Conference on Systems Engineering Research Series

Dinesh Verma
Azad M. Madni
Steven Hoffenson
Lu Xiao

The Proceedings of the 2023 Conference on Systems Engineering Research

Systems Engineering Towards a Smart and Sustainable World

Springer

Conference on Systems Engineering Research Series

The Conferences on Systems Engineering Research Series (CSER) push the boundaries of systems engineering research and respond to new challenges for systems engineering. CSER invites researchers and practitioners to submit their work to the conference each year in alignment with the meetings' annual thematic focus. Founded in 2003 by Stevens Institute of Technology and the University of Southern California, the conference returned to the Stevens campus in Hoboken, New Jersey, in 2023, and commenced with the Proceedings of that the 20th year of CSER.

Dinesh Verma • Azad M. Madni •
Steven Hoffenson • Lu Xiao
Editors

The Proceedings of the 2023 Conference on Systems Engineering Research

Systems Engineering Towards a Smart and Sustainable World

Conference on Systems Engineering Research

Springer

Editors

Dinesh Verma
Stevens Institute of Technology
Hoboken, NJ, USA

Steven Hoffenson
Endevor
Wilmington, DE, USA

Azad M. Madni
Astronautics Aerospace and Mechanical
Engineering Department
University of Southern California
Los Angeles, CA, USA

Lu Xiao
Stevens Institute of Technology
Hoboken, NJ, USA

ISSN 3004-9849 ISSN 3004-9857 (electronic)
Conference on Systems Engineering Research Series
ISBN 978-3-031-49178-8 ISBN 978-3-031-49179-5 (eBook)
https://doi.org/10.1007/978-3-031-49179-5

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Paper in this product is recyclable.

Preface

The International Conference on Systems Engineering Research (CSER) is the primary conference for research focused on the fast-evolving systems engineering discipline and associated engineering practices. It has become a global platform for creative research in systems engineering. It addresses systems engineering research that is focused on the complexity of modern cyber-physical systems and the context within which they provide value to society.

CSER was co-founded by systems faculty leaders at Stevens Institute of Technology and the University of Southern California in 2003; accordingly, CSER 2023 was the 20th edition of this venerable research conference series. Since its inception, CSER has become the primary conference for disseminating systems engineering research; germinating new research ideas; and nucleating new collaborative initiatives between academics and practitioners across the systems research and practice landscape.

The CSER 2023 theme emphasizes the pivotal role of a transdisciplinary systems engineering research community in conceiving and creating smart systems and the transition toward a more sustainable, safe and secure society. Smart systems are systems that apply some combination of artificial intelligence, machine learning, digitalization and data analytics to provide performance enhancements, generate automated insights and enable informed decisions. Modern systems, which have profound impacts on economic, environmental and social sustainability create complex challenges that demand a transdisciplinary approach.

These CSER 2023 proceedings feature 47 chapters, authored by researchers from around the globe, covering foundational and cutting-edge topics in systems engineering research, including Artificial Intelligence for Systems and Software Engineering, Systems and Software Engineering for Artificial Intelligence, Digital Engineering, Digital Twins, Digital Transportation, Industry 4.0 and Lean Manufacturing, Model-Based Systems Engineering, Cybersecurity and System Security Engineering, Uncertainty and Complexity Management, Human-Systems Integration, Big Data and Analytics, Cyber-Physical Systems, and System Thinking. Researchers and practitioners in systems engineering will find substantial value in the CSER 2023 proceedings.

We would like to take this opportunity to acknowledge the following for their dedicated engagement and hard work to help make this CSER 2023 proceedings possible:

- Authors of 47 chapters from around the globe for their insights shared in the proceedings.
- CSER 2023 Technical Committee, Dr. Zhongyuan Yu, Dr. Eman Almar, Dr. Feng Liu and Dr. Hao Chen, for overseeing the rigorous review process.
- Organizing team from Stevens Institute of Technology who made CSER 2023 such a wonderful success.
- The Springer team who executed a rigorous review and publication process.

Hoboken, NJ, USA Dinesh Verma
Los Angeles, CA, USA Azad M. Madni
Wilmington, DE, USA Steven Hoffenson
Hoboken, NJ, USA Lu Xiao

Contents

Part I
Model-Based Systems Engineering

PySysML2: Building Knowledge from Models with SysML v2 and Python

Keith L. Lucas, Thomas C. Ford, Jordan L. Stern, and John X. Situ

Abstract The systems engineering community is pushing toward the adoption of digital engineering to improve the design process and resultant systems across the life cycle. This shift depends on the ability to produce useful digital twins. Systems Modeling Language (SysML) version 1.x and its implementing tools do not contain truly open interfaces or otherwise enable data exchange in formats to integrate systems models with a variety of data analysis and simulation tools. Lacking data interoperability, SysML version 1.x models struggle to enable a systems engineering digital engineering vision. SysML v2 corrects many of these shortfalls with specification of a text-based language and a RESTful application programming interface (API). This chapter introduces PySysML2, a prototype software application to integrate SysML v2 models with the Python-based ecosystem of analysis tools. This work demonstrates the functionality and utility of PySysML2 by describing how to read into, manipulate within, and use SysML v2 models in a Python environment. Additionally, we demonstrate SysML v2 model serialization into a portable JSON (JavaScript Object Notation) format that supports interoperability with a wide range of commonly used data analysis tools. We close with a recommendation for continued open-source exploration of the pathfinder work presented in this chapter.

Keywords SysML · SysML v2 · Python · Data science · MBSE · Modeling and simulation · Digital thread

K. L. Lucas (✉)
Department of the Air Force Digital Transformation Office, Air Force Materiel Command, Wright-Patterson AFB, OH, USA
e-mail: keith.lucas.3@us.af.mil

T. C. Ford
KBR, Inc., Beavercreek, OH, USA

J. L. Stern · J. X. Situ
Department of Systems Engineering & Management, Air Force Institute of Technology, Wright-Patterson AFB, OH, USA

D. Verma et al. (eds.), *The Proceedings of the 2023 Conference on Systems Engineering Research*, Conference on Systems Engineering Research Series,
https://doi.org/10.1007/978-3-031-49179-5_1

1 Introduction

Systems engineering is undergoing a paradigm shift. The nature of system acquisition ensures that technology and capability requirements result in complex systems with challenging development schedules. Model-based systems engineering (MBSE) is a key element of the digital transformation intended to produce trusted, high-quality systems in an increasingly complex technical environment. The utility of MBSE relies almost entirely on the usability of the models it produces. If digital systems models cannot be analyzed and executed to produce useful insights into the physical twin's attributes, behavior, and performance, then the digital engineering vision cannot be fully realized. PySysML2 is an early pathfinder for integration between the SysML v2 modeling language [1] under development by the Object Management Group (OMG) and the vast Python data science ecosystem. PySysML2 is free and open-source and available on GitHub. Ideally, others with a vested interest in the intersection of MBSE with data science and analysis will either contribute to its development, build something better, or start related projects.

