PYSYSML2:
BUILDING KNOWLEDGE FROM MODELS WITH SYSML V2 AND PYTHON

KEITH L. LUCAS

Civilian, Department of the Air Force, Air Force Institute of Technology

AFIT/ASE/ENV Dept. of Systems Engineering & Management, Master of Science in Applied Systems Engineering

Faculty advisors:
Lt Col Thomas Ford, USAF, Ret., Ph.D.
Lt Col John Situ, USAF, Ph.D.

September 6, 2022

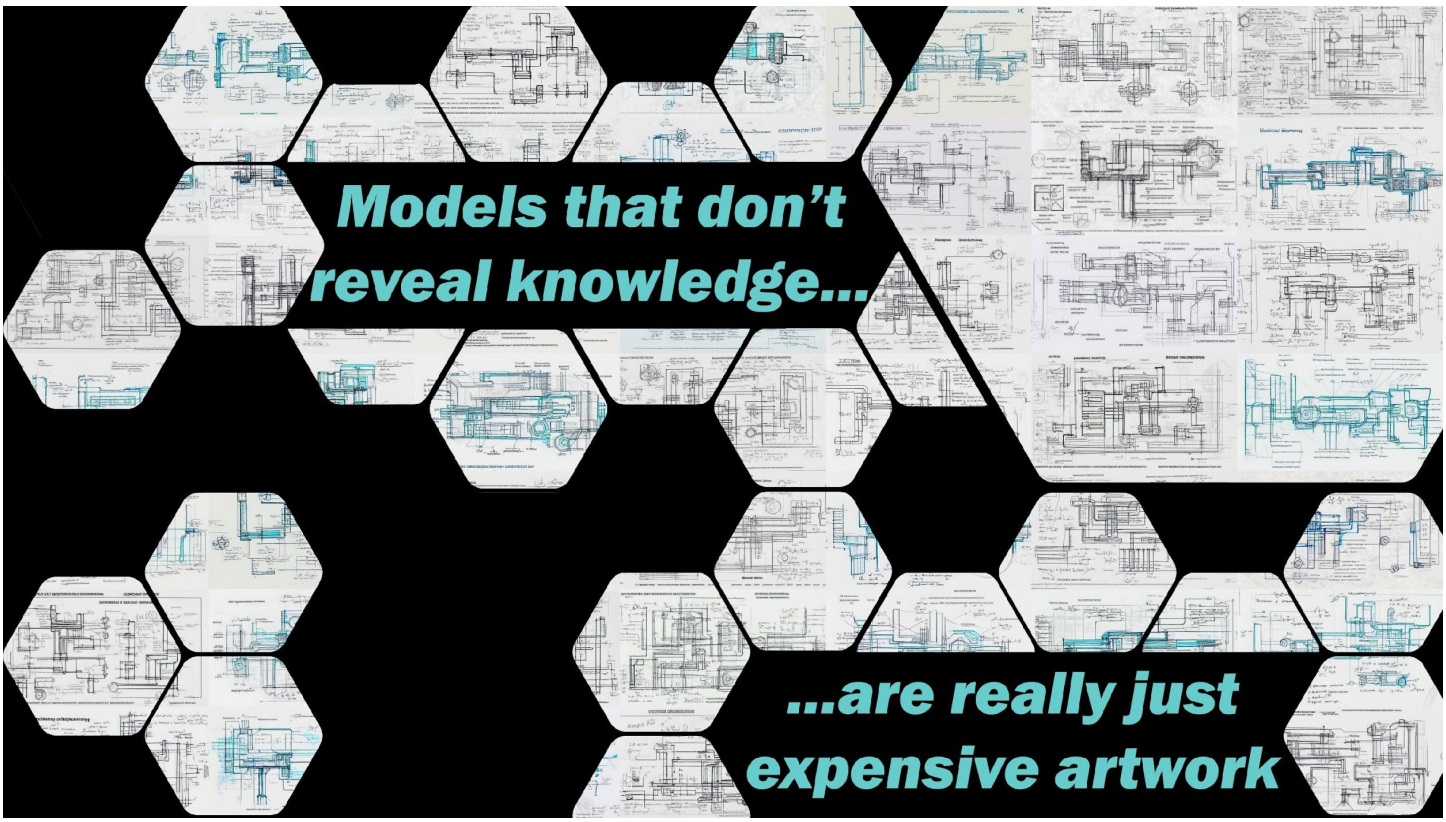
ABSTRACT

The promise of Model Based Systems Engineering (MBSE) to revolutionize decision making in Department of Defense (DoD) acquisition and sustainment projects is held back by a lack of integration with data science and analysis tools. Without the ability to bring the power of these tools to bear on MBSE models, knowledge that would empower decision makers and stakeholders remains hidden beneath a mountain of data and walled off behind poorly documented, often proprietary data formats. The digital thread is broken at the border between modeling and analysis. The forthcoming next generation of the Systems Modeling Language, SysML v2, seeks to remedy this problem by building interoperability into the requirements from the ground up, realized through the specification of a textual modeling language and RESTful API. A graphical specification is also defined, but graphical SysML v2 is derived from the textual model. This means that a SysML v2 model's authoritative source of truth will not reside in a tool specific format, but rather something more like source code: a human readable text file, formatted and structured under a well-documented, open standard. This is a potential gamechanger, pointing towards many exciting possibilities for MBSE. PySysML2 is a pathfinder project to prototype an open-source software application that interfaces the SysML v2 textual modeling language with the Python-based ecosystem of data science and analysis tools. This paper provides an overview of the PySysML2 project, the problems it seeks to solve, and a roadmap for continuing its future development.

