.

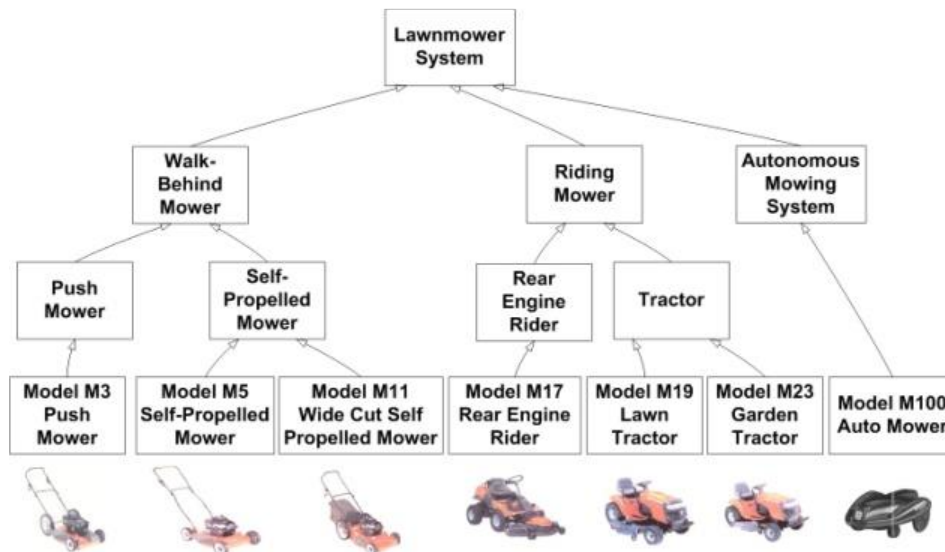


Figure 5: Configurable Systems--Product Line Example

Based on the above, an S*Pattern is effectively a model of a family of differently configured systems. The S*Pattern includes model-based configuration rules about generation of configured S*Models from it. This configuration process may be automated so that a stakeholder representative chooses desired configurations of the Stakeholder Features, and the rest of the S*Model is generated from that input and the S*Pattern's effective model-based "IF-THEN" configuration rules. In this project, that automation was provided by the Configuration Wizard supplement to the MBSE modeling tool, described further in Sections 4.3.3 and 5.1. During this generation process, a "cascade" through the S*Pattern uses already configured elements to generate additional configured elements, as shown in Figure 6:

	POPULATED METACLASSES ("THEN")																								
	Feature	Interaction	Role	Design Component	Requirement Statement	State	Event	Transition	Interface	Architectural Relationship	Input/Output	Port	System of Access	Failure Impact	Counter Requirement Statement	Failure Mode	Feature Attribute	Role Attribute	Design Component Attribute	Input/Output Attribute	Fitness Attribute Coupling	Decomposition Attribute Coupling	Characterization Attribute Coupling	IO Attribute Coupling	
TRIGGERING METACLASSES ("IF")																									
Stakeholder Input	■																								
Feature		■																							
Interaction			■																						
Role				■																					
Design Component					■																				
Requirement Statement						■																			
State							■																		
Event								■																	
Transition									■																
Interface										■															
Architectural Relationship											■														
Input/Output												■													
Port													■												
System of Access														■											
Failure Impact															■										
Counter Requirement Statement																■									
Failure Mode																	■								
Feature Attribute																		■							
Role Attribute																			■						
Design Component Attribute																				■					
Input/Output Attribute																					■				
Fitness Attribute Coupling																						■			
Decomposition Attribute Coupling																							■		
Characterization Attribute Coupling																								■	
IO Attribute Coupling																									■

Figure 6: A Pattern-Specified Cascade Populates a Configured S*Model from Features

To learn more about S*Patterns, refer to the resources listed in Section 2.5.

2.2.4. Ecosystem Model Basics

Because of the explicit "learning" or accumulative IP (Intellectual Property, intellectual assets, etc.) nature of S*Patterns (see the Section 2.2.3) when using pattern-based methods the overall business or ecosystem model shifts to something different than what is seen in many traditional "project-oriented" environments. In those traditional cases, it is more likely to see models created "from scratch", from personal experience, or from "desk drawer files", to satisfy a work procedure that calls for all the information items needed about a system to manage its life cycle. In an S*Patterns approach, design evolution or other learning can be characterized as introducing new variants or other aspects into the pattern while retaining other general aspects.

This means that patterns can reduce enterprises repeating the same work, or more importantly the same expensive or time-consuming errors. It facilitates group learning, in which IP assets have impact beyond a few individuals who originate them. It is closer to the information sharing ecosystem of the historical physical sciences, in which shared patterns of discovered phenomena advance group knowledge. While the use of patterns thus has advantages, it may also be less familiar than the traditional “from scratch” approach to models. For this reason, the INCOSE MBSE Patterns Working Group makes use of the related Innovation Ecosystem Pattern to describe the environment of the enterprise or supply chain ecosystem and how it may be different in the case of pattern-based versus non-pattern activities—both of which can exist together.

The Innovation Ecosystem Pattern is itself an S*Pattern. However, it is not a model-based description of the engineered product (such as a lawnmower or power converter). Instead, it is a model-based description of the organization, processes, enterprise, or supply chain ecosystem that is performing any kind of innovation, and how it learns, whether using formal patterns or not. It is a reference model that incorporates historical models such as ISO15288 and the INCOSE SE Handbook, but with special emphasis on how learning is performed and exploited. It helps to explain the use of S*Patterns. Refer to Figure 7, which defines three top-level reference boundaries.

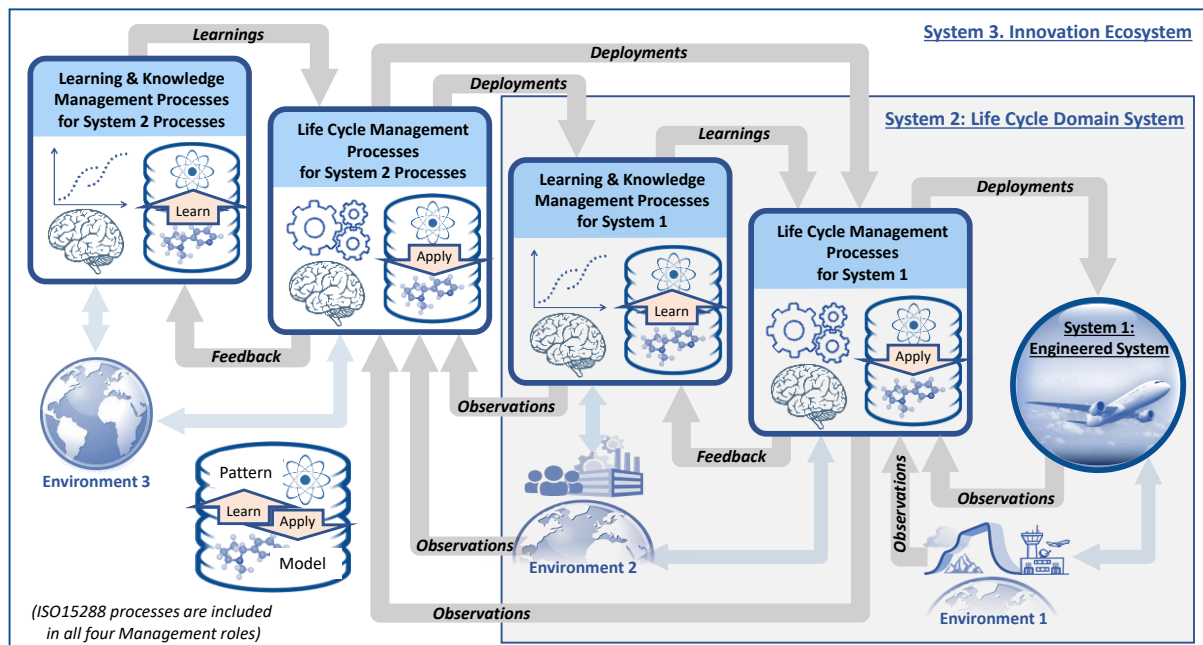


Figure 7: ASELCM Innovation Ecosystem Pattern Level 1, Reference Boundaries for Systems 1, 2, and 3

- System 1: The Engineered System.
- System 2: The Life Cycle Domain System, containing all the things that System 1 will interact with over its life cycle, and in particular the systems of engineering, production, distribution, operation and sustainment of System 1. System 2 is responsible to observe and learn about System 1 and its environment and use that learning in life cycle management of System 1.
- System 3: The Innovation Ecosystem: Responsible for life cycle management of System 2. Responsible to observe, learn about, and deploy the management processes of System 2.

The next level of decomposition of the reference pattern is Level 2, shown in Figure 8:

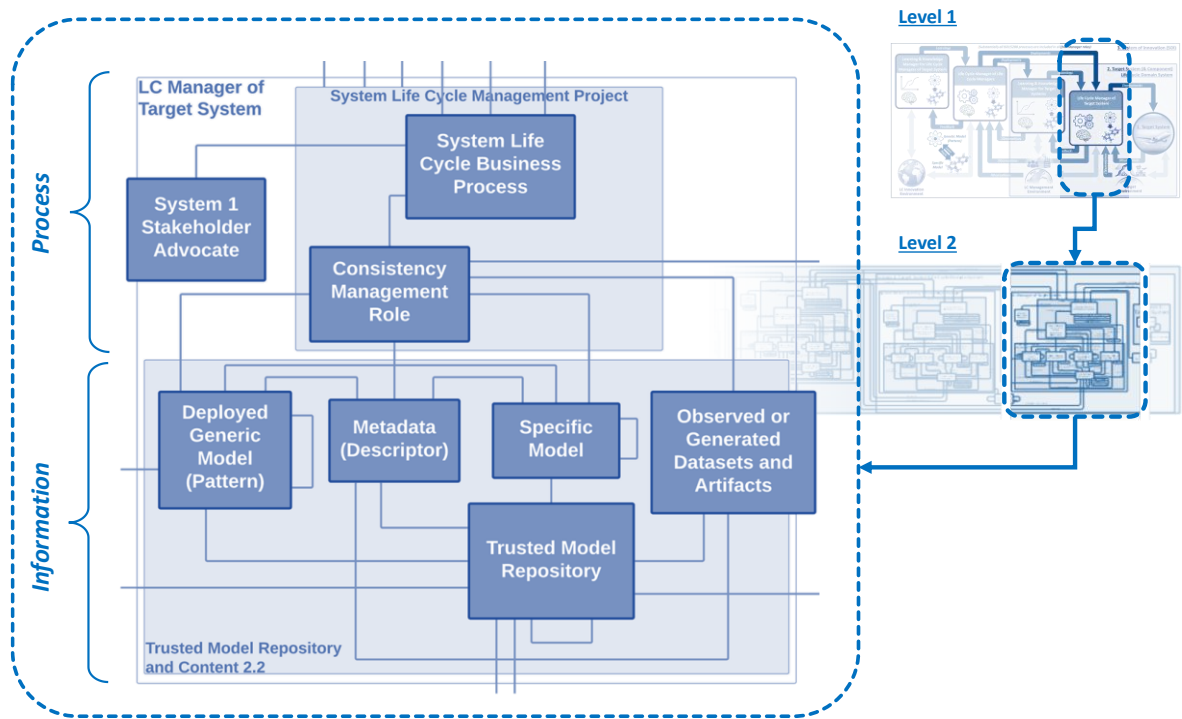


Figure 8: ASELCM Ecosystem, Reference Boundaries / Logical Roles for Level 2

There are eight interacting roles identified in Figure 8. The upper three represent business process roles and the lower five represent persistent information roles.

- **System Life Cycle Business Process:** Instance of this role are responsible to perform local enterprise processes of System 2. ISO15288 processes of the INCOSE SE Handbook or “Vee Diagram” are typical.
- **System 1 Stakeholder Advocate:** Instances of this role are responsible to represent the interests of System 1 stakeholders to the life cycle management processes.
- **Consistency Management Role:** Instances of this key role are important to the ST4SE Project, and are further described in Section 3.1. This role is concerned with all types of “consistency” concerning the information roles.
- **Specific Model:** Instances of this role are “models” of System 1, whether they are MBSE models, simulation models, or more general “models” such as manufacturing drawings, schematics, or other data structures.
- **Deployed Generic Model (Pattern):** Instances of this role are more general models (such as S*Patterns, architectural frameworks, PLE models, ontologies, or other frameworks that can be used to generate or at least check the Specific Model for conformance).
- **Observed or Generated Datasets and Artifacts:** Instances of this role can be datasets of empirical data observed or measured from the real System 1, or data generated by simulation from the Specific Model, or information captured from the System 1 Stakeholder Advocate, or documents or other artifacts of a traditional nature.

- Metadata (Descriptor): Instances of this role are information that explains the nature and content of the other three information roles (Specific Model, Deployed Generic Model, and Observed or Generated Datasets and Artifacts).
- Trusted Model Repository: Instance of this role are responsible for the providing persistent, secured access to the four information roles.

In this project, the demonstration illustrated shows how semantic technologies can enhance the digital engineering ecosystem Consistency Management role described by the Innovation Ecosystem Pattern. Advanced model generation from patterns, and model checking against patterns, are key enhancements.

To learn more about the Innovation Ecosystem Pattern, consult the resources listed in Section 2.6.