Systems engineering has been moving away from the traditional document-based approach toward the use of digital models to manage, maintain, understand, and interrogate a system's technical baseline. Challenges include acquiring and deploying software tools and infrastructure, training the workforce, building quality models, and interfacing or integrating models. Some tools support model integration (e.g., ModelCenter); however, these tools can be costly and are niche markets with limited distribution. Less expensive approaches require writing custom middleware to interface with different tool application programming interfaces (APIs). A more fundamental problem is the lack of an open, standardized serialization format with which to record and share systems models. These models are often files stored in tool-specific, proprietary formats. It is worth examining the roots of this problem, as it points toward a potential solution.

One problem stems from the fact that the Systems Modeling Language (SysML) is a "a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities" [2]. Because the language does not have a textual specification, individual tool vendors have created their own file formats. A common method for serializing SysML v1.6 models for export between tools is an Extensible Markup Language (XML) format called XML Metadata Interchange (XMI). While this is a standard maintained by the OMG, it has been heavily extended by the tool vendors. Two specific problems associated with XMI are (1) there is no definitive documentation for XMI and its tool-specific extensions and (2) XMI does not represent SysML completely and is error-prone; porting a model from one tool to another using XMI results in loss of fidelity.

The problem of model portability and interoperability between applications was at the forefront when the OMG drafted a request for proposal (RFP) for SysML v2 [3]. The RFP defines interoperability as the "ability to exchange data with other SysML models, other engineering models, tools, and other structured data sources."

In its list of objectives, the RFP states: “In particular, the emphasis for SysML v2 is to improve the precision, expressiveness, interoperability, and the consistency and integration of language concepts relative to SysML v1.” This objective is supported by several related requirements throughout the RFP. A proposed specification from a consortium of members of the systems engineering community from across the industry and academia called the SysML v2 Submission Team (SST) is currently under development. In its current draft state, the proposal includes a mature prototype specification of the SysML v2 language, in addition to an open-source repository on GitHub of supporting software applications [4].

2 Literature Search

In the draft SysML v2 specification, the SST defined a basic language called the Kernel Modeling Language (KerML), which includes abstract elements and concepts common to many modeling languages. SysML v2 was then extended from KerML. The SST explains their reasoning for building two modeling languages: “By intent, KerML provides a common kernel for the creation of diverse modeling languages that can be tailored to specific domains while still maintaining fundamental semantic interoperability. SysML v2 is such a modeling language, tailored to the systems modeling domain” [4]. SysML v2 has not only a graphical specification like SysML v1 but also a first-class textual specification. Like a programming language, this textual specification is built upon a rigorous and well-defined syntax and a set of semantics.

In addition to the textual language, the SST also proposed a SysML v2 Representational State Transfer (REST or RESTful) API [5]. The SST states in its proposal that the API will “provide standard services to access, navigate, and operate on KerML-based models, and SysML [v2] models. The standard services facilitate interoperability both across SysML modeling environments and between SysML modeling environments and other engineering tools and enterprise services” [5]. The SST’s decision to define both a textual modeling specification and a RESTful API for SysML v2 ensures that model interoperability and accessibility will remain the core features of the language as it matures. This development points to a future in which the authoritative source of truth for MBSE models need not be held in vendor-specific, proprietary formats that lock data in silos.

There are two ways to interface with the SysML v2 textual language: (1) by parsing it or (2) by interfacing via the SysML v2 API. In its prototype state, PySysML2 focuses on the former, whereas implementation of the latter is on the development roadmap. Currently, the application can parse a SysML v2 model from its textual language representation, instantiating it as a data structure within the Python environment. This supports the capabilities of both simulating the model and interfacing with modern, open-source data science libraries, widely used in the scientific, academic, engineering, and defense communities. Specifically, a straightforward approach to integrating SysML v2 with the Python programming language is demonstrated through the development of the open-source application PySysML2.

3 Methods

PySysML2 is developed as free and open-source, to be released and licensed under the Apache Commons 2.0 License [6] to encourage broad use of the tool and to support incorporation into or adoption by other projects [6]. Note that the SST has chosen to license the prototype SysML v2 Release under the General Public License (GPL) [7]. This license was considered for PySysML2 but was discounted because it requires derivative products to carry the same license forward and may discourage use and adoption. As PySysML2 does not use any SysML v2 Release source code, it is not necessary to carry the GPL forward. Additionally, other related projects, for example, the HUDS (Huddle Unified Data Schema) XML Cameo plugin by the Aerospace Corporation [8], have released their source under Apache Commons. HUDS XML is an alternative to the problematic XMI serialization format for SysML v1 mentioned earlier.

PySysML2 is developed using Miniconda, a minimal subset of the widely used Anaconda Python distribution, with some supporting features and libraries in other languages. This has many advantages, but the primary advantage is that Anaconda includes most required modules and libraries out of the box. It includes not only data science libraries like NumPy, Pandas, Keras, and TensorFlow but also general support utilities. Anaconda provides easy, secure, and free access to tested and verified versions of these open-source modules, while the Miniconda distribution of Anaconda allows only required dependencies to be added as needed. Furthermore, harnessing PySysML2 to a version of Miniconda ensures that all dependencies are verified to work correctly with one another, as each is developed independently. Finally, Miniconda supports the generation of an environment setup file that supports easy installations across multiple operating systems and configurations while clearly notating all dependencies.

The fundamental capability of PySysML2 is its ability to parse the SysML v2 textual language so that models may be instantiated and contextualized in Python as data objects that can be explored, analyzed, and manipulated. Since the SysML v2 textual language is a high-level programming language, like C++ or Java, building a Parser for it is nontrivial and is more akin to building a computer language compiler. Common approaches to building a language Parser include building it from the ground up or using a preexisting workbench of language and grammar tooling. Building from scratch enables the most flexibility to handle any nuances of the SysML v2 grammar, but using preexisting grammar frameworks and tooling significantly reduces development time, while supporting the extensibility and maintainability of the Parser.