Keywords: SysML, SysML v2, Python, Data Science, MBSE, Modeling and Simulation, Digital Thread

Author's Contact Information:

Keith L. Lucas
United States Air Force Institute of Technology, Wright-Patterson AFB, OH 45433-7765
keith.lucas.3@us.af.mil, keith.lucas@maatlockenterprise.com
Telephone: 703-928-7406



INTRODUCTION

The Department of Defense (DoD) is undergoing a systems engineering paradigm shift. Modern defense systems have increased in complexity by orders of magnitude, while development and acquisition schedule requirements have become more urgent. This is driven by many factors, but the two most pressing are, number one, the need to maintain sufficient capability margin over near-peer adversaries of the United States as they continue to advance, and, number two, the need to keep pace with advancing technology.

The traditional waterfall, document-based systems engineering approach that has driven the development of our nation's capabilities to this point is beginning to see diminishing returns in the face of these challenges. Traditional long-lead times for projects are becoming a luxury that the DoD can no longer afford. By the time a system can be deployed, if not even before, it runs the risk of becoming obsolete. Even if time were not such a scarce commodity, though, the exponentially increasing complexity of defense systems is rapidly overwhelming the workforce's capacity to manage the technical baseline effectively.

The 2018 DoD Digital Engineering Strategy recognizes this and outlines a multi-pronged approach for the DoD to leverage cutting-edge digital capabilities to overcome these challenges [1]. This project will focus on an important pilar of digital transformation: the synthesis of Model Based Systems Engineering (MBSE) and modern data science capabilities for analysis and management of the technical baseline at scale. The *PySysML2* tool will be an early pathfinder for integration between the SysML v2 modeling language under development by the Object Management Group (OMG) and the vast Python data science ecosystem.

PySysML2 will be free and open-source, available on GitHub. It is hoped that 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.

THE PROBLEM

Over the last decade, and especially since the release of the Digital Engineering Strategy in 2018, the DoD has been moving away from the traditional document-based approach of systems engineering and towards the use of digital models to manage, maintain, understand, and interrogate the technical baseline. While large, well-heeled programs with access to the necessary resources are well on their way to making this transition, much of the acquisition and sustainment community is falling behind.

Among many obstacles to overcome, some of the most difficult are acquiring and deploying the necessary software tools and Information Technology (IT) infrastructure, while onboarding and training the workforce to use those tools. Even after the workforce is onboarded, trained, and building models in the right software on suitable hardware, however, turning models into useful knowledge is proving more difficult. This issue will be laid out more fully in subsequent sections. The crux of the problem, though, is that there are few reliable, well maintained and documented interfaces between models developed in different toolsets—let alone between modeling tools and simulation and analysis tools. The digital thread between related models and between models and analysis is broken at the interfaces between tools.

THE INTERFACE BETWEEN MODELING TOOLS AND ANALYSIS TOOLS IS BROKEN

The most capable MBSE software applications (e.g. Sparx Enterprise Architecture, IBM Rhapsody, and the MagicDraw / Cameo suite of tools by Dassault Systems) are expensive and difficult to acquire and deploy for many DoD organizations. Although more accessible diagramming tools like Microsoft Visio and similar software may be used to model without these specialized MBSE tools, those solutions become ineffective and unmaintainable at any significant scale of complexity. Furthermore, tools like Visio only allow the architect to compose static diagrams that cannot be integrated with external software that expands modeling, simulation, and analysis capability.

Tools like Cameo, on the other hand, are intended to support this integration. In practice, though, integrating models built in these tools with the standard workbench of data science, simulation, and analysis tools (e.g. the Python ecosystem, RStudio, MathWorks MATLAB / Simulink, etc.) is quite difficult. Moreover, it often requires additional purchases of costly plugins and as much or more effort to set up than the original modeling endeavor—just to get started. Even when the plugins are available, shifting Application Programing Interfaces (APIs) on either side of the connection (i.e. the Modeling Tool Side and the Analysis Tool Side) often drive additional troubleshooting and configuration just to achieve the basic, advertised functionality. And oftentimes that functionality is not enough, driving yet more highly specialized engineering efforts to perform the required analysis.