2.3. Beyond the Basics: How to Learn More

2.3.1. How to Learn More about Semantic Technologies

- “Ontology Development 101: A Guide to Creating Your First Ontology”, Natalya F. Noy & Deborah L. McGuinness, 2000,
see <http://ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness-abstract.html>
and <https://protegewiki.stanford.edu/wiki/Ontology101>
Brief introductory guide on the basic principles of ontology development.
- “Ontology Engineering”, Elise Kendall & Deborah L. McGuinness, Morgan & Claypool Publishers, 2019, ISBN: 978-1681733081
Full textbook with examples and best practices.
- “Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL (ACM Books) 3rd Edition”, James Hendler, Fabien Gandon & Dean Allemang, 2020, ISBN: 978-1450376143
Full textbook with many application examples and best practices.
- “W3C Standards”, W3C,
<https://www.w3.org/standards/>
- “The Semantic Web”, W3C,
<https://www.w3.org/standards/semanticweb/>
A general introduction to the semantic web framework, for which RDF and OWL are major enablers.
- “Web Ontology Language (OWL)”, W3C,
<https://www.w3.org/OWL/>
A general overview of OWL.
- “OWL 2 Web Ontology Language Primer (Second Edition)”, W3C,
<https://www.w3.org/TR/owl2-primer/>
The primer provides an approachable introduction to OWL 2, including orientation for those coming from other disciplines, a running example showing how OWL 2 can be used to represent first simple information and then more complex information, how OWL 2 manages ontologies, and finally the distinctions between the various sublanguages of OWL 2.
- “Ontological Modeling Language (OML) 1.1”, California Institute of Technology,
<http://www.opencaesar.io/oml/>
OML has been developed by JPL / CalTech and published as open source under the Apache license. OML is inspired by the Web Ontology Language 2 (OWL2) and the Semantic Web Rule Language (SWRL) and can be considered a gentler and more disciplined way of using these standards in the context of Systems Engineering. By mapping the OML constructs to a number of patterns expressed in subsets of OWL2 and SWRL, OML inherits its expressivity, modularity, extensibility, and description logic (DL) semantics, but also provides a concise and user-friendly syntax. Moreover, OML is

implemented using the Eclipse Modeling Framework (EMF), which gives it a Java API and integration with a large ecosystem of modeling frameworks that has been used to develop useful tools, many of which are provided by the openCAESAR project.

- “Pellet Reasoner”, Clark & Parsia,
<https://www.w3.org/2001/sw/wiki/Pellet>
and <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.6784&rep=rep1&type=pdf>
Leading RDF/OWL2 automated reasoner.
- “SPARQL 1.1 Overview”, W3C,
<https://www.w3.org/TR/sparql11-overview/>
Overview of the SPARQL query language to query and manipulate RDF/OWL graph data.
- “SWRL: A Semantic Web Rule Language Combining OWL and RuleML”, W3C,
<https://www.w3.org/Submission/SWRL/>
Contributed submission to W3C. Not an adopted W3C recommendation yet, but implemented and already useful in practice.
- “Protégé”, Stanford University,
https://protegewiki.stanford.edu/wiki/Main_Page
Leading open source desktop and web-based tool for development of RDF/OWL ontologies.

2.3.2. How to Learn More about S*Models and the S*Metamodel

- “What Is the Smallest Model of a System?”, INCOSE International Symposium.
- https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:what_is_the_smallest_model_of_a_system_v1.4.4.pdf
- “System Interactions: Making the Heart of Systems More Visible”, INCOSE Great Lakes Symposium.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:system_interactions--making_the_heart_of_systems_more_visible_v1.2.2.pdf
- “S*Metamodel, Release 5.0”.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:systematica_5_metamodel_v7.1.6a.pdf

2.3.3. How to Learn More about MBSE S*Patterns

- Patterns WG. 2019b. “Methodology Summary: Pattern-Based Systems Engineering (PBSE), Based On S*MBSE Models”
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:pbse_extension_of_mbse--methodology_summary_v1.6.1.pdf
- Patterns WG. 2021. INCOSE MBSE Patterns Working Group Web Site:
<https://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:patterns>

2.3.4. How to Learn More About Innovation Ecosystems and Their Use of Patterns

- Schindel, W., Dove, R. 2016. “Introduction to the Agile Systems Engineering Life Cycle MBSE Pattern”. Paper presented at INCOSE International Symposium. Edinburgh, UK, 18-21 July.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:is2016_intro_to_the_aselem_pattern_v1.4.8.pdf
- Patterns WG. 2020a. “ASELCM Reference Pattern: Reference Configuration Stages for Models, Model Patterns, and the Real Systems They Represent”.

https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:configuration_stages_v1.4.5.pdf

- Patterns WG. 2020b. “Consistency Management as an Integrating Paradigm for Digital Life Cycle Management with Learning”.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:aselcm_pattern_-_consistency_management_as_a_digital_life_cycle_management_paradigm_v1.2.2.pdf
- Patterns WG. 2020c. “Example Use of ASELCM Pattern for Analyzing Current State, Describing Future State, and Constructing Incremental Release Roadmap to Future”.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:example_evolutionary_roadmap_v1.3.3a.pdf
- Schindel, W. 2022a. “Patterns in the Public Square: Reference Models for Regulatory Science”. In *Annals of Biomedical Engineering*, Special Issue on Models in Regulatory Science. October, 2022.
<https://link.springer.com/article/10.1007/s10439-022-03083-z>
- AIAA Digital Thread Subcommittee, 2022. “Report on the AIAA DEIC Digital Thread Position Paper”, in Proc. of AIAA Aviation Forum, American Institute of Aeronautics and Astronautics, Chicago, IL, June, 2022. https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:aiaa-aviation_dge-05-report_deic_digital_thread_position_paper_2_.pdf
- Schindel, W., 2022b. “Realizing the Promise of Digital Engineering: Planning, Implementing, and Evolving the Ecosystem”, in Proc. of INCOSE 2022 International Symposium, Detroit, MI, July, 2022.
https://www.omgwiki.org/MBSE/lib/exe/fetch.php?media=mbse:patterns:realizing_the_vision_of_digital_engineering_is2022_v1.3.4.pdf

3. Summary of the Project

3.1. Consistency Management as a Paradigm for Engineering and Life Cycle Management

This project, as summarized in earlier sections above, is about (1) generating system models from, and therefore consistent with, general system patterns, and (2) checking the consistency of systems models with general system patterns. Consistency management has deep historical roots in engineering and life cycle management, although usually traveling under diverse other names. This project illustrates basic examples of how consistency management is enhanced by semantic technologies and related methods.

The traditional systems engineering “Vee diagram”, along with other defense and commercial enterprise models, all remind us that all engineering methods in one way or another inherently manage a series of “gaps” into acceptable “consistencies” across system life cycles:

- Consistency of system requirements with stakeholder needs
- Consistency of system designs with system requirements
- Consistency of virtual simulations with empirical measurements (model VVUQ)
- Consistency of system component production with system design
- Consistency of system performance with system requirements
- Consistency of system operation with system requirements and design
- Consistency of system sustainment with system requirements and design, operation, and maintenance
- Consistencies of many aspects with applicable technical standards, regulation, and law
- Consistencies of many aspects with learned experiences, formal patterns of requirements and design, physical science, product line rules, architectural frameworks, shared ontologies, domain specific languages, and model semantics
- Managed consistencies of the Digital Thread and Digital Twin
- Many other types of consistencies

Nearly all of these were also required consistencies in the traditional, more “tolerant” human-performed ecosystems lacking as much digital technology, even if not recognized as so.

The Consistency Management Role in Figure 8 represents the configurable set of process roles responsible for that consistency management—whether performed by humans or automated, and whether effectively performed or not. It is understandable that much of this role has historically been performed by humans, because of the required skills, judgement, and experience.

The digital engineering and modeling community finds itself in frequent conversations about a perceived need for a “single source of truth” or “authoritative source of truth” (even if multiple), reflecting concerns with diverse and inconsistent information about systems. Figure 9 reminds us this situation is not as simple as might be assumed, showing the three main sources of information in any ecosystem:

1. What the stakeholders say (market and sponsor truths);
2. What experience says (accumulated, hard-won past discoveries, physical science);
3. What empirical observation says (scientific experiment, observation, measurement).

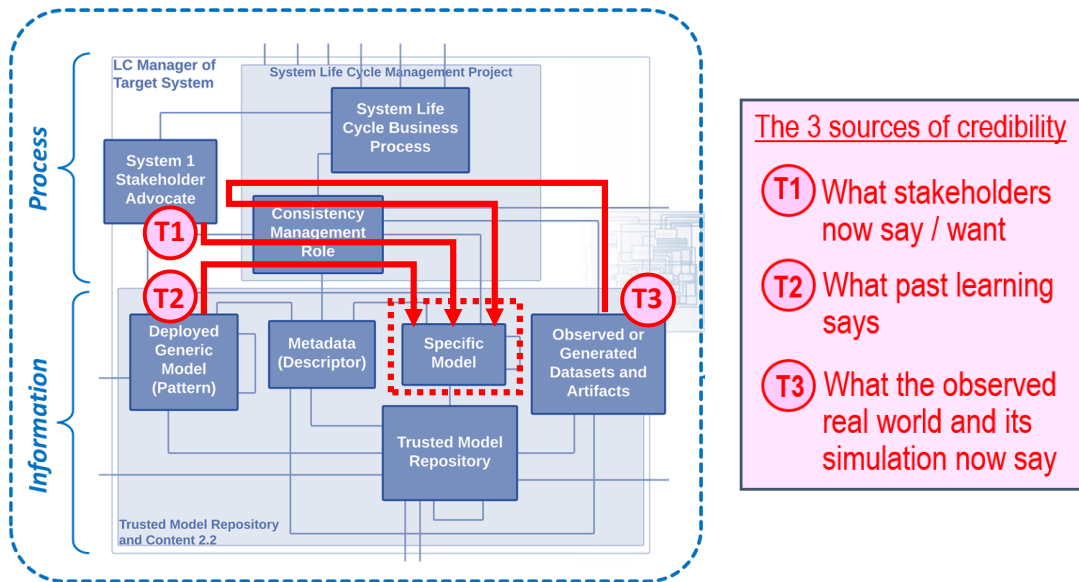


Figure 9. Roots of the consistency management challenge

The challenge is that these three sources will frequently be inconsistent (disagree with each other). The Figures 8 and 9 Consistency Management Roles of engineering and other life cycle management processes historically must recognize those inconsistencies and reconcile them.

The rise of interest in digital thread and digital twin methods aims to mitigate this consistency management challenge. This is currently being applied in a series of industry case studies by AIAA with INCOSE support. In the case of the digital twin, it reminds us of the importance of (1) managing both consistency between the virtual simulation model and the real system it simulates, and (2) managing the consistency of business processes and their information with what the trusted digital twin virtual model tells them. In the case of the digital thread, the central issue of the “thread” is managed consistency between a range of information objects along that thread. The historical predecessors to the digital thread bring important perspective to this evolution. Depending on industry domain, these include (SAE 2016), (AIAG 2006), (ISO 2016).

Because consistency gaps are often rooted in conflicting or competing interests of different parties, the Consistency Management role presents an opportunity for impactful multi-party collaboration across the ecosystem or supply chain. Enabling this collaboration with explicit models of the respective parties’ collaboration configuration spaces makes it easier to understand this setting. MBSE models and semantic technologies address the consistency of semantic structures, and can also provide the structure of quantitative relationships (for example, structure of attributes and their parametric couplings, shown in Figure 2). With that structure consistently identified, other types of automated agents (e.g., solvers and simulators) can check quantitative consistencies.

Many benefits sought through transformation to Digital Engineering have been discussed widely, such as basic issues of improved information accessibility, early virtual verification through simulation, and other gains. The Innovation Ecosystem Pattern reminds us, through the Consistency Management Role, of the wider promise that a variety of Consistency Management issues at the heart of every life cycle stage may ultimately be attacked more effectively through the aid of digital information technologies that assist in Consistency Management. These include semantic web technologies, machine learning, consistency thread signatures, configurable patterns, and pattern-based model metadata. (Herzig and Paredis 2014), (Herzig, Qamar and

Paredis 2014), (Kerstetter and Woodham 2014), (Redman, 2014), (Patterns WG 2020b). It is important to realize that consistency management describes a general paradigm covering all the alignments are the central goals of engineering, and STEM in general. For example, consistency applies not just to deterministic aspects, but includes probabilistic consistency, trust, and credibility, both formal in intuitive or heuristic.

3.2. The Interface Pattern; Use in the Traveler Power Converter Pattern

The Interface Pattern is a subset pattern of the S* Metamodel concerned directly with Interfaces, or with the larger context of Interfaces—such as would be seen in a traditional Interface Control Document. In this project, the Interface Pattern, along with other S* Metamodel elements, were applied to represent an international traveler’s power converter as discussed in Sections 2.2.2 and 3.3. This working group project followed a predecessor project called the Interface Patterns Project. The subsequent ST4SE project resulted in refinements to the Interface Pattern portion of the S*Metamodel, summarized here and in Figure 10.

The Interface Pattern portion of the formal S* Metamodel (see Section 2.4) consists of three segments: System Topologies, Interface Implementation, and Modeled Relationship Reification (Figure 10). The unifying element of the System Topologies is the Interface Element Relationship, which is up to a 4-way relationship among the Functional Interactions, Architectural Relationships, Input/Outputs, and Systems. Not all four roles are necessarily populated for each Interface Element Relationship. An example in the power converter pattern is the Interface Element Relationship among the Power Converter System and Local Environment (Systems) during the “Consume Electrical Power” interaction while dissipating Heat (I/O). A more complete set of examples for the System Topologies segment of the power converter pattern follows in Section 3.3.

The Interface Implementation portion of Figure 10 describes the connection among the Interface, its Port(s), Input/Outputs, and the System of Access (SOA). In this segment, the Port is the 3-way relationship among the Interface, I/O, and SOA. There is a 1:1 cardinality between the I/O and Port, but because a single I/O can occur in more than one Functional Interaction, the Port is coarser in granularity than the Interface Element Relationship. An example is the Port that relates the Power Output Interface with Output Power across a Device Power Connector System of Access. A more complete set of examples for the Interface Implementation segment of the power converter pattern is shown in Section 3.3.

The Modeled Relationship Reification portion of Figure 10 distinguishes between Simple Architectural Relationships and Reified Architectural Relationships. Simple ones have two relationship roles, but Reified ones may have more than two. For example, the Converts Electrical Power Architectural Relationship among the Power Converter, the Powered Device, and the Local Power Distribution System. Section 3.3 has a more complete set of examples for the Modeled Relationship Reification segment of the power converter pattern.