Because the SST made a great effort to define a grammar for SysML v2 that is intuitive and straightforward, using a Parser generation tool outweighs the benefit

Table 1 Grammar-related terminologies adapted from the definitive ANTLR4 reference [9]

Terms	Meaning
Syntax	Rules that govern the membership of a phrase in a language
Grammar	Specification that defines the syntax (or rules) of a language for each potential phrase
Character	Configuration of bytes that specifies a single letter, number, or another mark
Token	Combination of characters that have specific meaning; can be combined into valid phrases of the language
Tokenizing	Process by which characters are grouped into tokens
Lexer	Application that tokenizes characters and groups them into a stream of tokens that form phrases that have meaning in the context of the language according to the syntax defined by the grammar
Parser	Application that interprets the Lexer's token stream, assigning meaning to phrases of tokens as per the grammar
Tree	Data structure characterized by a root element that points to one or more child elements
Parse tree	A tree data structure that records the Parser's interpretation of the phrases defined in the token stream

gained from building a Parser from the ground up. ANother Tool for Language Recognition 4 (ANTLR4) is a powerful, open-source tool for developing Parsers from a defined grammar and is widely used across the industry and academia [9]. Twitter, for example, uses it as their primary query Parser that interprets searches by users [9]. ANTLR4 also abstracts the coding of the Parser itself from the definition of the grammar, which means that the primary artifact to develop, maintain, and extend is the SysML v2 grammar definition itself. ANTLR4 autogenerates the Parser from the grammar in a variety of programming languages (e.g., Java, C++, C#, Python, Swift) and is currently used in the prototype SysML v2 tooling released by the SST. The important terminologies related to ANTLR4 are included in Table 1.

4 Results

Following Fig. 1, the first step in interpreting a language is to read the characters. This is performed by the Lexer. The Lexer is a relatively lightweight routine for recognizing specific groups of characters as tokens (i.e., symbols or keywords) that have specific meaning in the syntax of the language's grammar. For example, the words `part`, `def`, `use`, and `case`, along with symbols like `{`, `}`, and `:>` have specific meanings in SysML v2, so its Lexer should recognize those combinations of characters as valid tokens. The Lexer is also responsible for things like handling whitespace (which, in the case of SysML v2, means ignoring it), along with numbers and strings.

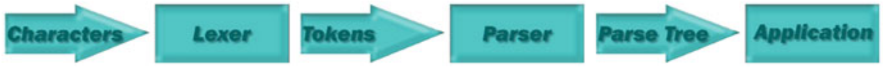


Fig. 1 Grammar parsing workflow

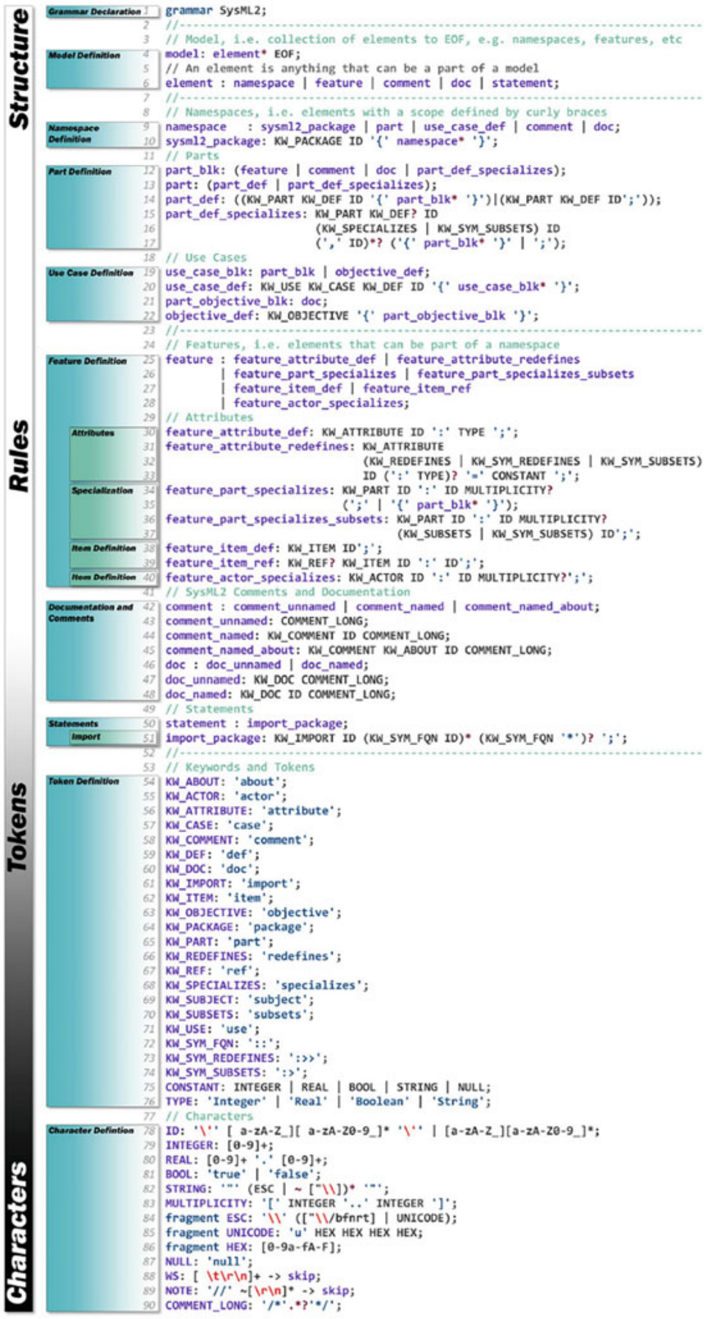
Once the Lexer converts all the characters into valid tokens, the much more complex Parser takes over. The Parser must interpret the tokens and the order in which they appear in the context of the grammar's syntax. This can be straightforward for certain constructs, but it becomes complicated very quickly. Nested constructs, for example, those involving braces or parentheses, are particularly common across most programming languages and can be quite complex. Peculiarities of the language (e.g., overlapping rules and redundant keywords, prominent features of SysML v2) compound any inherent complexities as well.