The unreliable APIs, however, are only one part of the problem. The more fundamental problem is the *lack of an open, standardized serialization[1] format with which to record and share MBSE models.* The authoritative source of truth for these models are files stored in tool-specific, proprietary formats. While it is possible to export models into formats suitable for limited sharing, both between different modeling tools and between modeling and analysis tools, the current methods are not reliable and are barely usable. Before going further, it is worth examining the roots of this problem, as its cause points towards a potential solution. For SysML, one of the most widely adopted MBSE modeling languages, the problem stems from the foundations of the language specification itself.

SYSML V1'S SPECIFICATION AS A GRAPHICAL MODELING LANGUAGE LIMITS ITS INTEROPERABILITY

OMG is the body that governs both the Unified Modeling Language (UML) and SysML. OMG defines SysML v1 as "a general-purpose **graphical** (*emphasis mine)* modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities" [2]. SysML v1 (version 1.6 as of this writing in 2022) is an extension of UML 2 (as shown in Figure 1), itself a graphical modeling language tailored especially to software engineering.

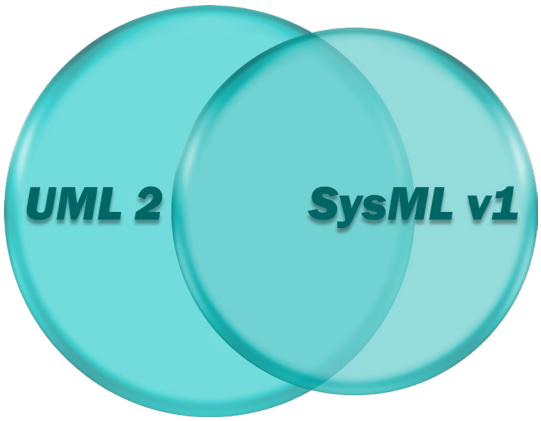


Figure 1 - UML and SysML v1
Adapted from the OMG SysML overview [2]

The power of MBSE applications lies in the fact that they harness that graphical language to a computational engine under the hood that adds precise engineering context to the diagrams, while also maintaining the elements in a relational database. This is what makes dynamic model architecting and interaction possible, and it lies at the heart of MBSE's advantage over traditional, document-based systems engineering approaches. For each of these applications, however, that computational engine under the hood is both proprietary and largely opaque to the user. For example, while models can be exported to image files (i.e. static diagrams like those created in tools like Visio), they can only be dynamically accessed and analyzed fully from inside the tools in which they were architected. There is often even difficulty when working with models in different applications of the same vendor tool suite.

[1] Serialization is the process of transforming a virtual construct, usually an object in computer memory, into text, either human readable characters or bytes, so that it can be placed in storage for later reuse or shared with other applications

Although there is some limited interoperability between the largest MBSE tool vendor suites, it is, for all intents and purposes, not feasible to transfer complex models from one to another without significant overhead and the near certainty of losing model fidelity. This issue is further compounded when combined with the API problem mentioned earlier, making effective and efficient analysis of models nearly impossible. Whether this was the intent of the MBSE tool developers or not, the fact of the matter is that once an organization commits to one of these applications, they are essentially vendor-locked to that tool

The primary driver of this issue is the fact that SysML (and most other modeling languages) *are only defined graphically*. While textual expression of SysML v1 models is possible, the language does not have a first-class textual specification, relying on the Action Language for Foundational UML (ALF) to do the heavy lifting. While this has driven significant effort through research and development on the part of MBSE tool vendors to build the required computational engines, each company has done it differently, and the result is the stove-piped situation we have today. Tools capable of modeling SysML v1, for example, all implement some form of ALF under the hood, but this is either completely opaque to the user or, as is the case for Magic Systems of Systems Modeler, paywalled behind additional, costly plugins.

The second-class status of textual expression in SysML 1.6's specification has also driven tool vendors to develop their own interface standards. The primary method for serializing SysML v1 models for export between tools, for example, 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. The problems associated with XMI are many, but the two listed next are showstoppers:

1. *There is no definitive documentation for XMI and its tool specific extensions*. Any organization that works with XMI must reverse engineer the standard from examples. *Many organizations have spent many person-months, perhaps person-years, to crack this code*. They have done this in the hopes of being able to perform basic analysis on models through interface with standard data science tools—a capability many assume to come "out of the box."

2. *XMI does not represent SysML completely and is error prone*. Porting a model from one tool to another through XMI results in loss of fidelity. For complex systems in which continuity of the technical baseline over their lifetimes is of utmost importance, this is a showstopper for model portability and integrity. As models scale in complexity, porting from one tool to another becomes infeasible and, for all intents and purposes, vendor-locks the organization into the original modeling tool.

DRIVING TOWARDS THE SOLUTION

SYSML V2'S SPECIFICATION AS A TEXTUAL MODELING LANGUAGE ENABLES ITS INTEROPERABILITY

The problem of model portability and interoperability between applications was at the forefront when OMG drafted the Request for Proposals (RFP) for SysML's next generation, SysML v2. 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." That objective is supported by several related requirements throughout the RFP [3]. A proposed specification from a consortium of members of the systems engineering community from across 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].