3.3. Enhanced Generation of a Model Consistent with a Trusted Pattern

The example trusted pattern illustrated in this project is a configurable S*Pattern (incorporating the Interface Pattern above, but larger) of an international traveler’s electrical power converter, used by the traveler in different countries with different public utility electrical standards, and the traveler’s personal electrically-powered devices. Figure 1 provides an overview which shows how the pattern and configurable model fit within the scope of the project. Figure 3 shows the domain of the power converter used as the system of interest in this project. Figures 11-21 show the S* Pattern as SysML views in Cameo Systems Modeler (CSM) tooling, with its various elements including Features, Feature Primary Keys, Feature Attributes, Interactions, Roles, Interfaces, Input/Outputs, Ports, Architectural Relationships, and Design Components. Figures 11-21 follow the “cascade” of configured model population from the pattern, as summarized by the grid of Figure 6.

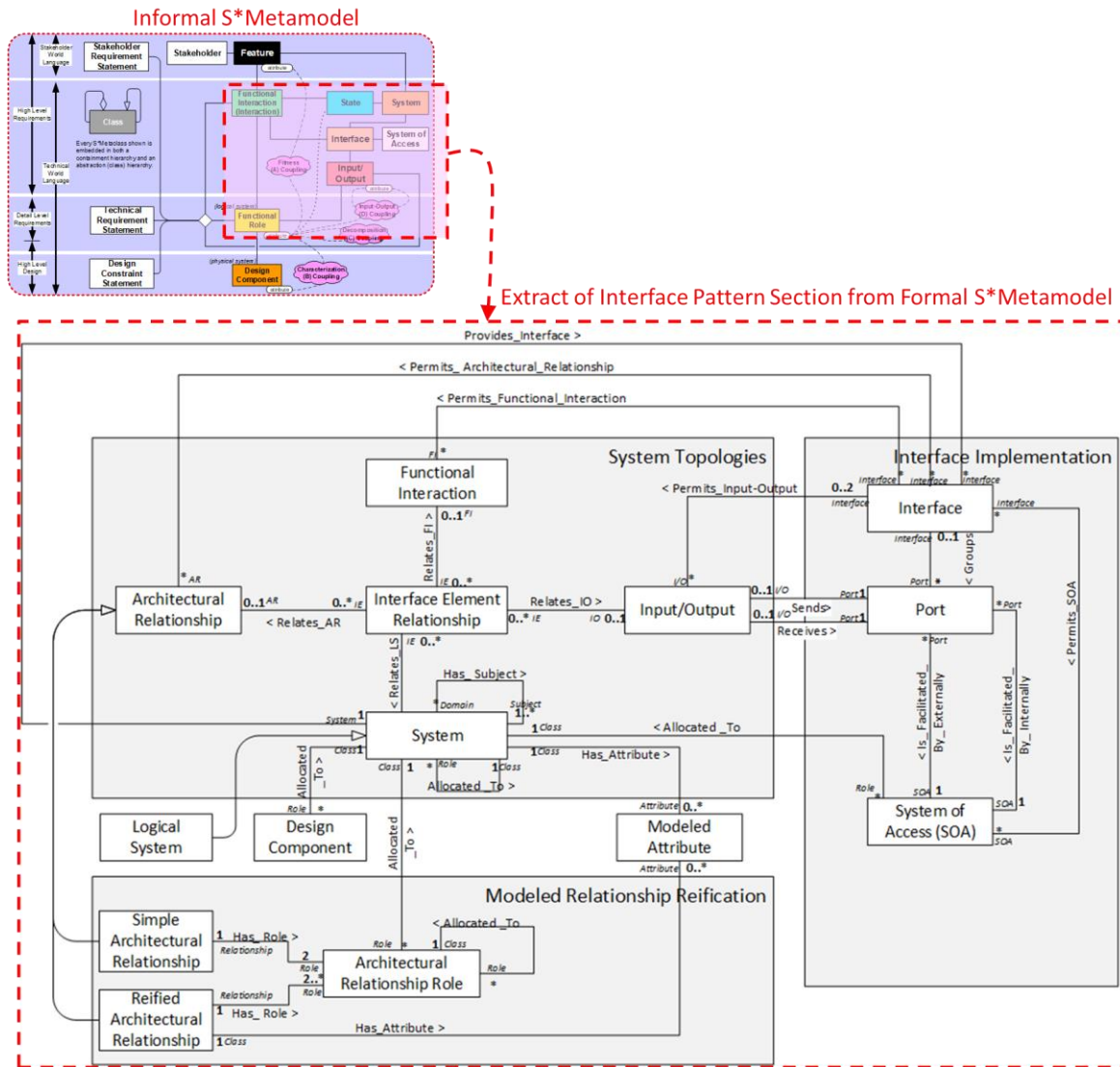


Figure 10: S*Metamodel Interface Context Segment

Additionally, the trusted S* Pattern has relationships in it between the classes/elements. Example instances of these SysML modeled relationships are also shown in Figures 11-21. These relationships not only explicate which specific pattern elements are related; they also detail the configuration rules for populating specific instances of the classes in the configured S* Model. Figure 6 details the cascade of which general pattern classes drive instantiation of the related configured model classes.

Figure 11 illustrates the S* Pattern Stakeholder Features (configurable stakeholder options built into the pattern) for the power converter. These Features describe the selectable Stakeholder value space that causes the configuration cascade, as further illustrated by Figure 6 and 23, further discussed in Sections 4.3.3 and 5.1. That cascade is moderated by the Pattern's Configuration Rules, which begin with indicating whether a given Feature has to be populated or not (e.g., Mandatory or Optional). Such Feature Configuration Rules can be more detailed by describing groups of Features whose population is mutually inclusive or exclusive. The configuration rules which moderate a cascade of population (Figures 6 and 23) are conceptually illustrated by Figure 23 and shown with examples for the Power Converter Pattern in Figures 13-18 and 21.

Features have variable parameters that are general Feature Attributes (FTA Icon in Figure 11) and more special Feature Primary Keys (FPK Icon in Figure 11). FTAs are populated repeatedly for each populated Feature instance according to the pattern, and available to carry values that further describe quantitative or qualitative details about the Features. FPKs are special parameters that allow a Feature to be multiply instantiated as specialized instances differentiated by the FPK values, to create specialized (differentiated) instances of their whole Feature, with each such instance carrying a full complement of FTAs. The possible FPK values for instantiation can either come from a present pattern list or can be manually entered at configuration time. See also Figures 23 and 25.

#		Name	Configuration Rule
1	☐ FT	Power Mains Compatibility	Mandatory
2		FTA Power Mains Type	
3	☐ FT	Powered Devices Compatibility	Mandatory
4		FPK Power Output Interface ID	
5		FTA Power Output Capacity	
6		FT Ease of Use	Mandatory
7		FT Safety	
8		FT Portability	Mandatory
9	☐ FT	Reliability and Durability	
10		FTA Design Life	

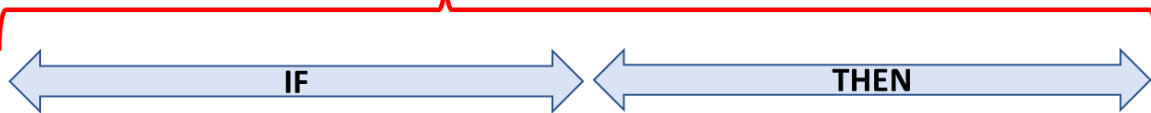
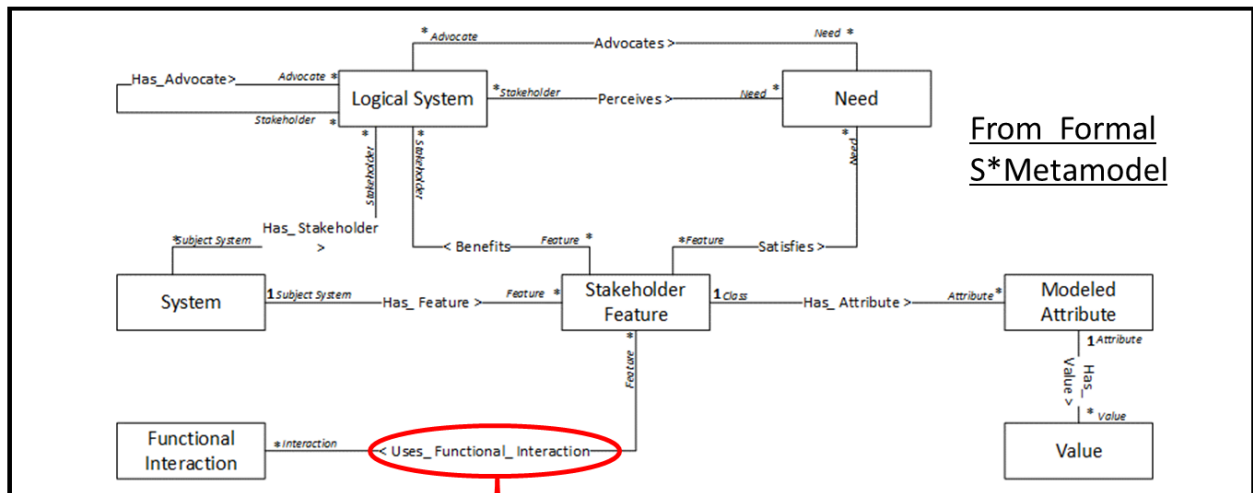
Figure 11: Stakeholder Features in the S*Pattern

Figure 12 illustrates the list of Functional Interactions for the power converter. Functional Interactions describe the interaction of the power converter with domain actors as they exchange material, energy flows, forces, or information.

#	△ Name
1	FI Consume Electrical Power
2	FI Convert Electrical Power
3	FI Display Status and Information
4	FI Handle Device
5	FI Purchase Device
6	FI Recycle Device
7	FI Resist Contamination
8	FI Set Configuration
9	FI Tolerate Moisture

Figure 12: Functional Interactions in the S*Pattern

Figure 13 illustrates the relationships between Features and Interactions in the S*Metamodel, and therefore in the example pattern, where those relationships become parameterized Configuration Rules for the “cascade” population of Interactions based on the “upstream” population of Features. An Interaction relates to a Feature when the Interaction directly describes the system behavior delivering the value of the Feature. During the configuration of a specific model from a pattern, configuration rules built into the pattern are automatically invoked to determine what to populate, and may be thought of as “IF-THEN” rules: IF a given Feature is populated (say, by stakeholder need/request), THEN a given Interaction should be auto-populated. The population can be further refined by the details shown in the FPK Value and IPK Rule columns. For the example shown in Row 9 with an FPK Value of “*ANY*” and an IPK Rule of “FPK”, any Feature Primary Key Value will be passed along to the Interaction as its Interaction Primary Key Value and differentiate it from the same Interaction type with a different primary key value.



Stakeholder Feature		Condition on FPK	Functional Interaction	Interaction PK Result
#	Type (Role B)	FPK Value	Type (Role A)	IPK Rule
1	FT Ease of Use		FI Display Status and Information	
2	FT Ease of Use		FI Handle Device	
3	FT Ease of Use		FI Set Configuration	
4	FT Portability		FI Handle Device	
5	FT Reliability and Durability		FI Resist Contamination	
6	FT Reliability and Durability		FI Tolerate Moisture	
7	FT Safety		FI Handle Device	
8	FT Power Mains Compatibility		FI Consume Electrical Power	
9	FT Powered Devices Compatibility	*ANY*	FI Convert Electrical Power	FPK

SysML Screen: Table View of S*Pattern

Figure 13: Feature-Interaction Relationships as Configuration Rules in the S*Pattern

Similarly, Figure 14 illustrates the example pattern’s relationships between its modeled Interactions and Roles, including the related configuration rules for Roles. A Role is shown related to an Interaction when the Role participates in the Interaction. During configuration, IF a given Interaction is populated THEN a given Role will be populated. The population can be further refined by the details shown in the IPK Value and RPK Rule columns. For the example shown in Row 3 with an IPK Value of “*ANY*” and an RPK Rule of “IPK”, any already

populated Interaction of the type Convert Electrical Power will cause population of an Electrically Powered Device role; also, per the “IPK” rule shown, the Primary Key Value of the Interaction will be passed along to the Role as its Role Primary Key Value and differentiate it from the same Role type with a different primary key value. For the example shown, this differentiates multiple Electrically Powered Device instances from each other (see Figure 3).

Functional Interaction		Condition on IPK	Functional Role	Role PK Result
#	△ Type (Role B)	○ «HasRole» IPK Value	Type (Role A)	○ «HasRole» RPK Rule
1	FI Consume Electrical Power	*ANY*	LS International Power Converter	
2	FI Consume Electrical Power	*ANY*	LS Local Power Distribution System	
3	FI Convert Electrical Power	*ANY*	LS Electrically Powered Device	IPK
4	FI Convert Electrical Power	*ANY*	LS International Power Converter	
5	FI Convert Electrical Power	*ANY*	LS Local Environment	
6	FI Display Status and Information	*ANY*	LS International Power Converter	
7	FI Display Status and Information	*ANY*	LS Device and Converter User	
8	FI Handle Device	*ANY*	LS International Power Converter	
9	FI Handle Device	*ANY*	LS Device and Converter User	
10	FI Resist Contamination	*ANY*	LS International Power Converter	
11	FI Resist Contamination	*ANY*	LS Local Environment	
12	FI Set Configuration	*ANY*	LS International Power Converter	
13	FI Set Configuration	*ANY*	LS Device and Converter User	
14	FI Tolerate Moisture	*ANY*	LS International Power Converter	
15	FI Tolerate Moisture	*ANY*	LS Local Environment	

Figure 14: Interaction-Role (I-R) Relationships and Configuration Rules in the S*Pattern

Figures 15-17 illustrate example pattern configuration rules for the three Segments of the Interface Context Pattern as described in Section 3.2. (The three segments are administered by the pattern owner as separate tables in SysML, but they are automatically joined during configuration as a single table describing the patterns entire Interface Context space.)