A parse tree is the output of a Parser based upon the grammar and serves as the raw interpretation of the source code being recognized. PySysML2 interfaces with the parse tree through a special class called the Visitor that retrieves all required information from the tree as needed. Figure 2 breaks down the SysML v2 grammar implemented so far in PySysML2. It is worth noting again that the Lexer, Parser, and base Visitor classes are automatically generated from this grammar. Subsequent sections describing PySysML2's implementation expand on the Lexer, Parser, and Visitor pattern through a demonstration on a notional but nontrivial SysML v2 model.

ANTLR4 is built around a specialized "metalanguage" suited for describing the grammars of other programming languages [9]. Using the ANTLR4 metalanguage, a grammar specification for SysML v2 is constructed, derived from the documentation released by the SST [10]. From this grammar specification, ANTLR4 generates the source code required to interpret textual language. The grammar specification is designed to be extensible. As PySysML2 evolves, this extensibility supports growth beyond the initial subset of the language chosen for this project, eventually including the whole specification. It also supports adaptability, as the language will inevitably change before the final adoption of SysML v2 by the OMG.

The process by which the SysML v2 textual language is interpreted by PySysML2 is straightforward. First, the character stream (i.e., the SysML v2 code) is read and tokenized by the Lexer according to the predefined grammar. For example, the SysML v2 code "`part def 'RAM';`" is read as a character stream, then the Lexer tokenizes the stream using the grammar by identifying SysML v2 keywords (e.g., "`part def`"), organizational symbols (e.g., semicolon), and identifiers (e.g., RAM). The Parser builds a data structure called a parse tree from the tokens.

Now that PySysML2 can parse SysML v2, the next task is to access and manipulate the model information in support of management, cost analysis, operations analysis, engineering analysis, and general data science. The task of PySysML2 is to expose, organize, and collate the data within a SysML v2 model in such a way that it is congruent with standard data science data structures. In other words, PySysML2



```
grammar SysML2;
//--------------------------------------------------------------------------
// Model, i.e. collection of elements to EOF, e.g. namespaces, features, etc
model: element* EOF;
// An element is anything that can be a part of a model
element : namespace | feature | comment | doc | statement;
//--------------------------------------------------------------------------
// Namespaces, i.e. elements with a scope defined by curly braces
namespace   : sysml2_package | part | use_case_def | comment | doc;
sysml2_package: KW_PACKAGE ID '{' namespace* '}';
// Parts
part_blk: (feature | comment | doc | part_def_specializes);
part: (part_def | part_def_specializes);
part_def: ((KW_PART KW_DEF ID '{' part_blk* '}')|(KW_PART KW_DEF ID';'));
part_def_specializes: KW_PART KW_DEF? ID
                      (KW_SPECIALIZES | KW_SYM_SUBSETS) ID
                      (',' ID)*? ('{' part_blk* '}' | ';');
// Use Cases
use_case_blk: part_blk | objective_def;
use_case_def: KW_USE KW_CASE KW_DEF ID '{' use_case_blk* '}';
part_objective_blk: doc;
objective_def: KW_OBJECTIVE '{' part_objective_blk '}';
//--------------------------------------------------------------------------
// Features, i.e. elements that can be part of a namespace
feature : feature_attribute_def | feature_attribute_redefines
        | feature_part_specializes | feature_part_specializes_subsets
        | feature_item_def | feature_item_ref
        | feature_actor_specializes;
// Attributes
feature_attribute_def: KW_ATTRIBUTE ID ':' TYPE ';';
feature_attribute_redefines: KW_ATTRIBUTE
                             (KW_REDEFINES | KW_SYM_REDEFINES | KW_SYM_SUBSETS)
                             ID (':' TYPE)? '=' CONSTANT ';';
feature_part_specializes: KW_PART ID ':' ID MULTIPLICITY?
                          (';' | '{' part_blk* '}');
feature_part_specializes_subsets: KW_PART ID ':' ID MULTIPLICITY?
                                  (KW_SUBSETS | KW_SYM_SUBSETS) ID';';
feature_item_def: KW_ITEM ID';';
feature_item_ref: KW_REF? KW_ITEM ID ':' ID';';
feature_actor_specializes: KW_ACTOR ID ':' ID MULTIPLICITY?';';
// SysML2 Comments and Documentation
comment : comment_unnamed | comment_named | comment_named_about;
comment_unnamed: COMMENT_LONG;
comment_named: KW_COMMENT ID COMMENT_LONG;
comment_named_about: KW_COMMENT KW_ABOUT ID COMMENT_LONG;
doc : doc_unnamed | doc_named;
doc_unnamed: KW_DOC COMMENT_LONG;
doc_named: KW_DOC ID COMMENT_LONG;
// Statements
statement : import_package;
import_package: KW_IMPORT ID (KW_SYM_FQN ID)* (KW_SYM_FQN '*')? ';';
//--------------------------------------------------------------------------
// Keywords and Tokens
KW_ABOUT: 'about';
KW_ACTOR: 'actor';
KW_ATTRIBUTE: 'attribute';
KW_CASE: 'case';
KW_COMMENT: 'comment';
KW_DEF: 'def';
KW_DOC: 'doc';
KW_IMPORT: 'import';
KW_ITEM: 'item';
KW_OBJECTIVE: 'objective';
KW_PACKAGE: 'package';
KW_PART: 'part';
KW_REDEFINES: 'redefines';
KW_REF: 'ref';
KW_SPECIALIZES: 'specializes';
KW_SUBJECT: 'subject';
KW_SUBSETS: 'subsets';
KW_USE: 'use';
KW_SYM_FQN: '::';
KW_SYM_REDEFINES: ':>>';
KW_SYM_SUBSETS: ':>';
CONSTANT: INTEGER | REAL | BOOL | STRING | NULL;
TYPE: 'Integer' | 'Real' | 'Boolean' | 'String';
// Characters
ID: '\'' [ a-zA-Z_][ a-zA-Z0-9_]* '\'' | [a-zA-Z_][a-zA-Z0-9_]*;
INTEGER: [0-9]+;
REAL: [0-9]+ '.' [0-9]+;
BOOL: 'true' | 'false';
STRING: '"' (ESC | ~ ["\\])* '"';
MULTIPLICITY: '[' INTEGER '..' INTEGER ']';
fragment ESC: '\\' (["\\/bfnrt] | UNICODE);
fragment UNICODE: 'u' HEX HEX HEX HEX;
fragment HEX: [0-9a-fA-F];
NULL: 'null';
WS: [ \t\r\n]+ -> skip;
NOTE: '//' ~[\r\n]* -> skip;
COMMENT_LONG: '/*'.*?'*/';
```