First, the SST defined a basic language called Kernel Modeling Language (KerML), which includes abstract elements and concepts common to many modeling languages. SysML v2 was then extended from KerML, as shown in Figure 2. The SST's submission 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" [5]. SysML v2 has not only a graphical specification like SysML v1, but also a first-class textual specification. Similar to a programming language, this textual specification is built upon a rigorous and well-defined syntax and set of semantics.

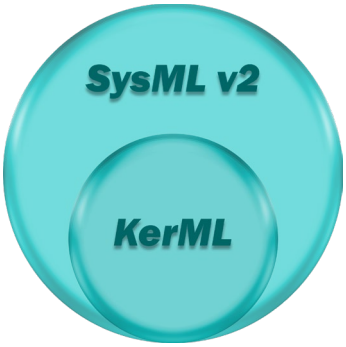


Figure 2 - SysML v2 and KerML

Rather than leave the development of an appropriate SysML v2 computational engine framework to the MBSE tool vendors, the SST built it as a feature of the specification from the ground up. While one can be nearly certain that the tool vendors will build their own parsers, lexers, and underlying relational database structure, those will, presumably, all be harnessed to a common syntax and set of semantics for SysML v2. Furthermore, this would be the case for all future textual modeling languages based on KerML.

In addition to the textual language, the proposed SysML v2 specification also includes a Representational State Transfer (REST, also known as RESTful) API. Although this project focuses on the textual language of SysML v2, it is worth considering some of the implications of this before continuing. The principles of REST are foundational to the World Wide Web. Simply put, REST provides a layer of abstraction between servers and clients, enabling them to transfer data agnostically of one another's unique implementation, provided they agree on the medium of communication. In the case of the web, that medium is the HyperText Markup Language (HTML). In the case of servers hosting SysML v2 model repositories, the medium of communication is the JavaScript Object Notation (JSON) schema for queries and responses described in the API documentation.

The SST states in its proposal that the API will "provide standard services to access, navigate, and operate on KerML-based models, and in particular 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" [6].

The decision to define both a textual modeling specification and a RESTful API for SysML v2 ensures that model interoperability and accessibility will remain core features of the language as it matures. This development is a critical for the utility of MBSE in the DoD. It 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. The authoritative source of truth for models built in SysML v2—and any other future KerML based language—will be accessible through a platform and tool-agnostic RESTful API. Moreover, their definition as textual source code means that models can be maintained with modern versioning tools like Git. Additionally, in the event a primary MBSE tool is unavailable, the model can always be read and edited through a simple text editor or full programming Integrated Development Environment (IDE) tool (e.g. Visual Studio Code).

PYSYSML2 PATHFINDER PROTOTYPE

This project seeks to demonstrate how the SysML v2 textual language can interface with modern, open-source data analysis tools widely used in the scientific, academic, engineering, and defense communities. Specifically, a straightforward approach for integrating SysML v2 with the Python programming language will be demonstrated through the development of the open-source application, PySysML2.

The PySysML2 project seeks to take advantage of the radical, paradigm-shifting opportunity presented by the initial SysML v2 Release. The goal is to open just a bit wider the gates that separate the data science and engineering workforce from the systems they are tasked with developing, acquiring, and analyzing.

PYSYSML2

This section documents the initial development of PySysML2, highlighting its design philosophy and providing context for the software engineering decisions made.

OPEN-SOURCE DEVELOPMENT PHILOSOPHY

The driving force behind PySysML2 is the momentum started with the SysML v2 RFP in support of interoperability and open standards. Additionally, the current difficulty with interoperability between SysML v1 models and standard analysis tools has driven an immediate need for this capability. The subsequent open sourcing of the prototype SysML v2 Release has brought together stakeholders from across industry and academia to take part in its development. It is the author's belief that this will result in a more robust and usable MBSE language. While powerful commercial MBSE applications will always have an important role to play—and indeed, applications developed by individuals and small teams are unlikely to replace them—the development of open-source MBSE tools and open APIs will accelerate adoption of MBSE and help realize its full potential.

PySysML2 will be free and open-source, released and licensed under the Apache Commons 2.0 License [7]. Apache Commons was chosen to encourage broad use of the tool and to support incorporation in or adoption by other projects. Note that the SST has chosen to license the prototype SysML v2 Release under the General Public License (GPL) [8]. This license was considered for PySysML2, but it 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 was not necessary to carry the GPL forward. Additionally, other related projects, for example the HUDS XML Cameo plugin by the Aerospace Corporation, have released their source under Apache Commons [9]. HUDS XML is an alternative to the problematic XMI serialization format for SysML v1 mentioned earlier.