Similar to the above, an IF-THEN perspective also applies to the pattern’s configuration rules for the Interface Context components. For these components, the main “IF” part is related to the incoming configured Interaction-Role pairs (already populated from the upstream selected Features, Interactions, Roles, as described above). Refer to Figure 18. IF an incoming populated Interaction-Role pair (Functional Interaction-System pair) matches a Pattern ICT entry, THEN the ICT classes may be populated as configured components, if they are found in the same Pattern ICT entry (See Figure 18). The relationships between the modeled classes are automatically populated during configuration, based on the S*Metamodel and their appearance together in the Pattern and Configured Interaction-Role Pair. PK Values for the populated items are determined by PK Rules in the Pattern as follows: When there is a match of a configured model Interaction-Role pair with a Pattern ICT entry, then the references below to “Interaction PK Value” and “Role PK Value” refer to the PK values of that Interaction and Role.

- For Interface PK Values: If Interface PK Rule = “IPK”, then Interface PK value = Interaction PK Value, else blank.
- For Input/Output PK Values: If Input/Output PK Rule = “IPK”, then Input/Output PK value = Interaction PK Value, else blank.
- For Port PK Values: Port PK Value same as Interface PK Value (independent of Port PK Rule).
- For SOA PK Values: If SOA PK Value = “IPK”, then SOA PK Value = Interaction PK Value, else blank.

- For Reified Architectural Relationship PK Values: If AR PK Rule = “IPK”, then AR PK value = Interaction PK Value, else blank.
- For Reified Architectural Relationship Role PK Values: If AR Role PK Rule = “IPK”, then AR Role PK value = Interaction PK Value, else blank.
- For Simple (2-way) Architectural Relationship PK Values: If AR PK Rule = “”, then AR PK Value = blank. If AR PK Rule = “IPK”, then AR PK Value = Interaction PK Value. If AR PK Rule = “R1PK”, then AR PK Value = Subject System role RPK value. If AR PK Rule = “R2PK”, then AR PK Value = Object System role RPK value. If AR PK Rule = “R12PK”, then AR PK Value = composite (Subject role RPK value)-(Object role RPK value). If AR PK Rule = “IR12PK”, then AR PK Value = composite (Interaction PK value)-(Subject role RPK value)-(Object role RPK value)

#	Interaction Name	System Name	IO Name	Arch Relat
1	FI Consume Electrical Power	LS International Power Converter	IO Input Power	AR Converts Electrical Power
2	FI Convert Electrical Power	LS International Power Converter	IO Output Power	AR Converts Electrical Power
3	FI Consume Electrical Power	LS International Power Converter	IO Heat	
4	FI Consume Electrical Power	LS International Power Converter	IO EMI	
5	FI Resist Contamination	LS International Power Converter	IO Contaminants	
6	FI Tolerate Moisture	LS International Power Converter	IO Moisture	
7	FI Set Configuration	LS International Power Converter	IO Configuration Setting	
8	FI Display Status and Information	LS International Power Converter	IO Device Status and Information	
9	FI Handle Device	LS International Power Converter	IO Handling Force	
10	FI Consume Electrical Power	LS Local Power Distribution System	IO Input Power	AR Converts Electrical Power
11	FI Convert Electrical Power	LS Electrically Powered Device	IO Output Power	AR Converts Electrical Power
12	FI Handle Device	LS Device and Converter User	IO Handling Force	
13	FI Set Configuration	LS Device and Converter User	IO Configuration Setting	
14	FI Display Status and Information	LS Device and Converter User	IO Device Status and Information	
15	FI Resist Contamination	LS Local Environment	IO Contaminants	
16	FI Tolerate Moisture	LS Local Environment	IO Moisture	
17	FI Consume Electrical Power	LS Local Environment	IO EMI	
18	FI Consume Electrical Power	LS Local Environment	IO Heat	

Figure 15: Interface Context: System Topologies Segment & Configuration Rules in S*Pattern

In Figure 16, the column with heading “Name” represents the Port.

#	System Name	Interface Name	Interface PK Rule	Name	Port PK Rule	SOA Name	SOA PK Rule
1	LS International Power Converter	IFC Configuration and Information Interface		IO CL.1	IFPK	HMI Buttons	
2	LS Electrically Powered Device	IFC Device Power Interface		IO DPL.1	IFPK	Device Power Connector	IPK
3	LS International Power Converter	IFC Environmental Interface		IO EN.1	IFPK	Airspace Convection System	
4	LS International Power Converter	IFC Environmental Interface		IO EN.3	IFPK	Airspace Transport System	
5	LS International Power Converter	IFC Environmental Interface		IO EN.4	IFPK	Airspace Transport System	
6	LS Local Environment	IFC EMI Receiver Interface		IO ENV.3	IFPK	EMI Radiation	
7	LS Local Environment	IFC Environment Thermal Sink Interface		IO ENV.4	IFPK	Airspace Convection System	
8	LS International Power Converter	IFC Power Input Interface		IO PI.1		Local Power Connector	
9	LS Device and Converter User	IFC Vision Interface		IO UC1.1	IFPK	HMI Display	
10	LS International Power Converter	IFC Handling Interface		IO HI.1	IFPK	Handle	
11	LS Device and Converter User	IFC Hand Interface		IO UHO.1	IFPK	Handle	
12	LS International Power Converter	IFC Configuration and Information Interface		IO CO.2	IFPK	HMI Display	
13	LS International Power Converter	IFC Environmental Interface		IO EN.2	IFPK	EMI Radiation	
14	LS Local Environment	IFC Contaminant Source Interface		IO ENV.1	IFPK	Airspace Transport System	
15	LS Local Environment	IFC Moisture Source Interface		IO ENV.2	IFPK	Airspace Transport System	
16	LS Local Power Distribution System	IFC Power Mains Interface		IO LPO.1	IFPK	Local Power Connector	
17	LS International Power Converter	IFC Power Output Interface	IPK	IO PO.1	IFPK	Device Power Connector	IPK
18	LS Device and Converter User	IFC Finger Interface		IO UCO.1	IFPK	HMI Buttons	

Figure 16: Interface Implementation Segment and Configuration Rules in the S*Pattern

In Figure 17, the column with Heading “Name” represents the name of the Architectural Relationship Role.

#	System Name	Arch Relat	AR PK Rule	Name	AR Role PK Rule
1	LS International Power Converter	AR Converts Electrical Power		Converter	
2	LS Electrically Powered Device	AR Converts Electrical Power		Sink	IPK
3	LS Local Power Distribution System	AR Converts Electrical Power		Source	

Figure 17: Modeled Relationship Reification Segment and Configuration Rules in the S*Pattern

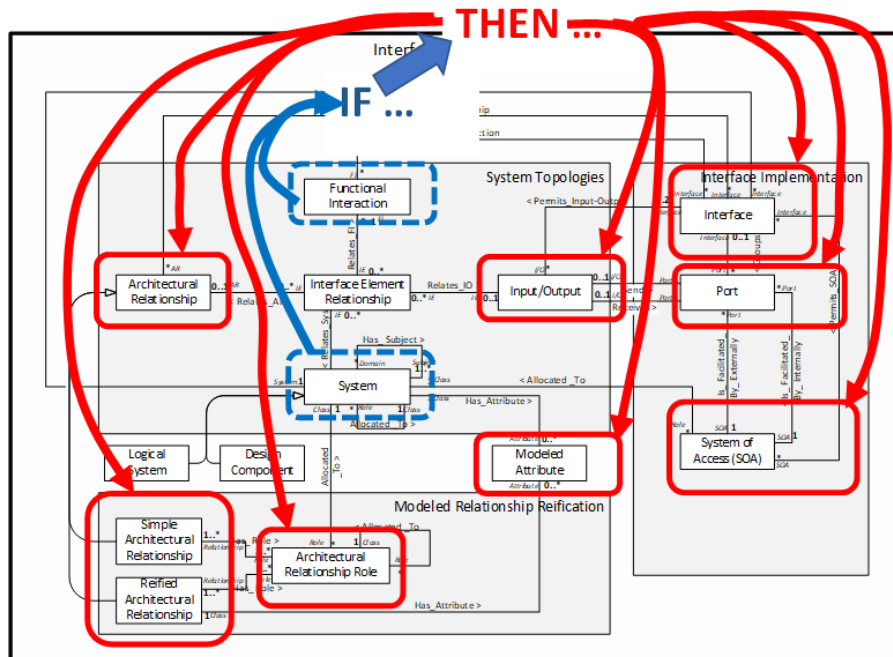


Figure 18: Details of IF-THEN Configuration Population in Interface Context Part of S*Metamodel

Figure 19 illustrates the Input-Outputs for the power converter. Input-Outputs are material, energy, force, or information exchanged through Interactions, by the power converter interacting with domain actors.

#	Name
1	EMI
3	Heat
5	Input Power
7	Output Power
9	Device Status and Information
10	Handling Force
11	Contaminants
12	Configuration Setting
13	Moisture

Figure 19: Input-Outputs in the S*Pattern

Figure 20 illustrates the list of Design Components for the power converter. Design components are the target of behavioral roles and requirement allocation.



#	Name	
1		Power Converter Assembly
2		Part Number

Figure 20: Design Components in the S*Pattern

Figure 21 illustrates the pattern’s described allocation relationships between Roles and Design Components, including the related configuration rules. A Design Component relates to a Role when the former is allocated the responsibility of meeting the behavior of the latter. During configuration, IF a given Role is populated THEN a given Design Component will be populated. The population can be further refined by the details of the Value and Rule columns. For example, an RPK Value of “*ANY*” and an DCPK Rule of “RPK” means for any Role Primary Key Value it will be passed along to the Design Component as its Design Component Primary Key Value and differentiate it from the same Design Component type with a different primary key value.

#	Type (Role B)	◉ «FunctionalRoleAllocation» IPPK Value	Type (Role A)	◉ «FunctionalRoleAllocation» Configuration Rule
1	 Power Converter Assembly		 International Power Converter	

Figure 21: Logical Systems – Design Components Allocations and Configuration Rules in S*Pattern

While the above figures and discussion describe the simple example Trusted S*Pattern utilized in the project, Section 5.1 describes the resulting S*Model configured from this pattern, and the semi-automated configuration process generating that model.

3.4. Teams, Scale, and Trust: Implications in the Larger Innovation Ecosystem Environment

Shared trust in a given model, theoretical law, engineering design, domain specific language semantic framework, or standard is a key enabler of scaled-up use of that intellectual asset. When such shared trust in a model worthy of such trust is missing, healthy growth that could have occurred is inhibited. When such shared trust is awarded to a model not worthy of such trust, problems lie ahead.

How can a large community best manage the awarding, conveyance, and management of trust in a candidate model? For simpler models, a basic approach of the physical sciences and business is to “start anew” by testing the model in real situations. For simple cases, this empirical approach also has the advantage of being repeatable by others, thereby spreading confidence. For more complex, higher expense or risk situations, this approach may be at least partly replaced by activities solely in the model domain. This project explored two means of advancing the ability of a community to do so:

1. Model Generation: Generating such a candidate model from an already trusted model of a more general nature. This requires a trusted means to generate (synthesize) a new model, suited to a specific need, from a more general trusted model or models already available. If the more generic existing model is trusted, and the method of specializing it is trusted, then trust may be awarded to the generated model.
2. Model Checking: Checking such a candidate model against an already trusted model of a more general nature. The second of these requires a trusted means to check a candidate model against a more general trusted model or models already available. Physics-based numerical simulation represents a large class of this type. The current project was concerned not with numerical calculation but with semantic reasoning about the non-quantitative aspects of a modeled system. If the more generic existing model is trusted, and a method is trusted for reasoning about a new candidate model compared to the generic model, then trust may be awarded to the new candidate model.

Both of the above means are also historical paths through which human scientists, mathematicians, and engineers have applied human analysis and synthesis skills to create proofs or derivations that bridge from more familiar trusted knowledge to new candidate knowledge. The current project has explored expansion of this capability into the realm of automated or semi-automated reasoning, using trusted computational technologies (the “methods” referenced in (1) and (2) above) holding the promise of greater capacities of certain types.

In the present state of the art, considerable large-scale impact has been enjoyed by the trust awarded in numerical models of engineered systems. Even this has been limited by many factors, as evidenced by the state of regulatory evidence for “certification by analysis simulation” in flight and medicine. Awarding trust in model semantics is at a much earlier stage, but semantic technologies offer candidates for the “trustable methods” referenced above. This project has provided a simple awareness demonstration of what is involved in both cases.

Related, but outside the scope of this demonstration project, is how the trust results of such methods may be conveyed across a large community interested in awarding or denying such trust. More can be learned about this by consulting the literature concerned with model metadata generated by numerical model VVUQ, Credibility Assessment Frameworks, and Model Credibility Life Cycle Management. (See the References Section.)

4. Tooling and Technologies Utilized

This section describes the specifics of the automated tools and technologies used in this project. Goals of this project include enabling others to repeat the demonstrations themselves.

4.1. Semantic Technologies Applied, Placed in the Larger Information Ecosystem

At its core, the Semantic Web is a set of specifications for representing knowledge, storing and retrieving those representations, drawing inferences, etc. At a low level, the specifications define how a simple fact like “the Washington Monument is 169 m high” in terms of naming and identification conventions for arbitrary *resources* as subjects, (e.g., Washington Monument), predicates (e.g., has height), and objects (e.g., the tuple (169, metre)). This so-called *Resource Description Framework* prescribes a simple, largely syntactic convention for cooperating parties to exchange assertions. The *meaning* of such assertions, that is, their correspondence to the state of the world is outside the scope of the framework; within the framework they are simply symbol triples that conform to conventions ensuring global uniqueness, interoperability, etc.

At a somewhat higher level, *RDF Schema* introduces notions of class and subclass, property and subproperty, along with entailment schemes to allow simple deductions such as “x is a Car; Car subclass of Vehicle” implies “x is a Vehicle” and “y has brother z; has brother subproperty of has sibling” implies “x has sibling y”.

At a higher level still, *OWL* (Web Ontology Language) introduces a set of axiom types from Description Logic [ref] that permit more complex entailments, e.g., every girl with at least one parent with at least one sibling is a niece.

It is well-established in logic that the more expressive the language, the more computationally complex the reasoning required to extract entailments. Description Logic (DL) is intermediate in expressivity between Propositional Logic and First-Order Logic. It represents a practical compromise in that not only can it be shown that reasoning for DL is tractable in principle, but algorithms for such reasoning have been implemented in multiple programming languages. That is, DL is the most expressive general logic for which practical reasoning codes exist.