Fig. 2 SysML v2 grammar source represented in the ANTLR4 format

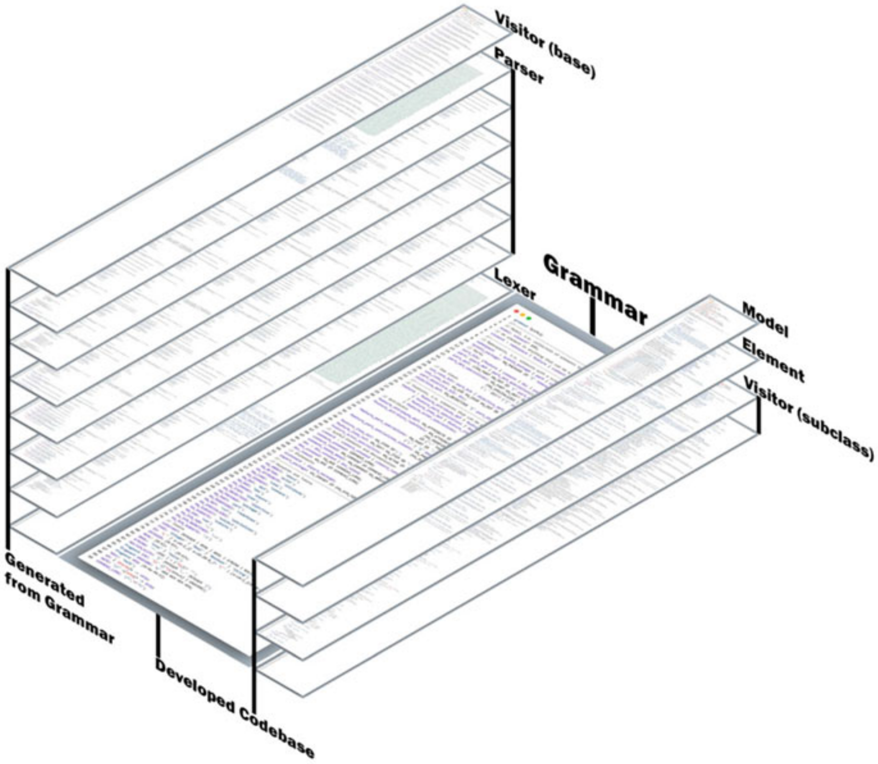


Fig. 3 PySysML2 codebase

must transform the data of a SysML v2 model so that it can be structured as trees, graphs, data frames, and multidimensional arrays.

The Lexer, Parser, and Visitor classes are automatically generated by ANTLR4 from the grammar. Figure 3 shows the relative size and complexity of these generated classes. The classes generated from the grammar, shown on the left-hand side of the figure, constitute approximately 2800 SLOC (source lines of code), while the grammar itself, shown in the middle, is only 100 SLOC. The code developed for PySysML2 to implement the SysML v2 modeling language, shown on the right-hand side, is about 1200 SLOC.

A SysML v2 model is, at its core, a tree data structure, hierarchically organized into packages and elements like parts with attributes, users, use cases, and others. Additionally, once the SysML v2 model has been taken in by the Parser, all its data exist in a parse tree. This drives the architecture quite naturally toward a tree as its foundational data structure.

Using our method, a PySysML v2 model is represented by a Model class in Python. A Model is defined as a root node that points to the first element of a SysML v2 model, which may then point to arbitrary sub-elements and so on.

Since a Model is a tree of Elements and Behaviors, each Element or Behavior must also be a tree since they too can have child nodes. A SysML v2 part, for example, may have many attributes, and a use case may contain actors (which are specialized parts), along with objectives, which, in turn, can contain comments. The underlying tree architecture of SysML v2 is driven by this feature of KerML: "A root Namespace is a Namespace that has no owner. The owned members of a root Namespace are known as top level Elements. Any Element that is not a root Namespace shall have an owner and, therefore, must be in the ownership tree of a top level Element of some root Namespace" [10]. "Model" elements are populated by interrogating the parse tree through a subclass of the Visitor class generated by ANTLR4. The PySysML2-specific Visitor subclass can traverse the parse tree on demand, executing application-specific code driven by the rules of the modeling language—which, in the case of PySysML2, is to transform the model elements into data science-friendly data structures.

The first task, then, is to implement a general tree data structure in Python. We make use of the popular Python module Anytree [11]. With the base tree class in hand, the "Model," "Element," and "Behavior" classes are now defined as customizable subclasses extending the Anytree abstract class. Additionally, the PySysML2-tailored "Visitor" subclass is also be defined. Once the Model class is implemented, functions for transforming the data into tables for data frames, graphs, and arrays can be implemented. With these in hand, interfacing a SysML v2 model with Numpy, Pandas, SciPy, and others is straightforward.

Figure 4 shows the inheritance structure of the primary user facing PySysML2 classes, which includes "Model," "Element," and "Behavior." All three inherit from the Anytree abstract class, via "NodeMixin." Any class that inherits from "NodeMixin" can now be part of a tree as either the root, nodes with children, or leaves. Essentially, "NodeMixin" defines a field that points to the node's parent and children, supporting tree traversal. "Element" and "Relationship" each inherit from "RootSyntacticElement" as well as "NodeMixin." The SysML v2 specification defines a "root syntactic element" as the primary super class of any model element or behavior [4]. This class, then, has all the fields that "Elements" and "Behaviors" have in common, in addition to functions shared by both. In turn, "Element" and "Behavior" each have subclasses of their own that define specific elements of the language, like `part`, `attribute`, and the `redefines` relationship. Encapsulating each element and behavior in this way supports maintenance as well as future extensibility. The first version of PySysML2 only supports the basic elements of the language, enough to build useful but still simple models. Future iterations, though, will gradually incorporate the rest of the specification.