The intent is for PySysML2 to be available for use as is or to be adapted and incorporated into other related projects. It is hoped that this project serves to inspire future development in the open-source MBSE capability space with respect to model integration with data analysis applications, and to the Python data science ecosystem in particular. Releasing PySysML2 under Apache Commons 2.0 will support this goal.

DEVELOPMENT IN PYTHON WITH THE ANACONDA DISTRIBUTION

PySysML2 will be developed mainly in Python, with some supporting features and libraries in other languages. It will be developed using Miniconda, a minimal subset of the widely used Anaconda Python Distribution. This has many advantages, but the primary advantage is that Anaconda includes most required modules and libraries out of the box. It includes data science libraries like NumPy, Pandas, Keras, and TensorFlow, but also general support utilities as well. 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 particular version[2] 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 will support easy installations across multiple operating systems and configurations, while clearly notating all dependencies[3].

PYSYSML2 USE CASES

The PySysML2 application will initially fulfill the three use cases shown in Figure 3. Aside from these three, the nearest one on the horizon is to build an initial SysML v1 to v2 refactoring capability. Although that use case and more are under consideration, the following are foundational to most, if not all, future use cases of interest.

[2] As of this writing, the PySysML2 prototype uses Miniconda version 4.12.0, which ships with Python 3.9.7.
[3] For a list of all dependencies, see Appendix A: PySysML2 Dependencies

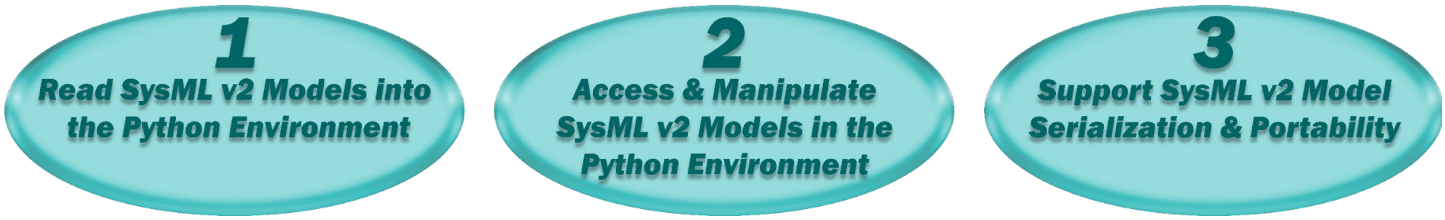


Figure 3 - PySysML2 Use Cases

USE CASE 1: READ SYSML V2 MODELS INTO THE PYTHON ENVIRONMENT

The most fundamental capability of PySysML2 is the ability to read models built in the SysML v2 textual language into the Python computing environment as objects that can be explored, analyzed, and manipulated. All future use cases of the application depend on the realization of this capability.

USE CASE REFINEMENT

The SysML v2 textual language is very similar to a programming language like C++ or Java, and therefore building a reader function for it is non-trivial, more akin to building a computer language compiler. When developers use the word "reader," however, they are usually referencing the straightforward reading of data into memory without any additional context for what the data represents. Most programming languages, for instance, come equipped to handle file I/O, usually by a function referred to as a "reader." When a file is read by such a function, however, it is usually stored simply as an array of bytes in memory ready for additional processing. On the other hand, a specific type of file "reader," say one for Comma Separated Value (CSV) files, is slightly more complicated, since it provides context to specific byte configurations, e.g. those representing commas. Because of this additional capability, a CSV "reader" is more appropriately called a parser.

In contrast to a reader, a parser has additional logic that interprets what it is reading and assigns specific meaning and context to the data. Although PySysML2 does read the content of SysML v2 textual models, it must also interpret that content if it is to do anything more meaningful than storing it as an array of characters. Use Case 1, therefore, drives the development of a parser rather than a reader. Use Case 1 now becomes:

Parse the SysML v2 textual language so that models may be instantiated and contextualized in Python.

TRADESPACE ANALYSIS

The first trade was choosing to build a full parser over a simple reader for the textual language. There are many approaches to building a language parser. The primary decision, however, is whether to build the parser code from the ground up or to use a pre-existing workbench of language and grammar tooling. On one hand, building from scratch enables the most flexibility to handle any nuances of the grammar. On the other hand, using pre-existing grammar frameworks and tooling significantly reduces development time, while supporting extensibility and maintainability of the parser.

The SST have made a great effort to define a grammar for SysML v2 that is intuitive and straightforward, so the benefits of a bespoke parser are outweighed by the benefits of using a grammar implementation tool. The application ANother Tool for Language Recognition 4 (ANTLR4) is a powerful, open-source tool for developing parsers from a defined grammar. It stands out over other tool-sets because, number one, it is widely used across industry and academia. Twitter, for example, uses it as their primary query parser that interprets searches by users [9]. Number two, though, it abstracts the coding of the parser itself from the definition of the grammar. This means that the primary artifact to develop, maintain, and extend is the grammar definition itself. Meanwhile, the parser—which may be thousands of Source Lines of Code (SLOC)—is autogenerated from the grammar. Once a grammar is defined, ANTLR4 automatically generates a corresponding parser in a variety of programming languages (e.g. Java, C++, C#, Python, Swift, and others). Because SysML v2 is still under development, one would expect that it will continue to change for some time to come. For these reasons, the parser at the core of PySysML2 is built using ANTLR4.