It should be obvious that a great many assertions of interest in engineering in general and in systems engineering in particular are well beyond the expressivity of DL. DL reasoners, for example, cannot solve orbital mechanics problems, although such solutions are entailed consequences of assertions about the underlying physics, particularly Newton’s Second Law (which is a second-order differential equation). This is not a fatal flaw; in fact, it is not a flaw at all. It is merely a limitation we must bear in mind. The strength of DL reasoners is in deduction and inference.

Integrated analysis of complex systems, if it is to be trustworthy at all, must proceed from principled construction. The leaf-level (or atomic) components are described by analytic expressions (e.g., equations) that specify their intrinsic behaviors. As atoms, some of these expressions may have unbound terms that correspond to interconnections, environments, command inputs, etc. At higher levels, composites specify how their constituent components are arranged and interconnected. Analysis scenarios further specify environments, commands, etc. Principled construction in this context means that the assembly of a set of analysis equations proceeds by instantiating canonical expressions for atoms, interconnections, environments, etc. The information that drives this construction is a set of assertions about the system, its compositional structure, its interconnection structure, the environments it inhabits and their pertinent properties, inputs, etc. This information is generally well within the expressivity of Description Logic. Consequently, reasoning can provide assurance that we have constructed the correct set of expressions to present to some specialized mathematical reasoner, e.g., Mathematica, Modelica, etc. There is a great deal of sophisticated theory and software for solving specialized mathematical problems. The challenge for us as systems engineers is *to construct the right problem*; Description Logic and the Semantic Web can help us with that.

A simple example of the foregoing is the use of Kirchhoff's Laws in circuit analysis. Given a lumped element description of a circuit, that is, its voltage sources, resistors, and their interconnections, Kirchhoff's Laws describe a set of algebraic relations among voltages and currents in the circuit. In that sense it can be seen as a transformation from a structural description to its consequent analytical description. Circuit structural description is easily within the expressivity of Description Logic. It is straightforward to represent Kirchhoff's Laws as a model-to-model transformation, the output of which is another Description Logic representation. DL Reasoners cannot *solve* the algebraic equations, but DL can provide assurance that the equations represent our true intent by corresponding faithfully to the structural description.

4.2. Modeling Languages and Representations Utilized

Semantic models were constructed in OWL 2 DL, a restricted sublanguage of OWL 2 with favorable computational properties.

The things asserted to be true in an OWL 2 ontology are expressed in the form of *axioms*. Some axiom types define and circumscribe *classes* or *properties*, e.g., every *Car* is a *Vehicle*, the property *hasWheel* relates an individual of type *Car* to zero or more individuals of type *Wheel*, etc. Collections of such axioms are commonly called *TBoxes* (*T* for terminology) or *vocabularies*. Other axiom types make assertions about individuals, e.g., the individual named *http://example.com/x* is a *Car*. Collections of those axioms are commonly called *ABoxes* (*A* for assertion) or *descriptions*.

4.2.1. Representation Levels

For the purpose of reasoning, every axiom is true and there is no need to segregate by type or any other distinction. For the purpose of building a trustworthy modeling practice, however, it is helpful to segregate axioms according to their scope and asserting authority.

As a simple example, in this project we used a simple power converter as an example of a device as an object of semantic modeling. At the most concrete level, we might make assertions about an individual power converter: its serial number, hours of service, etc. These assertions are not assumed to be true of any other individual.

At a slightly higher level of abstraction, we might collect axioms assumed to be true of all power converters, e.g., every power converter has at least one input power interface and one output power interface. The collection of such axioms could be considered a *power converter vocabulary* that establishes uniform modeling patterns for all power converter applications.

At an even higher level of abstraction, we might capture modeling patterns that apply to engineered devices in general, for example, a whole-part composition relationship, patterns for relating devices to their interfaces, for interconnecting interfaces, etc. The collection of such axioms could be considered a *systems engineering vocabulary* that establishes uniform modeling patterns for the general class of interconnected, composed devices, e.g., systems.

Segregating axioms in this way allows us to associate vocabularies with engineering authority. Systems engineering patterns are formalized in a vocabulary by some appropriate authority (perhaps an Organization's Chief Systems Engineer, perhaps INCOSE); power converter patterns as formalized by a design authority for power converters. Finally, assertions about a specific individual converter are made by the manufacturer and/or its user.

4.2.2. Representation Patterns

Although OWL 2 DL is a usefully-expressive language for systems engineering, it is often cumbersome in practice to apply it directly. We illustrate using two small examples.

4.2.2.1. Disjointness Assertions

OWL defines a thing called *Class*; a *Class* is a set of individuals. We use *Class* to define things like *Requirement* and *Interface*. Although it's implicit, what we mean when we define classes like *Requirement* and *Interface* is that nothing can be both a *Requirement* and an *Interface*. In OWL, if that's what we mean, we have to say it, either explicitly:

```
DisjointClasses (:Requirement :Interface)
```

or implicitly by asserting incompatible constraints on the two classes. It's important to assert disjointness when it's what we intend because it enables a reasoner to catch a broad class of modeling errors.

We also use *Class* to define things like *IdentifiedThing* and *AggregatedThing*. In this case we do not mean that nothing can be both; it makes perfect sense for an aggregate to have an identifier. OWL gives us the low-level tools to distinguish between these (disjoint vs non-disjoint) use cases, but in practice it is tedious and error-prone to manage large taxonomies with only low-level tools.

4.2.2.2. Object Property Reification

In OWL we can say a camera acquires an image like this

```
ObjectPropertyAssertion (:acquires :camera :image)
```

The assertion itself has no identity, which means we can't say anything about it. It would be useful, however, to be able to express "Requirement R.x.y specifies that the camera shall acquire an image", i.e., to state a functional requirement. It's possible, using a technique called *reification*, to create a named individual to represent the condition that the camera acquires an image in such a way that implies the assertion above--in which a reasoner will conclude that. Because the reification itself has an identity, e.g., '*camera_acquires_image*' and we can make that the target of another assertion like this

```
ObjectPropertyAssertion(:specifies :R.x.y :camera_acquires_image)
```

This reification technique is extremely useful and we have incorporated it into our practice; nearly every relationship is reified.

4.2.2.3. Ontological Modeling Language

The Integrated Model-Centric Engineering Team at Jet Propulsion Laboratory, California Institute of Technology has encountered issues such as these for more than a decade and has practical approaches to address them in an open-source project called OpenCAESAR (see Section 4.3.2). A higher-level language they call *Ontological Modeling Language* (OML) is a key component of OpenCAESAR. OML is best thought of as a patterns-based front end for modeling in OWL2 DL. We constructed the examples for this project in OML and used OpenCAESAR tooling to express those models in OWL2 DL for reasoning and analysis. More information about OpenCAESAR is in Section 4.3.2 below.

4.3. Modeling, Semantic, and Pattern Configuration Tools Utilized

4.3.1. Modeling Tools Used in this Project

Cameo Systems Modeler (CSM) is the SysML MBSE modeling tool used in this project. Specifically, Version 19.0 SP3 is used along with the S* Profile and Project Template created for CSM. The Report Wizard functionality in CSM is used to generate the integrated set of export tables for use in the Configuration Wizard. See Section 4.3.3 below for details of the Pattern Configuration Tools used (i.e., the Configuration Wizard). Additionally, the Import CSV plug-in is used in CSM to enable the final step of the Pattern Configuration Process.

4.3.2. Semantic Tools Used in this Project

OpenCAESAR is a suite of tools developed by the Integrated Model-Centric Engineering team at the Jet Propulsion Laboratory, California Institute of Technology. It includes, among other components,

- a workbench to support building models in *Ontological Modeling Language* (OML),
- an OML-to-OWL2 DL converter that applies a disjointness management policy to simplify construction of disjointness axioms,
- a reasoner based on the Openllet open source OWL reasoner,
- a Jena Fuseki RDF triple store,
- a facility for executing SPARQL queries against the triple store, and
- workflow tooling for constructing automated analysis processes.

4.3.3. Pattern Configuration Tools Used in this Project

The general form of the enhanced generation of a Configured S* Model from a Trusted S* Pattern follows the steps shown in Figure 22. The S* Pattern data shown above is exported from the modeling tool (Cameo Systems Modeler [CSM] in this case, as SysML) into CSV [comma separated variables] files. Those CSV files are read by the Configuration Agent tool. The “User” configuring the Pattern then makes selections of specific Features and Feature Primary Key Values which the Configuration tool uses in conjunction with the rules described above to create a Configured S* Model. This Configured S* Model is then imported back into the modeling tool. See Section 5.1 for example data and additional details.

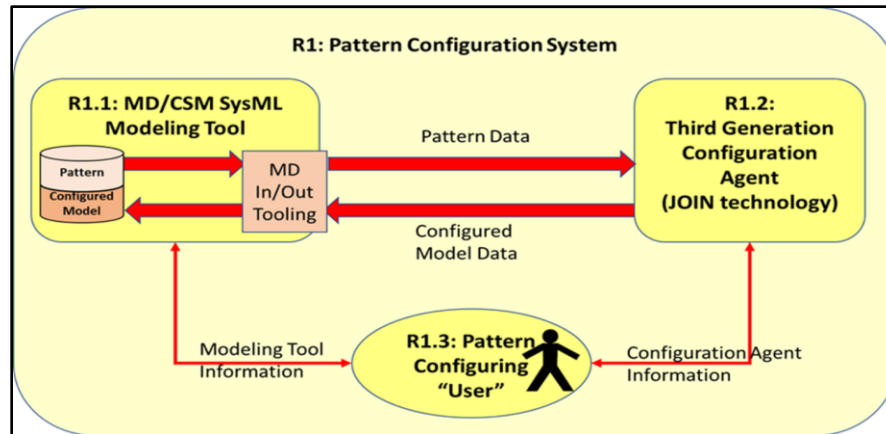


Figure 22: Overview of Repository for Pattern and Configured Model, Plus Pattern Configuration Agent and Pattern User

The S* Metamodel describes Patterns that include configuration rules used to produce configured models from those patterns. The configuration process exists independent of implementing technology, but periodic improvements are made in the latter to facilitate the former. The most recent version of implementation is the 3rd generation configuration technology created using MS Power Query in the form of database joins. Early emphasis was on configurable requirements, which was later expanded to configuring high level designs. The most recent generation expands the configuration scope to include configuring the details of interfaces and associated context.

Independent of the tool technology used to implement the configuration algorithm, the nature of the configuration rules are IF-THEN rules. Each such rule asks IF a partially configured model contains certain information, and if it is found present, the THEN part describes what should be added to the configured model. This is a reasoning-based propagation process, which is summarized in the matrix of Figure 6. A more detailed view for a subset of the S*Metamodel configuration process is provided by Figure 23.

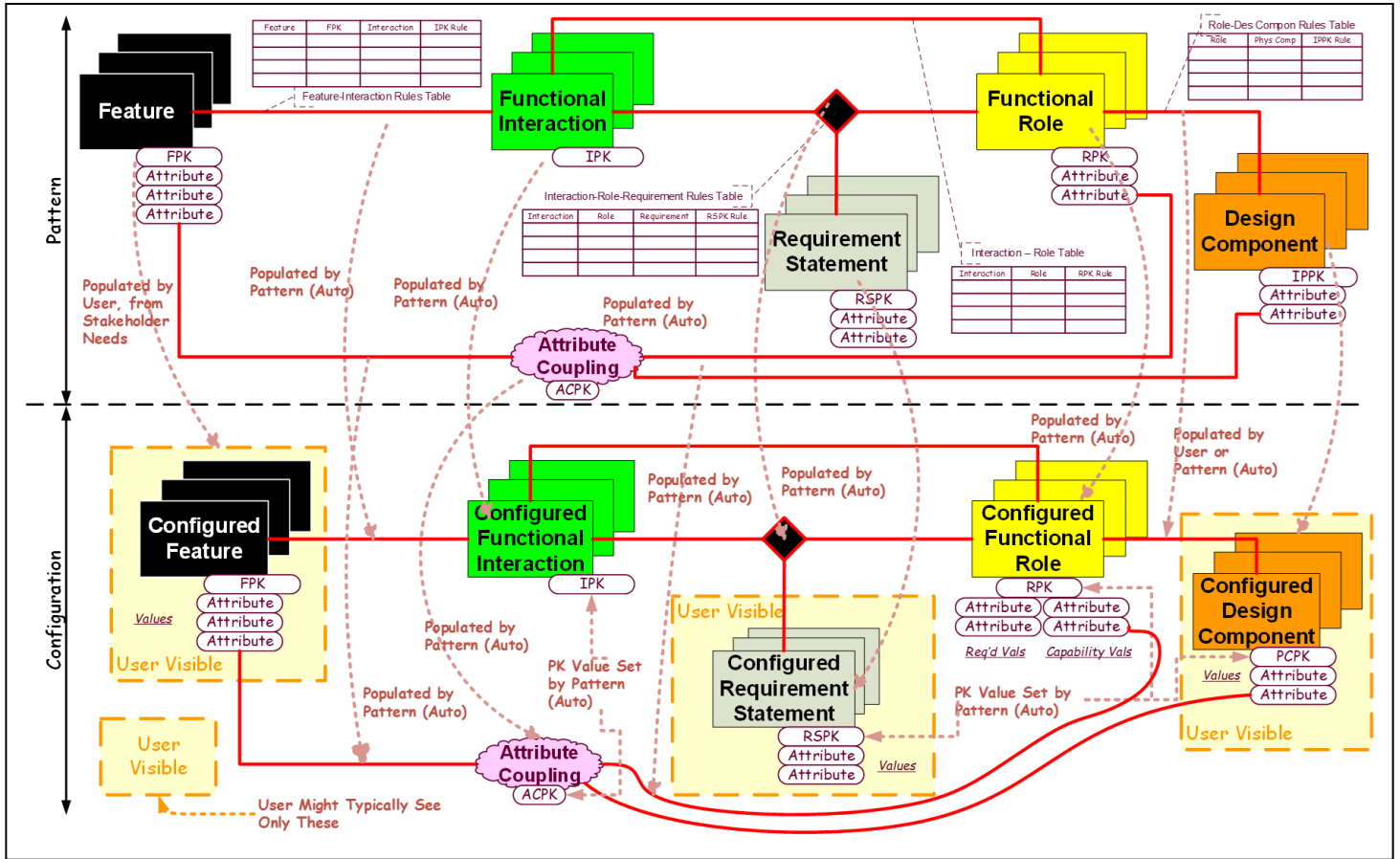


Figure 23: Conceptual Reasoning Dependency Flow of Configuration Algorithm,

The latest (third generation) automation uses JOIN technology to implement that reasoning, because of the inherent JOIN/Projection nature of what it means to generate a high dimension configured model from a low dimension compressed pattern. A simple way to think of a JOIN is creating a set of data from two sets of data, based on the similarity of some of their properties. JOIN originated in RDBMS SQL technology, but is also provided for in Semantic Technologies (e.g., SPARQL). In expanding a pattern into a populated model, the repeating JOIN paradigm is that we are comparing [a part of the pattern] with [some already populated model data], to create some [additional populated model data]:

- (a) [Pattern Feature Rules] JOIN with [Stakeholder Selections] → [Populated Features]
- (b) [Pattern Feature-Interaction Rules] JOIN with [Populated Features] → [Populated Interactions]
- (c) [Pattern Interaction-Role Rules] JOIN with [Populated Interactions] → [Populated Roles]
- (d) [Pattern Interaction-Role-Requirement Rules] JOIN with [Populated Interactions & Roles] → [Populated Requirements]

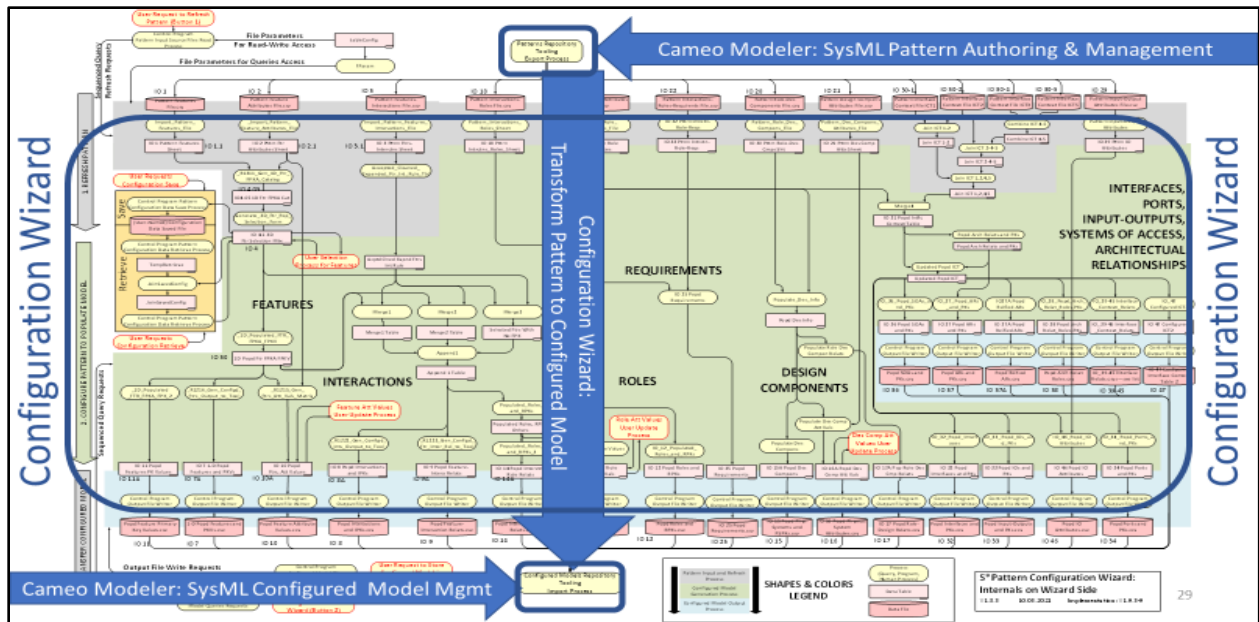


Figure 24: Flow of Configuration Agent JOINS to Populate Configured Model from Pattern

5. Project Results Demonstrated

5.1. Enhanced Generation of a Model from a Trusted Pattern

A configured model of the International Power Converter was populated automatically from the International Power Converter S*Pattern, as in the right side of Figure 1, based on the precedencies summarized in Figure 6. These populations were further governed by the configuration rules and pattern user inputs of Feature selections. Figure 25 illustrates the entry, by the Pattern Configuring User, of selected Features (Columns AI) to populate, and the Feature Primary Keys (Columns AJ-AL) entered by the user, after the rest of this form was auto-populated by the Configuration Wizard as a Pattern Features “Menu”:

Figure 25: Pattern User Input, Selecting Features and FPK Values to Populate

	A	B	C	AI	AJ	AK	AL	AM
	Feature Configuration Rule	Feature Name	Feature Attribute	Populate? Yes/No	Selection 1	Selection 2	Selection 3	Selection 4
1		Ease of Purchase		Yes				
2	Mandatory	Ease of Use		Yes				
3		Environmentally Friendly		Yes				
4	Mandatory	Portability		Yes				
5	Mandatory	Power Mains Compatibility		Yes				
6	Mandatory	Powered Devices Compatibility	Power Output Interface ID	Yes	Power Output 1	Power Output 2	Power Output 3	
7		Reliability and Durability		Yes				
8		Safety		No				
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								

Figures 26-28 illustrate some of the populated model generated as a result, including populated model Features, Functional Interactions, Logical Systems, Input-Outputs, Interfaces, Architectural Relationships, and other aspects that resulted.

#	Name
1	<input checked="" type="checkbox"/> FT Ease of Use
2	<input checked="" type="checkbox"/> FT Portability
3	<input type="checkbox"/> FT Power Mains Compatibility
4	<input type="checkbox"/> FTA Power Mains Type
5	<input type="checkbox"/> FT Powered Devices Compatibility [Power Output 1]
6	<input type="checkbox"/> FTA Power Output Capacity
7	<input type="checkbox"/> FPK Power Output Interface ID
8	<input type="checkbox"/> FT Powered Devices Compatibility [Power Output 2]
9	<input type="checkbox"/> FTA Power Output Capacity
10	<input type="checkbox"/> FPK Power Output Interface ID
11	<input type="checkbox"/> FT Powered Devices Compatibility [Power Output 3]
12	<input type="checkbox"/> FTA Power Output Capacity
13	<input type="checkbox"/> FPK Power Output Interface ID
14	<input type="checkbox"/> FT Reliability and Durability
15	<input type="checkbox"/> FTA Design Life
16	<input checked="" type="checkbox"/> FT Safety

Figure 26: Resulting Configured Model: Populated Features and Feature Attributes










#	△ Name
1	 Consume Electrical Power
2	 Convert Electrical Power [Power Output 1]
3	 Convert Electrical Power [Power Output 2]
4	 Convert Electrical Power [Power Output 3]
5	 Display Status and Information
6	 Handle Device
7	 Resist Contamination
8	 Set Configuration
9	 Tolerate Moisture

Figure 27: Resulting Configured Model: Populated Functional Interactions

Figure 28 illustrates the portion of the resulting configured model including Logical Systems, Interfaces (“Owner” Column), Ports (“Name” Column), and Systems of Access (“SOA Name” Column).

















































































#	Logical System	Owner	Name	SOA Name
1	 International Power Converter	 Configuration and Information Interface	 CL.1	 HMI Buttons
2	 Electrically Powered Device [Power Output 2]	 Device Power Interface	 DPL.1	 Device Power Connector [Power Output 2]
3	 Electrically Powered Device [Power Output 1]	 Device Power Interface	 DPL.1	 Device Power Connector [Power Output 1]
4	 Electrically Powered Device [Power Output 3]	 Device Power Interface	 DPL.1	 Device Power Connector [Power Output 3]
5	 International Power Converter	 Environmental Interface	 EN.1	 Airspace Convection System
6	 International Power Converter	 Environmental Interface	 EN.3	 Airspace Transport System
7	 International Power Converter	 Environmental Interface	 EN.4	 Airspace Transport System
8	 International Power Converter	 Power Input Interface	 PI.1	 Local Power Connector
9	 Device and Converter User	 Vision Interface	 UC1.1	 HMI Display
10	 International Power Converter	 Handling Interface	 HI.1	 Handle
11	 Device and Converter User	 Hand Interface	 UHO.1	 Handle
12	 International Power Converter	 Configuration and Information Interface	 CO.2	 HMI Display
13	 International Power Converter	 Environmental Interface	 EN.2	 EMI Radiation
14	 Local Environment	 Contaminant Source Interface	 ENV.1	 Airspace Transport System
15	 Local Environment	 Moisture Source Interface	 ENV.2	 Airspace Transport System
16	 Local Power Distribution System	 Power Mains Interface	 LPO.1	 Local Power Connector
17	 International Power Converter	 Power Output Interface [Power Output 1]	 PO.1 [Power Output 1]	 Device Power Connector [Power Output 1]
18	 International Power Converter	 Power Output Interface [Power Output 2]	 PO.1 [Power Output 2]	 Device Power Connector [Power Output 2]
19	 International Power Converter	 Power Output Interface [Power Output 3]	 PO.1 [Power Output 3]	 Device Power Connector [Power Output 3]
20	 Device and Converter User	 Finger Interface	 UCO.1	 HMI Buttons

Figure 28: Resulting Configured Model: Populated Logical Systems, Interfaces, Ports, and Systems of Access













#	Name	Architectural Relationship
1	 Configuration Setting	
2	 Contaminants	
3	 Device Status and Information	
4	 Input Power	 Converts Electrical Power
5	 Moisture	
6	 Output Power [Power Output 1]	 Converts Electrical Power
7	 Output Power [Power Output 2]	 Converts Electrical Power
8	 Output Power [Power Output 3]	 Converts Electrical Power

Figure 29: Resulting Configured Model: Populated Input-Outputs and Architectural Relationships

5.2. Enhanced Checking of a Model Against the Same Trusted Pattern

A pattern can be viewed as a set of constraints on a model. If those constraints are conveniently expressible in OML, we encode them as axioms in a vocabulary such that a violation of the pattern causes a logical inconsistency. When the reasoner checks a model for inconsistency, it is implicitly checking all such constraints. Other constraints may be beyond the expressivity of OML, or may be inconvenient to check in this way. We elaborate below. In such cases, we can perform the checks by querying the model for all cases in which the pattern should apply and evaluating whether it does. In some cases, the evaluation can be performed in the query itself.

We provide examples of both cases.

5.2.1. Checking Constraints by Reasoning

An OML vocabulary defines, among other things, classes, properties, and relations of interest in a particular domain.

As an example, the Interface Pattern defines (implicitly via graphics and prose) notions of *System* and *Interface* and a relation between them called *Provides Interface*. In the parlance of OML, *System* and *Interface* are *concepts* and *Provides Interface* is a *relation entity*:

```
concept System :> Identified [
    key hasIdentifier
]
```

This code snippet says *System* is a subclass of another class called *Identified* and that *hasIdentifier* is a key for the class *System*. To understand what that means, we examine the definition of *Identified*:

```
aspect Identified
scalar property hasIdentifier [
    domain Identified
    range xsd:string
    functional
]
```

This snippet says *hasIdentifier* is a property whose domain is the class *Identified* and whose range is a string. The *functional* characteristic says that no individual may have more than one distinct identifier. The *key* clause above says that any two systems that have the same identifier are the same system. This will turn out to be important because of another important (but sometimes surprising) feature of the open world assumptions of the Semantic Web.

Individuals in the Semantic Web (and in OML) are represented by Internationalized Resource Identifiers (IRIs), e.g.,

<http://incose.org/pwg/s-star/vocabulary/power-converter#IPC>

The Semantic Web does not incorporate the so-called *unique name assumption*. That is, an individual with a different IRI, e.g.,

http://incose.org/pwg/s-star/vocabulary/power-converter#IPC_1

is not assumed to represent a distinct individual in the real world. It is possible to assert in OWL 2 that certain individuals are different (or the same), and reasoners can, with the proper axioms, conclude that certain individuals are different (or the same).

The latter is precisely what we are setting up with *hasIdentifier*. *hasIdentifier* is a user-defined property in addition to and independent of the IRI. The combination of the key clause and the functional characteristic above ensures that (1) any two individuals with the same identifier are the same individual, and (2) any two individuals with different identifiers are different individuals.

With all that in mind, we can talk about how we relate systems and interfaces. In the encoding of the Interface Pattern in OML we find these statements:

```
concept Interface :> Identified [
    key hasIdentifier
]

relation entity ProvidesInterface [
    from System
    to Interface
    forward providesInterface
    inverse functional
]
```

The first paragraph simply says that interfaces also have identifiers and that, as before, any two interfaces with the same identifier are the same interface. The OML tooling also incorporates the disjointness management mechanism described in Section 4.2.2.1. As a consequence, the classes System and Interface are declared disjoint: no system is an interface and vice-versa.

The second paragraph says there is a reified (in the sense of Section 4.2.2.2) relationship between system and interface such that we can say a particular system *providesInterface* a particular interface. The inverse functional characteristic says that an interface may be provided by at most one system.

The latter illustrates why we need the *hasIdentifier* property as defined above. In order to ensure that an interface is provided by no more than one system, we need a way to determine whether two systems, referred to by the IRIs (or some abbreviation thereof) are different.

With just this vocabulary so far, we can enlist the reasoner in checking for a number of violations of vocabulary constraints. A few examples follow:

- **functional properties** For each property declared *functional*, the reasoner will ensure that no individual has more than one distinct value for that property. In the case of object properties (i.e., relations) the reasoner will ensure that all values of the property refer to one individual. For example, it means no individual may have more than one identifier.
- **inverse functional properties** For each property declared *inverse functional*, the reasoner will ensure that no individual has more than one distinct value for the inverse of that property. For example, no Interface may be provided by more than one System.