To provide an initial validation of the PySysML2 method, we created a simple, notional, integrated system called the tabletop roleplaying game (TTRPG) eToken. The primary use case for this system is to display a user-defined image on a small, round liquid crystal display (LCD) screen that can be used as a customizable game token in a tabletop game. It is driven by a micro-central processing unit (micro-CPU) controller with Wi-Fi and/or Bluetooth capability and is battery-powered. The user

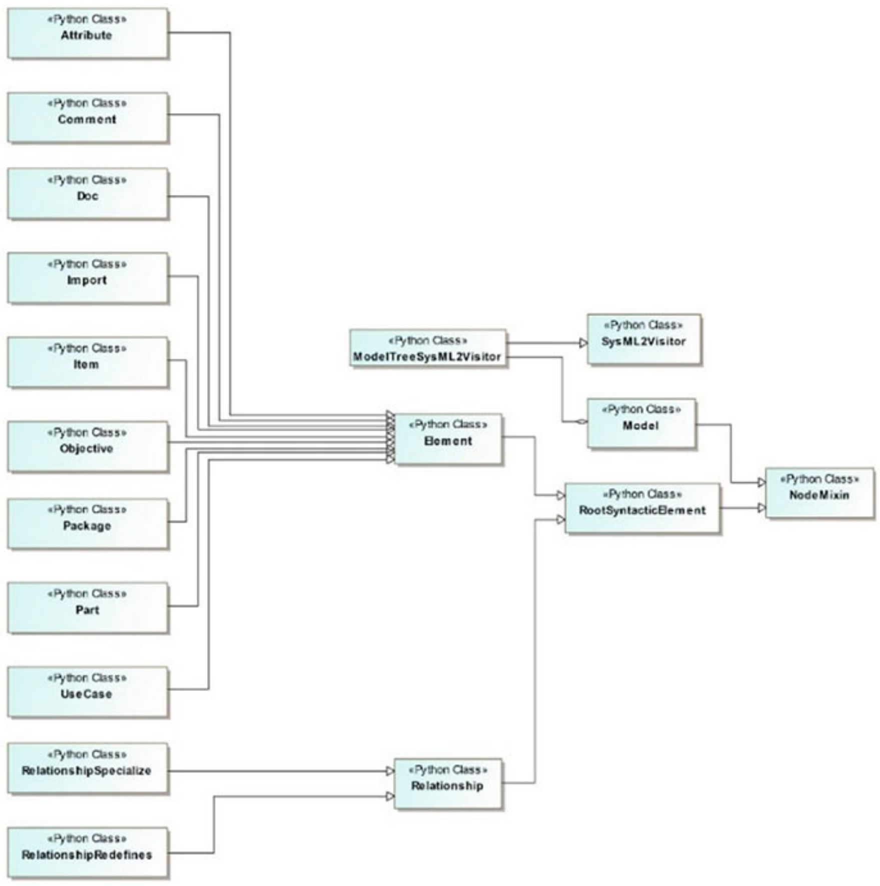


Fig. 4 PySysML2 inheritance structure

can upload new images to the token, remove existing images, and change displayed images stored in a buffer to fit their needs as the game progresses.

A walk-through of the usage of PySysML2 for the TTRPG notional systems model is described next. First, the user accesses PySysML2 through a command line interface and points to a SysML v2 source file (e.g., the TTRPG eToken model). PySysML2 parses the model source using the Lexer and Parser code that was generated from the grammar shown in Fig. 2. The model of the TTRPG system is built-in memory from the PySysML2 classes, a "Model" object is instantiated, and its root node is defined. "Model" then interacts with the TTRPG SysML v2 model through the SysML2 Visitor class, which walks the parse tree. Element and Behavior objects are instantiated by SysML2 Visitor and saved in a temporary array in "Model." After walking the parse tree, "Model" loops through the array of "Elements" and "Behaviors," building out the model tree with nodes pointing

Table 2 Data structure transformation

Data structure	Implementation
Multidimensional array	Numpy array. The primary interface to the majority of Python-based data science tools
DataFrame	Pandas DataFrame. A specially formatted multidimensional array with spreadsheet functionality
Tree	Anytree. Stores a model in a hierarchical data format like HDF5 [12]
Nested dictionary	Pure Python. Serializes JSON and XML; efficiently handles large data sets [13]

to their children and back to their parents. Now that "Model" is fully instantiated, the TTRPG eToken model may now be accessed and manipulated by the user in their data analysis environment. Several helper capabilities have also been included and, depending on the command line arguments, the model can be transformed into any one of several data structures. Once PySysML2 completes a run, multiple output files are generated: a tabular version of the model, a JSON (JavaScript Object Notation) serialization of the Python object states, a text file listing all elements in a hierarchy, and a visual graph of the model tree structure.

The PySysML2 application was developed using test-driven development (TDD) and was validated through test cases based on the notional TTRPG eToken model. Specifically, this model was built using the Java-based modeling tools provided by the SST and confirmed to be a valid model through the built-in linting and error-checking capability. Then, the output files were evaluated after running PySysML2, comparing the transformed, formatted model in both tabular and JSON forms to the original textual code. Lastly, the visual graph was compared to the model hierarchy.

With the ability to read, parse, build, and manipulate models in memory now implemented as data structures listed in Table 2, the ability to serialize models into a portable file format rounds out the foundational capabilities of PySysML2. One of the most important goals of PySysML2 is increasing model portability and interoperability, both between other modeling tools and between external data science, simulation, and analysis tools. The foundation of this capability is the implementation of well-documented serialization schemas that allow models to be exported and imported across tools with full fidelity. Additionally, multiple serialization formats tailored for specific purposes are useful. The primary purpose of serialization is to preserve the precise state of objects in memory in a binary or textual representation so that they can be saved and reconstituted later, either by the originating application or after transmission to other applications. A secondary benefit of serialization is the flattening of complex models into simpler formats that can be analyzed. For example, serializing a model to a relational table would allow its direct examination in a spreadsheet for any number of purposes. Hence, we built two serialization formats: one optimized for general portability between applications and the other optimized to support applications built around relational tables, like databases and spreadsheets.

Serialization of SysML v2 models could be accomplished somewhat by the textual language itself. The main problem with relying on textual language for serialization, though, is that there is no precise, one-to-one correspondence from a model back to the textual source from which it is compiled. In other words, the same model can be generated from multiple configurations of the textual source. This is because there are so many ways to represent the same concept through SysML v2 syntax. This problem is not unique to SysML v2, but rather is prevalent in all compiled programming languages as evidenced, for example, by the difficulty of reverse compiling a binary executable file back to its original source code. While it is possible to do this, it is inherently difficult, and, even if it is accomplished, there is always loss that occurs when reverse compiling. This greatly limits the utility of textual language for serialization.