IMPLEMENTATION

Before continuing with the explanation of the implementation, it is worth reviewing some terms of art for language parsing. These words may be familiar to the reader in other contexts, but they have precise meaning in the context of parsing languages, which is critical to the realization of Use Case 1.

Table 1 - Grammar Related Terminology

Adapted from the Definitive ANTLR4 Reference, Terence Parr [9]

Syntax	Rules that govern membership of a phrase in a language
Grammar	A specification that defines the syntax (or rules) of a language for each potential phrase
Character	A configuration of bytes that specifies a single letter, number, or other mark
Token	A combination of characters into words, i.e. symbols, that have specific meaning and can be combined into valid phrases of the language. Note that, syntactically, a *token*, *i.e. symbol* has a specific meaning in the context of the language, whereas *characters* only have meaning in terms of the tokens they can express
Tokenizing	The process by which characters are grouped into tokens
Lexer	An application or function that tokenizes characters. The lexer groups characters 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	An application or function that interprets the token stream generated by the lexer, assigning meaning to phrases of tokens according to the language grammar
Tree	A data structure in computer memory characterized by a root element that points to one or more child elements, which in turn may point to other child elements of their own
Parse Tree	A tree data structure that records the parser's interpretation of the phrases defined in the token stream

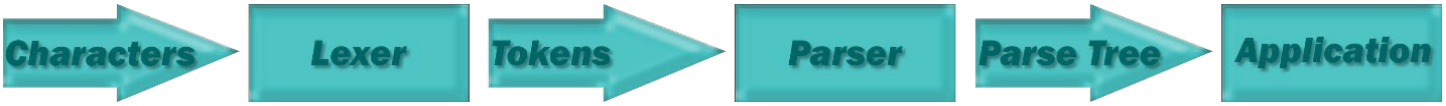


Figure 4 - Grammar Parsing Workflow

Following Figure 4, the first step in interpreting a language is to read in 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.

Once it 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.

The parse tree is the output of the parser and serves as the raw interpretation of the source code being recognized. This data structure is the foundation for any application that executes behavior specified by the source. PySysML2 interfaces with the parse tree through a special class called *Visitor* that retrieves all required information from the tree as needed. Table 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. The *Parser* code for PySysML2 comes in alone at around 2400 SLOC. It is by far the most complex and lengthy part of the code base. Offloading that complexity to a grammar workbench like ANTLR4 will yield massive development and maintenance efficiencies as PySysML2 evolves. Subsequent sections describing PySysML2's implementation will expand on the *Lexer*, *Parser*, *Visitor* pattern through a demonstration on a notional but non-trivial SysML v2 model. In preparation for that, though, Figure 5 demonstrates the parsing process described above with a simple example.

ANTLR4 is built around a specialized "meta" language suited for describing the grammars of other programming languages [9]. Using the Antlr4 meta language, a grammar specification for SysML v2 was constructed, derived from the documentation released by the SST [10]. From this grammar specification, ANTLR4 generated the source code required to interpret textual language. The grammar specification was designed to be extensible. As PySysML2 evolves, this extensibility will support growth beyond the initial subset of the language chosen for this project, eventually including the whole specification. It will also support adaptability, as the language will inevitably change before final adoption of SysML v2 by the OMG.

It is worth noting that, while SysML v2 is straightforward as far as languages go, the source code required to interpret it is complex, around 2,800 SLOC to accommodate just the initial base language elements. The ANTLR4 grammar specification that auto-generates that code base, however, is less than 100 SLOC, excluding documentation, and will grow much more slowly than the source code generated. This is a massive boost to PySysML2's maintainability and extensibility.

PROCESS

The process by which the SysML v2 textual language is interpreted by the application is demonstrated in Figure 5. The two primary components that execute this process are the *Lexer* and the *Parser*, both of which work in sequence to read the contents of the file and then build a corresponding model object in memory.

First, a small model, that of a basic computer system in this case, is defined in a text file. It is read into memory as a character stream and stored in an array with all whitespace removed (note that whitespace is ignored in the SysML v2 textual language and can be disregarded).

Next, the *Lexer* evaluates characters, grouping them into tokens with meaning in the context of the grammar's syntax. The tokens are color coded in the figure, with keywords like "package" and "part def" colored blue; organizational symbols like braces, single quotes, and semicolons colored purple; and identifiers like "Peripherals" and "RAM" colored gray.

Next, the tokens are interpreted based on the syntax of grammar specification. The *Parser* builds a data structure called a parse tree that records how component phrases of the textual model were recognized and organized. Step 4 leads to Use Case 2, accessing and manipulating SysML v2 models in the Python environment. To get there, however, further interpretation and refinement of the parse tree by the application is required through the *Visitor* class and specific classes designed to implement a SysML v2 model. Those details are discussed next in Use Case 2.