- **property cardinality** functional and inverse functional are simple cases of more general restrictions on minimum, exact, and maximum cardinality.
- **property domains and ranges** Any two classes for which we have not explicitly allowed an intersection will be declared (by our OML tooling) to be disjoint. That means, for example, if we define a System, we cannot say anything about that System that would imply that it is anything other than a System. If we were to assert, say, that it permits a FunctionalInteraction, then the reasoner would conclude that it must be an Interface, and since System and Interface are disjoint, our model is inconsistent.
- **range restrictions** The range of a property may be restricted over some subdomain. We might say, for example, that a particular subclass of Port may send only a restricted class of InputOutput. The reasoner will ensure all such restrictions are satisfied.

Perhaps a simple way to understand the value of reasoning in this context is to remember the distinction between vocabulary and description in OML. A vocabulary consists of axioms that establish rules for description; in a sense they define a grammar for a description language. This grammar establishes both syntactic and semantic constraints on every valid description. When we employ a reasoner in a model analysis workflow, we are checking the compliance of a *particular* description with the (vocabulary) rules that apply to *all* descriptions. That is, we are checking whether our work complies with our owl local rules, plus any superior rules we have incorporated by inclusion.

While it is reassuring to know that a model contains no logical inconsistencies, in practice it is sometimes challenging to identify the root cause(s) of an inconsistency. Some reasoners will produce an *explanation*, which is simply a small (possibly the smallest) set of axioms containing an inconsistency. From there it is left to the modeler to determine which statements are incorrect. There are undoubtedly useful heuristics to be discovered. One such heuristic would be to assume that all vocabulary axioms are correct and scrutinize description assertions for errors.

We provide examples of both cases.

5.2.2. Checking Constraints by Query

We encoded above the constraint that an Interface must be provided by at most one System. The actual constraint, however, is that in a well-formed description of an Interface there must be exactly one System that provides it. In principle, we can encode that constraint in OML as well by asserting minimum or exact cardinality on the inverse of *providesInterface*. This approach, however, produces surprising results if one fails to remember the nature of open-world reasoning.

If we assert in OML that every Interface must be provided by some System and then describe an Interface without associating it with a System, the reasoner will not find an inconsistency. In the open world, the absence of a statement associating that Interface with a System does not mean that the association does not exist. To cause an inconsistency we would further have to assert that the exact or maximum cardinality of Systems providing this Interface is zero. In that sense we are closing the description of the Interface with regard to Systems.

While this approach can work in principle, it is inconvenient for at least three reasons:

1. Generating the “closing” assertions is cumbersome and can be applied only when the description is complete.
2. Closing assertions may substantially enlarge the size of the model with consequent degradation of reasoning performance.
3. Constraints that enforce model completeness may be appropriate only at certain points in the life cycle. It may be appropriate, for example, at some early stage of development to have defined Systems and Interfaces but not yet to have allocated a System to each Interface. Forcing the reasoner to find an inconsistency at that point is not helpful. Of course, the completeness constraints can be captured in a separate vocabulary module and excluded from reasoning until wanted, but there are easier ways.

Because OML is based on OWL 2 DL, an OML modeling ecosystem can take advantage of the entire Semantic Web technology stack, which includes a powerful query language for RDF triples (SPARQL) and multiple implementations of compliant data stores. It is very convenient to check these completeness criteria using a particular query pattern.

To check the constraint that every Interface must be provided by a System, we could straightforwardly write a query (using, perhaps, the NOT EXISTS clause of SPARQL) to look for violations. Because in a well-formed model we expect no results to match the query, we are at risk of overlooking simple query errors that return empty results for the wrong reason. A somewhat more robust approach is to partition the query into two parts: the first part finds all cases for which the constraint applies, and the second part evaluates whether the constraint is satisfied for each case. In this formulation we nearly always expect a non-empty result set and the evaluation gives us a measure of progress (passing cases as a fraction of total cases).

Here is the SPARQL query to perform the check as we just described:

```
# An S*Interface must have at least one associated S*System

prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix interface: <http://incose.org/pwg/s-star/vocabulary/interface#>

select distinct ?if_l ?ok
where {
    ?if a interface:Interface ; interface:hasIdentifier ?if_l .
    bind(exists { [ a interface:System ] interface:providesInterface ?if . }
        as ?ok)
}
```

Without delving too deeply into the semantics of SPARQL,

- the `prefix` lines establish abbreviations for vocabulary namespaces
- the `select` line defines the variables returned by the query: `if_l` and `ok`
- the first line of the `where` clause finds an interface and binds `if_l` to its identifier
- the second line of the `where` clause sets the variable `ok` to true if and only if some System provides this Interface

Running the query against our test model yields this result set:

<code>?if_1</code>	<code>?ok</code>
Power Input Interface	true
Power Output Interface [Power Output 1]	true
Power Output Interface [Power Output 2]	true
Power Output Interface [Power Output 3]	true
Environmental Interface	true
Handling Interface	true
Configuration and Information Interface	true

We implemented similar queries to check the following:

- Every Interface must have at least one associated Functional Interaction
- Every Interface must have at least one associated Input/Output
- Every Interface must have at least one associated System of Access
- Every Input/Output associated with an Interface must be associated with one Functional Interaction also associated with that Interface
- Every Input/Output associated with an Interface must be associated with one System of Access also associated with that Interface
- Every Architectural Relationship associated with an Interface must be associated with one Input/Output also associated with that Interface

5.3. Gaining Access to the Project's Tooling and Information

This project report, SysML model files, and related tooling discussed in this report are available on line through the INCOSE Patterns Working Group web site, at:

https://www.omgwiki.org/MBSE/doku.php?id=mbse:patterns:st4se_project_deliverables_2022

That repository includes a link to the semantic technologies tooling and data github site, which may also be directly accessed at:

<https://github.com/opencaesar/incose-example>

6. Observations, Conclusions, and Implications for Action

6.1. Observations During the Project

6.1.1. Learning About Pattern Representation for Both Applications

One of the original motivations for this project was to gain perspective from two different kinds of uses of the same MBSE pattern, especially including the potentially different ways those applications represent the pattern and apply technologies to use it. (Recall that the general International Power Converter MBSE pattern provided the example, and within it the more general Interface Pattern portion of the S*Metamodel.)

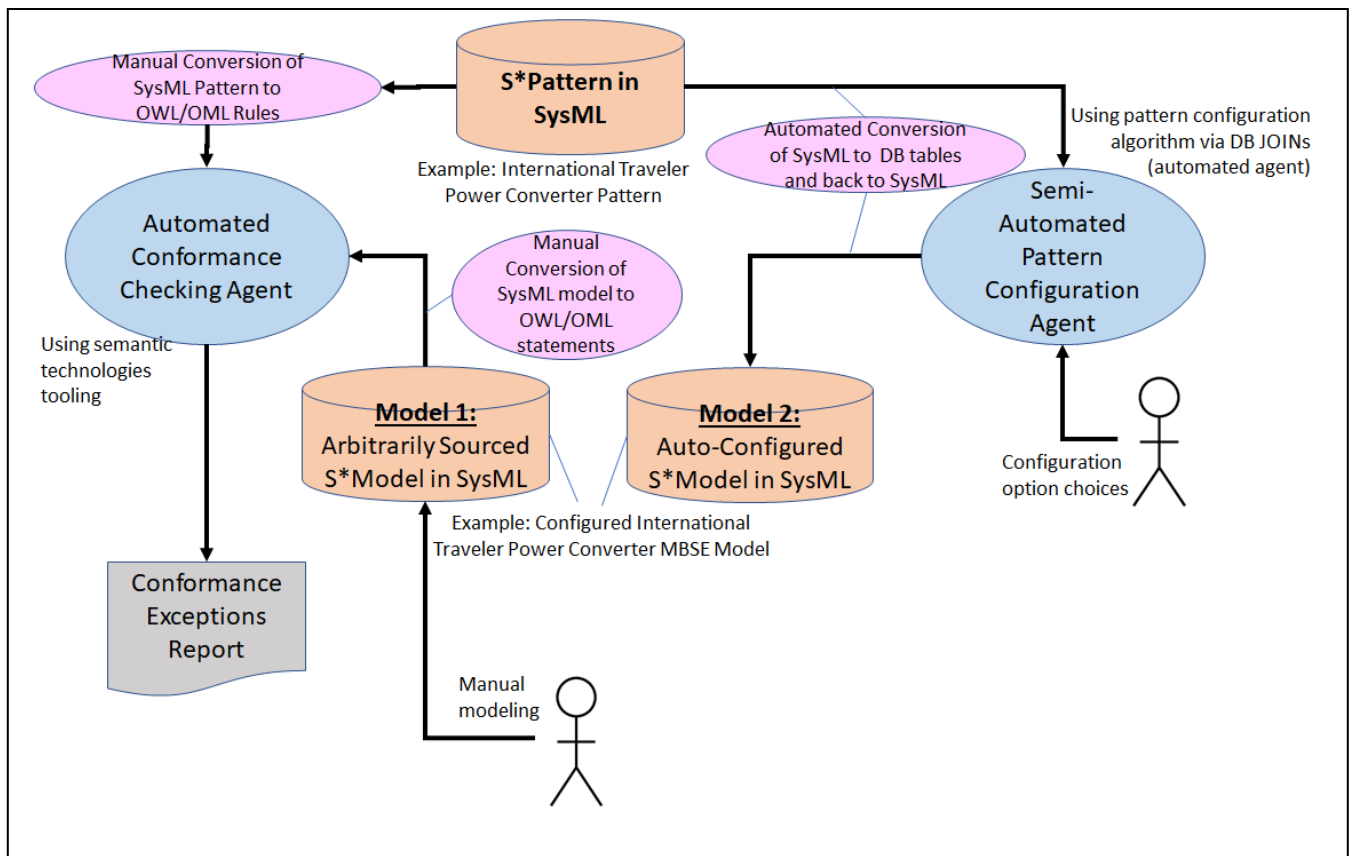


Figure 30: Conversion of a common pattern to different pattern representations, for use by different automated agents.

Referring to Figure 30, the two different representations and uses of the pattern were:

- How the S*Pattern was represented for purposes of generating configured MBSE models of specific configurations: It was converted by automated export means from SysML to tables that participated in DB JOINS, then automated imports converted back to a SysML model.
- How the very same S*Pattern was represented for purposes of testing (checking, inspecting) candidate MBSE models of specific cases, generated by other means, for conformance with the same S*Pattern: it was converted by manual means to OWL and OML representation of rules for use in semantic technologies queries to check

The use of two manual conversions as part of (B) above was not considered particularly important for purposes of this project phase, because various automation paths appear to be available or forthcoming. For example, announced plans for OMG SysML V2.0 include specified capabilities to represent SysML models in coded statement form and to automatically generate equivalent OWL statements. Both of those capabilities suggest that the manual steps on the left side of Figure XY would be automated in the near future.

Other differences between and comparisons of the two representations are of greater interest, and surface the following observations and questions for additional consideration:

1. **Use of each technology in the opposite case:** What would be the pros and cons of applying each of the two different representations and technologies above, for the opposite case?
2. **RDBMS Concern:** For MBSE models in general, relational database representations are generally not favored in comparison to network-oriented representations such as the triple store data structures frequently underlying SysML implementations. On the other hand, the situation in this case is the much more limited use of narrow scope tables (shown in Figure YY) containing configurable “edges” of the connected graph, participating in bulk JOINS to rapidly generate configured model networks which are returned to SysML as a configured model. In fact, to the extent that future SysML tooling allows for SPARQL JOINS operating directly on a SysML model pattern, the benefits of JOIN technologies may be enjoyed without the need to “map in and out of SysML” to perform the JOINS.
3. **Performance at Scale:** The performance (speed) of JOINS can be impacted by the computational fact that in the most general case, JOIN computations grow in proportion to the product of growth of the length of the two JOINed tables—a kind of quadratic growth with scale. JOINS are inherently combinatorial, on the surface. However, for the case of interest (Pattern JOINed with Configuration), this can be avoided through the use of well-established JOIN algorithms such as sort-merge JOIN or hashing JOIN. Because the Pattern side of the JOIN is available “in advance” (changes less often), it can be pre-sorted or pre-hashed. Taking advantage of these facts means that the computational load of JOIN in this case only scales linearly with growth in the size of the models involved. The fact that RDBMS JOIN technology at scale has been the subject of decades of improvement and use is of interest.
4. **Trustable Models and Reasoning:** First order/predicate logic representation, supported by trusted/proven technologies such as semantic reasoners, query engines, and languages, are attractive in a context in which future decision-making may rely on even greater use of digital models and algorithms such as explored here. That is, the criticality of earned trust will likely increase. Ability to rely on both proven and “provable” assets will increase. At the same time, we should understand that at the level of system models / patterns encoded in those frameworks, we can still encode models which are logically consistent but invalid in other ways, requiring a regime of multi-level validation and usage protocols in any case. More complete representations of just how and why we are trusting our models, managing the related uncertainties and risks, and managing change will likely continue to grow in importance. In a sense, this might be considered a form of growth of the cybersecurity challenge. Trustable underlying technologies provide an important foundation if not the whole building.