Rather than reverse compile a SysML v2 model back to the original textual language for the purposes of serialization, the problem can be greatly simplified by serializing the Python objects built from the original source. Since the "Model," "Element," and "Behavior" classes were designed with serialization in mind from the ground up and can all be represented as nested dictionaries in addition to trees, JSON is well-suited as a serialization format. This can be done trivially through Python's built-in JSON package. Serializing to a flattened, tabular format, on the other hand, is somewhat more challenging. It requires designing the table itself in addition to custom coding, but the task is relatively straightforward. Furthermore, tables can be easily read and written to CSV (comma-separated value) files (or even Excel) through the Pandas DataFrame data structure [14], which is already a supported PySysML2 data transformation. Lastly, it is useful to have a simplified string representation of a model. While this is not serialization per se, it is related, in that it transforms a model into a human readable string. This is useful in the short term for debugging and development purposes but may have additional uses in the future.

Having discounted reverse compilation as a viable alternative to serialization, the JSON and tabular formats are prioritized for the first version of PySysML2, along with a string format suitable for quick model viewing. A reverse compiler for the language will be explored in future work. Implementation of the model as a tree makes this task straightforward. The underlying tree of the Model object can be traversed from the root to each leaf, and a hierarchical string providing an overview of the model can be constructed. Although it is static text rather than a navigable containment tree, it provides a clear, concise view of the model's structure.

A "data schema" is a blueprint that describes how data are organized. It is a detailed specification of a data set that facilitates interoperability between producers and consumers of those data. It is defined at the granular level, specifying fields, types, boundary conditions, relationships, triggers, procedures, and other contextual information about the individual data elements. A data schema should be defined to a sufficient level of detail such that a database can consume the described data (e.g., a relational database specification) or that an application can generate or receive serialized data (e.g., through file I/O or message queuing through formats like JSON or XML). Although it can be defined in a human readable form, a data schema is often defined in a format suitable for database software and other applications to read programmatically. A data schema should be specified according to a language or standard, i.e., well-defined and documented, accepted rules of grammar, and syntaxes that structure the description of the data set. Two schemas for serialization are currently supported for PySysML2, including a JSON and tabular CSV format, detailed next. The JSON schema is straightforward and automatic. It is derived from the class definitions of Model, Element, and Behavior and is generated from a simple call to the built-in Python JSON package. While this is useful for writing and reading model states, the lack of documentation due to volatility as PySysML2 evolves currently limits its utility for cross-application portability. This will change as development stabilizes, however, and the time investment of documentation becomes worthwhile.

In contrast to the JSON schema, the CSV schema is designed to support model portability between applications. PySysML2-specific fields necessary for complete JSON serialization were removed, leaving only fields common to modeling and analysis tools. The tabular fields were chosen based on their criticality to the model's organization and descriptiveness of model elements. The tabular export provides any data fields that may be useful for analysis in general. For example, a trade study of different alternatives for a component would require threshold and objective parameters, which may be recorded as values of attributes. A graph theory analysis would require relationship information between elements. This is captured in the table through the inclusion of names and Unique Identifier (UIDs) for parent and related elements, along with other information about relationships captured by the keywords used to define them.

5 Conclusions

The goal of this project is to build a pathfinder for a Python-based application, capable of interfacing with the SysML v2 modeling language through its textual specification. PySysML2 achieves this goal, while also providing a robust framework for continued development of the application. The extensibility of the grammar definition using the ANTLR4 language parsing workbench, for example, supports eventual inclusion of the rest of the SysML v2 specification. PySysML2 is currently limited because the full SysML v2 grammar has not yet been implemented in PySysML2. As new features of the SysML v2 textual language and its parent language KerML are released, the workflow supports both the adaption of existing rules to future changes and the inclusion of new rules.

This pathfinder project demonstrates the possibilities that open-source development and open data standards can realize for systems model usage. PySysML2 was made possible using open-source technology and by leveraging the SST's work. The decision of the OMG to prioritize model interoperability across tools in the SysML v2 RFP is truly a paradigm shift for systems modeling standards and tool development. We hope that our research inspires future work in related areas. PySysML2 is built from the ground up using computer programming and data science skills that most new engineering and computer science graduates possess as they enter the workforce. A dedicated team with access to higher-tier expertise could accomplish so much more. As digital transformation across government, industry, and academia continues to progress, issues with data portability and interoperability between tools will continue to pose significant challenges. Supporting open standards and formats, in addition to open source where appropriate, should be the cornerstone of the strategy to meet these challenges head on. This project demonstrates how the availability of open standards and formats, in conjunction with the availability of open-source technology, could lead to a future in which models are no longer siloed, locking away data from analysis. In that future, systems models become what they were intended to be in the first place: repositories of organized, integrated systems engineering data, easily accessible and discoverable by the workforce across both engineering and functional support fields, and compatible with powerful data science, simulation, and analysis tools available to the modern workforce.

Future development of PySysML2 will include the implementation of the RESTful API. Our draft tabular format can serve as the initial schema for a database view, which facilitates this connection. Our goal is also to finalize the open-source release of PySysML2 under the Apache Commons license and then seek out and engage with other organizations and developers working in the area of modeling, simulation, and analysis tool interoperability. There are several related open-source projects under development, and collaboration with that community is vital going forward. In addition to the open-source community, the defense sector is also highly active in this space. Many organizations, ranging from large contractors and federally funded research and development centers (FFRDCs) to small business innovation research (SBIR) firms, are pursuing solutions to the interoperability problems outlined at the beginning of this chapter.

Future features of PySysML2 include building out the grammar definition and accommodating the rest of the SysML v2 specification. The tabular CSV schema will be refactored for compatibility with the open HUDS format, currently under development by The Aerospace Corporation [8], in addition to a corresponding XML serialization schema. Further on the horizon, new capabilities include the implementation of the RESTful API and integration with the SysML v2 model repository and development of a reverse-compiling feature capable of generating SysML v2 textual code. Development of a basic simulation/analysis engine in PySysML2 could also be straightforward.

Acknowledgments The authors thank the Air Force Institute of Technology (AFIT), Department of Systems Engineering and Management, for providing the opportunity to conduct this research and also thank the Department of the Air Force Digital Transformation Office (DTO) at the Air Force Materiel Command (AFMC) for sponsoring this research. This research was partially completed under contract # GS00Q14OADU416 and task order # 47QFCA21F0013.