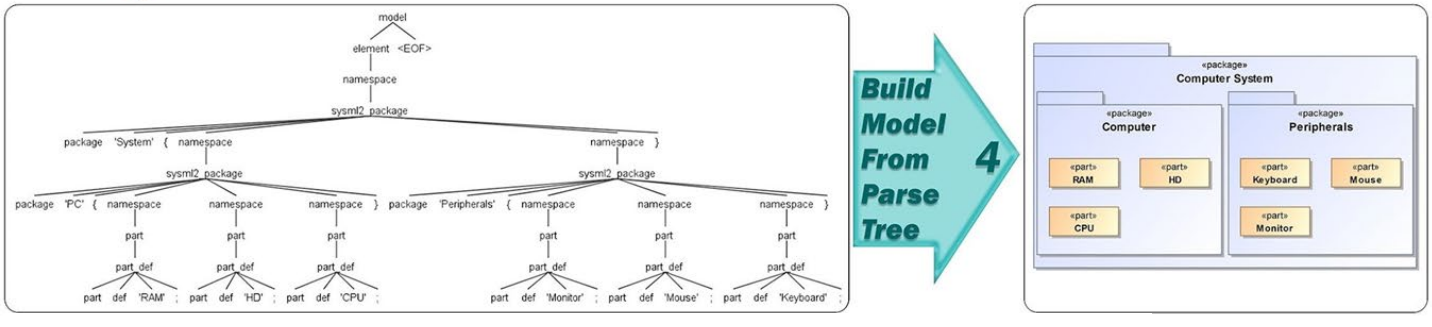


Figure 5 - Language Parsing Process

Table 2 - SysML v2 Grammar Source, ANTLR 4


```
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: '/*'.*?'*/';
```

USE CASE 2: ACCESS AND MANIPULATE SYSML V2 MODELS IN THE PYTHON ENVIRONMENT

Now that PySysML2 can parse the language, the task becomes to build the capability to access and manipulate those models. That use case is explored further in this section, and the implementation of SysML v2 within PySysML2 is explained.

USE CASE REFINEMENT

Models mean nothing if they cannot be accessed, analyzed, and evaluated by the appropriate technical workforce. This workforce includes program managers; cost and operations research analysts; engineers of all kinds, including electrical, mechanical, software, and systems engineers; data scientists; and many other groups from across the Science, Technology, Engineering, and Mathematics (STEM) fields. Furthermore, this workforce is in high demand and in diminishing availability across the research, industry, and defense sectors that compete for talent. As the DoD seeks to realize the benefits of digital transformation, it is imperative that the models and authoritative sources of truth that make up the technical baseline of our nation's defense capabilities be usable and understandable by current and incoming members of the technical workforce.

Additionally, they must be compatible with the tools and skills that the technical workforce is bringing with them from academia and industry. Very few college graduates come into the government with skills in MBSE, let alone with specific MBSE tools—they do, however, come equipped with sophisticated and powerful data analysis skills common across the STEM fields. The task of PySysML, then, 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. Use Case 2 now becomes:

Transform the data of a SysML v2 model so that it can be structured as trees, graphs, data frames, and multi-dimensional arrays.

TRADESPACE ANALYSIS

Although Python and its ecosystem of data science modules were chosen early on, the question of how best to implement SysML v2 in Python remains. The SST has done an enormous amount of work building sophisticated tooling for the language in Java, including a full implementation of the SysML v2 grammar parser in that language. Additionally, they have built robust support for accessing model content through a RESTful API. At this point, several approaches present themselves.

One approach would be to populate a database with model content after the textual source code is handled by the PySysML2 parser, adhering to the API. Doing this would require significant engineering, though, and, while PySysML2 will eventually support database integration through the API, it is not necessary to realize Use Case 2. Keeping the RESTful API in mind during development, however, will be important for future features and compatibility.

Another approach would have been to leverage the source code in the SysML v2 Release through a Python to Java interface. This second approach was discounted, primarily because of the limitation that it requires using the Java implementation of Python, Jython, which has only been updated to Python 2.7 and is not under active development. It would also limit compatibility with the latest updates to Python data science modules, thereby largely defeating the purpose of PySysML2. Finally, incorporating code from the SST's SysML v2 Release would require adoption of the GPL license, which may limit adoption of and contributions to PySysML2.

Having considered the previous two approaches, the decision was made to build PySysML2 in as much pure Python as possible, utilizing the latest stable release. Doing this will provide the most flexibility as the application matures.