5. **Recognition How Deeply Configuration Rules Are Embedded in the Pattern:** Early in the exposition of S*Patterns by the INCOSE Patterns Working Group, it was recognized that an S*Pattern used to generate configured S*Models represents nearly all its “rules” about the models to be generated in the way that is “natural” and efficient—the pattern is simply a model of the overall class / ensemble of possible systems. An S*Pattern is really not different in character from any S*Model; there are not a large amount of “extra things” that make it a pattern. Nevertheless, we overlooked this simple truth when we were planning the list of test cases for the semantic technology “model conformance” examples of this project, in the following sense. The test cases listed in Section 5.2 can be seen to omit facts that are unique to the Power Converter Pattern, instead focusing on S*Metamodel Interface Pattern level test cases that are true of all interfaces, plus a few that are true of all electrical interfaces. In retrospect, prominently missing from this list of test cases are references to the IPC S*Pattern itself. For example, another test case on model conformance could have been the following: If the Feature configuration does not demand a second Interface for powered user devices, there should be no such interface in the S*Model being tested. This reminds us that the suite of conformance tests needs to include explicit reference to the configuration rules that are specific to / built into the S*Pattern. This “reminder” of a key economy of representation inherent in S*Patterns is a key insight for future model checking using patterns. The needed information was already built into the S*Pattern we used. Model checking code does not need to include any code that is unique to a given S*Pattern—but it needs to be parameterized by (use the content of) that Pattern in its checking, if we are to claim conformance to a Pattern. The “model checking code” can be completely generic, as long as it references the Pattern. This seems more obvious in retrospect!
6. **Configuration Spaces, Projections, Understanding:** Increased theoretical understanding of what it is that a system pattern represents in system configuration space promises to improve our ability to mix and apply multiple tools and representations in the most effective ways. Once it is understood that the S*Metamodel is simply an S*Pattern itself, with multiple specializations and configurations “below” it in a class hierarchy of webs related by the Gestalt Rules for specialization, it becomes clearer what the configuration rules embedded in an S*Pattern can tell us. They describe the structure of the connected relational cross product sub-space(s) that configuration (specialization) permits for a given pattern. This seems to be richer and more complex than some descriptions of product line engineering and “150% models” imply, but are nevertheless real configurations. Making more use of mathematical projection in this space seems to be more in line with the JOINS applied in this project, but a better understanding of how projections shine onto sub-spaces as cross-sections of the more general space could help unify the use of JOINS with the use of “is a type of” generalizations in the propositional logic representations applied in this project. Translated to the nature of JOINS, this suggests JOIN criteria other than just equality (“is a”) alone, such as “is a type of” relationships. Other relationships such as uncertainty propagation appear to also be possible in this way.
7. **Test Bed:** This project is a still-early phase effort toward a test bed in which the questions above and others may be addressed in a relatively public space accessible to the systems community.

6.2. Project Conclusions

This project has provided a proof-of-concept demonstration of propositions set forth in the “In a Nutshell” first page of this report. A shared (same, common) MBSE pattern has been used (1) to automatically generate a configured model from stakeholder needs, and (2) to automatically check another model for conformity to that pattern. Diverse contemporary information technologies, including MBSE SysML authoring tools, RDBMS Joins, and semantic reasoners have been used in this demonstration, applied to a simple pattern illustrating the principles.

In the course of the project, additional learning and questions have surfaced to suggest future work, discussed below.

6.3. Explore and Gain from this Project: Suggested Incremental Actions You Can Take

The interested reader is invited to consider taking the following incremental actions:

1. Identify a second party also interested, and work together on the following.
2. Consult the references listed in Sections 2.3-2.6 and Section 9.
3. Access the project materials and repeat the demonstration for yourself.
4. Identify a model-based pattern of interest to you and create an S*Pattern.
5. Configure specific models from the MBSE pattern of interest.
6. Separately, construct a “test model” and subject it to automated model checks.
7. Learn from the ASELCM Innovation Ecosystem Pattern about the impact of these techniques on the Innovation Ecosystem.
8. Contact members of this project team or the MBSE Patterns Working Group, and tell us about your interests, questions, and ideas.

6.4. Additional Questions for Future Work

Observations during the project, discussed in Section 6.1, have suggested additional questions outside this project’s original scope, for future work:

1. Improving Automated Model Checking Exception Explanations: Similar to situations involving programming language compilers and AI expert systems technologies, automated checking can detect exceptions that are not easy to understand by the human user—that is, a challenge for effective automated explanation of exceptions. This project has suggested that automated checking of a model can be phased against a series of patterns that progress from abstract (e.g., S*Metamodel) to more specialized (e.g., International Power Converter S*Pattern), with exceptions accordingly separated and ordered for most effective human understanding of what has been violated.
2. Further exploration at large scale of performance of the related automation technologies, in optimized conditions, can be carried out.
3. Automation of generation of the pattern checking semantic rules from the pattern can be performed. This would be of particular interest in the SysML 2.0 and OWL setting, for example.
4. Testing of the two automation technologies used in the project, for performing the opposite roles would add to understanding.
5. More complete description of the connection of this type of approach to enterprise level trust and risk management would be of value.

6. The INCOSE ASELCM Innovation Ecosystem Pattern emphasizes the role of Consistency Management across the life cycle of managed systems. A more complete exploitation of that approach would involve automated checkers not limited to semantic consistency—for example, numerical solvers, but with their invocation governed by semantic technology that recognizes types of consistencies that apply
7. A more complete description of the connections between JOINs as projections in configuration space with propositional logic would be of value to understanding of practitioners, including issues of trustable technologies.

7. Engaging with the S*Patterns Community

Interested readers are invited to contact the INCOSE Patterns Working Group or the members of this project team. Sharing questions, experiences, and further efforts to understand unfamiliar subjects is a good way to make progress in your own skills as well as your team's or enterprise's capabilities.

8. Definitions

Term	Definition
ASELCM Ecosystem Pattern	A configurable reference pattern describing the use of information assets in support of the innovation life cycle processes, including use of models, patterns, reference frameworks, datasets, and other artifacts.
Axioms	The basic assertions, in a formal descriptive language (e.g., OWL), provided by an ontology describing some domain.
Configure, Configuration	In the context of model-based patterns, the transformation of a pattern (representing a general model of a class of systems) into a specific model still conforming to the pattern.
Domain	The domain of an RDF descriptor is the class of its possible subjects.
INCOSE	International Council on Systems Engineering--the professional society of systems engineering.
MBSE; MBSE Model	When systems engineering performance incorporates the lessons of three centuries of science, engineering, and mathematics, many aspects are more effectively expressed through system models, across the systems engineering process areas and life cycle stages. Model-Based Systems Engineering emphasizes precision of representation, effective use of abstraction, and computational methods, to align multiple engineering and scientific disciplines with each other and the interests of stakeholders.
Metamodel; S*Metamodel	A metamodel establishes the formal governing information structures and meanings used for generating and interpreting models which conform to it. The S*Metamodel is a metamodel providing the minimum set of information that has been found to be necessary for the purposes of engineering and science across the life cycle of systems.
Model Checking	Verification that a given model conforms to the formal rules governing it, such as a metamodel, ontology, architectural framework, reference model, or pattern.

Term	Definition
OMG	The Object Management Group® (OMG®) is an international, open membership, not-for-profit technology standards consortium
OML	Ontology Modeling Language is a language for expression of ontologies, inspired by OWL.
Ontology	For information systems, an ontology formally represents, names, and defines the categories, properties, and relations between the concepts, data, and entities that describe one, many, or all domains of discourse.
OWL	Web Ontology Language is an W3C standards-based Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things, particularly to describe ontologies to implement the Semantic Web and related applications.
Range	The range of an RDF descriptor is the set of its possible property values.
RDF	Resource Description Framework is a W3C standard for representing information (originally for use on the World Wide Web) as primitive elements to be processed by automated agents, not just viewed by people. RDF representation is based on a "triple store" data structure primitive in which two things (resources, sometimes but not always represented by web URLs) are related by a predicate relationship, and can be considered a generalization of web hyperlinks that can be processed by SPARQL or other agents.
RDFS; RDF Schema	The RDF Schema is a set of classes, based on RDF representation, capable of representing ontologies, but less expressive for that purpose than OWL.
Reasoner; Semantic Reasoner	An automated agent capable of generating logical inferences from a set of asserted facts or axioms, with reasoning rules often described by an ontology.
S*Model	Any formal model, in any modeling language or toolset, that satisfies (has been mapped to) the S*Metamodel.
S*Pattern	A configurable, re-usable S*Model, describing a family of systems, capable of being configured to a more specific S*Model.
Semantic Web	An extension of the World Wide Web through standards set by the World Wide Web Consortium (W3C), with the goal of making Internet data machine-processable (compared to only viewable by human navigators).
Semantic Web Stack; Semantic Technology	A collection of technologies that together enable the Semantic Web or related applications; such technologies include XML, RDF, RDFS, SPARQL, OWL, SWRL, and others.
Semantics	Semantics in general are said to be about meaning; in the case of formal models, the semantics of such a model are expressed by the model itself, as well as the related ontology, metamodel, pattern, or modeling language which govern the interpretation of the model.
SPARQL	A query language for processing queries of data that is in RDF form.
ST4SE	A project of the INCOSE MBSE Patterns Working Group, concerned with targeted demonstrations about semantic technologies relevant to Systems Engineering.
SysML	A language for expressing system level models (MBSE models), based on standards generated by OMG in collaboration with INCOSE.

Term	Definition
Triple Store	A database built to store RDF primitives.
Vocabulary	For a semantic technology application, a vocabulary defines the formal ontology for the domain addressed by the application.
W3C	The World Wide Web Consortium (W3C) is an international community that develops open standards to ensure the long-term growth of the Web.
XML	XML is a text-based language for exchange of information. It is intended to be "extended" (through the addition of tags) for use in specific domains (e.g., description of molecules, description of general system models, description of content of commercial catalogs, etc.), then used to express and exchange information in those domains (e.g., description of a specific molecule, description of a specific system model, description of content of a specific catalog, etc.). XML has provided the base for many "mark up" languages across different domains, for exchange of machine processable information.

9. References

Some of the most important core technical standards and other references concerning the subject matter of this project were listed in the "How to Learn More" Sections 2.3-2.6, and are not repeated here. In addition, the following supplemental references were also cited in the body of this report:

1. AIAG 2006, 'APQP & PPAP Requirements for Automotive', Automotive Industry Action Group, Southfield, MI (US). <https://www.techstreet.com/standards/aiag-ppap-4?product_id=1257705>
2. Herzig, S & Paredis, C 2014, 'A Conceptual Basis for Inconsistency Management in Model-Based Systems Engineering', *Proc of CIRP 2014 Design Conference*, International Academy for Production Engineering, Paris, France.
<<https://www.sciencedirect.com/science/article/pii/S2212827114007586/pdf?md5=c9bdd8aba94e820ec43b56330225daa6&pid=1-s2.0-S2212827114007586-main.pdf>>
3. Herzig, S, Qamar, A & Paredis, C 2014. 'Inconsistency Management in Model-Based Systems Engineering', *Proc of 2014 Global Product Data Interoperability Summit*, Southfield, MI (US).
<http://gpdisonline.com/wp-content/uploads/past-presentations/AC45_GeorgiaTech-SebastianHerzig-InconsistencyManagementInMBSE.pdf>
4. ISO 2016, 'ISO 13485: 2016 Medical devices — Quality Management Systems — Requirements for Regulatory Purposes', International Standards Organization, Geneva, Switzerland.
<<https://www.iso.org/standard/59752.html>>
5. Kerstetter, M & Woodham, K 2014, 'SAVI Behavior Model Consistency Analysis', in *Proc. of 2014 Global Product Data Interoperability Summit*, Southfield, MI (US). <<http://gpdisonline.com/wp-content/uploads/past-presentations/AVSI-Kerstetter-SAVIBehaviorModelConsistencyAnalysis-CAE-Open.pdf>>
6. Redman, D 2014, 'Importance of Consistency Checking in the SAVI Virtual Integration Process (VIP)', in *Proc. of 2014 Global Product Data Interoperability Summit*, Southfield, MI (US).
<http://gpdisonline.com/wp-content/uploads/past-presentations/SE_67_AVSI-Redman-ConsistencyCheckingInSAVI.pdf>
7. SAE 2016, 'AS9145: APQP & PPAP Requirements for Aerospace and Defense', SAE International, Warrendale, PA (US). <<https://www.sae.org/standards/content/as9145/>>

8. Document Change History

Version	Date	Updates	Revised By
1.1.1 – 1.5.9	01.24.2021 – 04.05.2022	Initial draft and revisions by project team	Blackburn, Cureton, de Koning, Jenkins, Lewis, Schindel, Trase
1.6.0	04.05.2022	Consists of Version 1.5.9 uploaded by Stephen Lewis on March 22, plus insertions from Steve Jenkins made on April 4, 2022. Steve Jenkins' insertions are to Sections 4.1, 4.2, 4.3.2, and 5.2, and are marked using MS Word change highlighting. Since Steve Jenkins' uploaded insertions were made to a still earlier Version 1.5.5, those same insertions were copied verbatim by Bill Schindel into this file on April 5, creating this merge to avoid any confusion by reviewers.	Schindel
1.7.1	04.16.2022	Address reviewers' prior comments; relocate some material to fit outline more clearly; add interim cover page. (See also V1.7.2 below.)	Schindel; Jenkins
1.8.1	05.02.2022	Recover multiple diagrams lost in the editable DOCX file Version 1.7.1 saved on the group file sharing site. They appear undamaged in the 1.7.1 PDF saved on the same site on 04.16, but the DOCX file shows a further edit on 04.19.	Schindel
1.8.2	05.02.2022	Insert sections 6.2, 6.3, 6.4, and 7.	Schindel
1.8.3	05.03.2022	Add details to Section 3.3 further describing views and configuration rules.	Lewis
1.8.4	05.04.2022	Include insertions from Steve Jenkins to Section 5.2. Delete Figure 18 and move Figure 29 to Figure 18 location.	Lewis; Jenkins
1.8.5	05.04.2022	Update Configured Model Views in Section 5.1	Lewis
1.8.6	05.17.2022	Recover multiple diagrams lost in the editable DOCX file Version 1.8.5 saved on the group file sharing site during an edit shown as 05.16.22. This recurring condition likely caused by editing directly in Google instead of downloading the file, editing in Word, and uploading. This version also includes Steve Jenkins updates to Section 5.2 through 05.16.22	Schindel, Jenkins
1.8.7	05.17.2022	Filling out balance of Section 5.2	Jenkins
1.8.8	06.13.2022	Incorporate definitions section, including 05.31.22 meeting team feedback. Insert initial Section 2.3 Semantic Technologies References.	Schindel
1.8.9	06.14.2022	Expand section 2.3 ST references; add Appendix of sample ST source code	Jenkins; de Koning

1.9.1	06.24.2022	Merge above contributed sections; prepare pink team review version.	Schindel
1.10.1- 1.10.5	10.01.2022- 10.18.2022	Edits throughout document text and figures, responding to pink team review feedback of version 1.9.1. For details, refer to review feedback and disposition record.	Schindel
1.10.6	10.31.2022	Add links to on-line materials from this project.	Schindel