Disclaimer The views expressed in this chapter are those of the authors and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense, the United States Government, or KBR, Inc.

References

1. M. Bajaj, S. Friedenthal, E. Seidewitz, Systems modeling language (SysML v2) support for digital engineering. Insight **25**(1), 19–24 (2022)
2. Object Management Group, What is SysML? (2022) [Online]. Available: https://www.omgsysml.org/what-is-sysml.htm. Accessed 12 Dec 2022
3. Object Management Group, Systems modeling language (SysML) v2 request for proposal (RFP) (2 Dec 2017) [Online]. Available: http://www.omg.org/cgi-bin/doc.cgi?ad/2017-12-2. Accessed 8 Dec 2022
4. SysML v2 Submission Team (SST), 2022–10 SysML v2 release (2022) [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release. Accessed 8 Dec 2022
5. SysML v2 Submission Team (SST), SysML v2 API and services (2022) [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-API-Services. Accessed 8 Dec 2022
6. Apache Software Foundation, Apache license, version 2.0 (2004) [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0. Accessed 8 Dec 2022
7. Free Software Foundation, GNU general public license (2007) [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html. Accessed 8 Dec 2022
8. The Aerospace Corporation, Modeling tool integration plugin for Cameo Systems Modeler (MTIP-Cameo) (17 Oct 2022) [Online]. Available: https://github.com/the-aerospace-corporation/mtip-cameo-plugin. Accessed 8 Dec 2022
9. T. Parr, *The Definitive ANTLR4 Reference* (The Pragmatic Bookshelf, Dallas, 2012)
10. I. Model Driven Solutions, Introduction to the SysML v2 language textual notation (Oct 2022) [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/doc. Accessed 8 Dec 2022
11. Anytree Contributors, Anytree (14 Jan 2020) [Online]. Available: https://github.com/c0fec0de/anytree. Accessed 8 Dec 2022
12. S. Friedenthal, Requirements for the next generation systems modeling language (SysML v2). Insight **21**(1), 21–25 (2018)
13. Object Management Group, SysML V2: the next generation systems modeling language (2017) [Online]. Available: https://www.omgsysml.org/SysML-2.htm. Accessed 8 Dec 2022
14. Pandas Contributors, Pandas (22 Nov 2022) [Online]. Available: https://pandas.pydata.org/. Accessed 8 Dec 2022

Model-Based Verification Strategies Using SysML and Bayesian Networks

Joe Gregory and Alejandro Salado

Abstract In this chapter, the authors outline an approach to formally model verification strategies using Systems Modeling Language (SysML) in a way that enables the automatic generation of the corresponding Bayesian network. The approach includes the development of a verification metamodel that can be represented as a SysML profile. A notional example is included, in which a CubeSat verification strategy is produced in accordance with the SysML profile and a representative Bayesian network is created. Results from the Bayesian update are presented, and the impact on the SysML model is discussed. Further work will focus on the continued development of this metamodel, the integration of the plug-in to automatically generate the corresponding Bayesian network, and more detailed case studies.

Keywords Model-based systems engineering · Verification · Bayesian networks · SysML

1 Introduction

Verification activities, which usually take the form of a combination of analyses, inspections, and tests, consume a significant, if not the biggest, part of the development costs of large-scale engineered systems [1]. Verification occurs at various levels of a system's decomposition and at different times during its life cycle [1]. Under a common master plan, low-level verification activities are executed as risk mitigation activities, such as early identification of problems, or because they are not possible at higher levels of integration [1]. Therefore, a verification strategy is planned, "aiming at maximizing confidence on verification coverage, which facilitates convincing a customer that contractual obligations have been met;

J. Gregory · A. Salado (✉)
Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ, USA
e-mail: alejandrosalado@arizona.edu

D. Verma et al. (eds.), *The Proceedings of the 2023 Conference on Systems Engineering Research*, Conference on Systems Engineering Research Series,
https://doi.org/10.1007/978-3-031-49179-5_2

minimizing risk of undetected problems, which is important for a manufacturer's reputation and to ensure customer satisfaction once the system is operational; and minimizing invested effort, which is related to manufacturer's profit" [2]. Essentially, verification activities are the vehicle by which contractors can assess evidence of contractual fulfillment in acquisition programs.

In current practice, verification planning requires the production of several artifacts, such as verification requirements, test configurations, test procedures, and verification reports [1, 3–5]. Executing these activities often requires the maintenance of configuration logs (that are in sync with design artifacts) and verification control documents (that are in sync with requirement artifacts), among others [6]. Traditionally, all these artifacts are generated or controlled by a combination of independent or stand-alone documents. As with other document-based approaches in systems engineering [7], a document-based approach to verification planning generates several problems [6, 8], including information inconsistencies (due to multiple information sources and recurring updates and task repetitions), wasted effort in manually typing and transferring information across information sources, difficulty in identifying and controlling change propagation, and loss of a holistic view of the verification enterprise. Furthermore, the lack of modeling artifacts for verification planning restricts the opportunity to connect design and verification models to quantitatively assess the confidence yielded by the verification evidence. In summary, current document-based approaches to verification planning and assessment are inefficient, prone to inconsistencies, and unable to quantitatively inform about the confidence level on the verification status of the system of interest.

A model-based approach to verification planning and assessment can overcome these weaknesses [6, 8]. By connecting verification strategy models with other engineering artifacts within a model-based systems engineering (MBSE) environment or, more generally, by leveraging digital engineering (DE), we anticipate significant impacts on velocity and deployed capabilities by reducing the time needed to complete verification activities (since documentation efforts can be automated directly from the model), improving consistency in the design and development process (since information points to single sources of truth), improving the holistic evaluation of the verification enterprise (engineering teams that include design engineers, verification managers, and integration and test engineers work concurrently in shared models), and enabling the use of quantitative methods to compute the confidence in the correct state of the system of interest.

The concept of model-based systems integration (MBSI) was proposed to attempt to obtain the same benefits for later life cycle phases of systems development [9]. MBSI expands model-based methods beyond the early definition phases of a system and uses its capabilities to explore the implications of the architectural design on the future integration and testing of a system [9].

In this chapter, we outline an approach to develop a model-based verification planning and assessment approach in which verification planning is formally modeled and connected to quantitative models of verification strategies and verification evidence as well as to models of system design. Verification planning will be modeled using the behavioral and structural modeling features offered by Systems