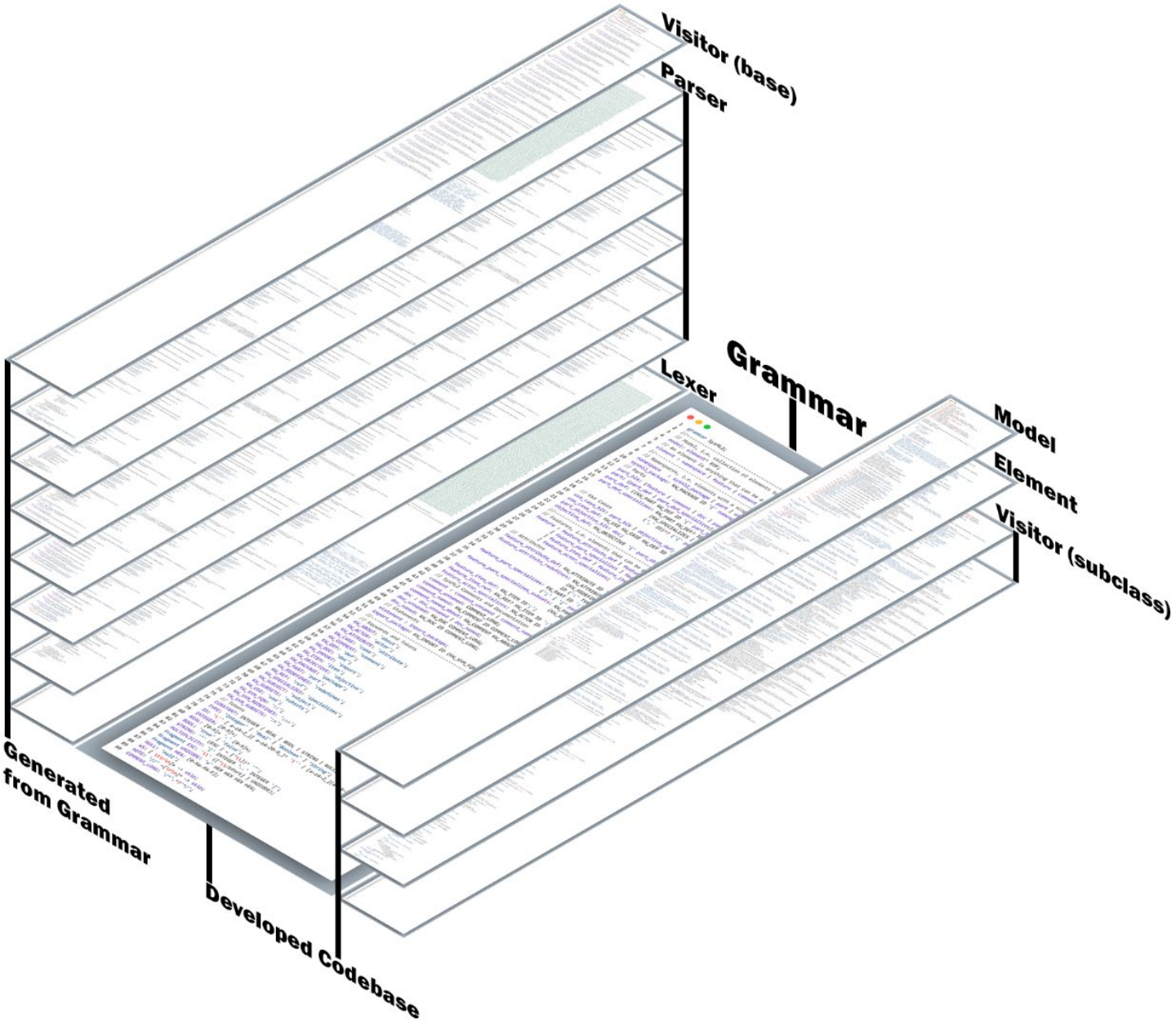


Figure 6 - PySysML2 Codebase

As mentioned previously, the *Lexer, Parser, and Visitor* classes are automatically generated by ANTLR4 from the grammar. Figure 6 above shows the relative size and complexity of these generated classes, which are foundational to PySysML2. The classes generated from the grammar, shown on the left-hand side of the figure, constitute approximately 2800 SLOC, 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 textual language has been taken in by the parser, all its data exists in a parse tree, as described under Use Case 1. This drives the architecture quite naturally towards a tree as its foundational data structure.

A PySysML v2 model will be represented by a *Model* class in Python. A *Model* will be defined as a root node that points to the first element of a SysML v2 model, which then may point to arbitrary sub elements, and so on. Since a *Model* will be a tree of *Elements* and *Behaviors*, this means that 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***, and so on. 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" [11]. Model elements will be 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. Fortunately, this has already been done for us, and we can make use of the popular Python module *Anytree* [12]. The base tree class in hand, the *Model*, *Element,* and *Behavior* classes will now be defined as customizable sub classes extending *Anytree*. Additionally, the PySysML2 tailored *Visitor* subclass will 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 the likes of Numpy, Pandas, SciPi, and others will be straightforward. Next, the software architecture is discussed more in depth, in addition to the use of Test-Driven Development to validate the code.

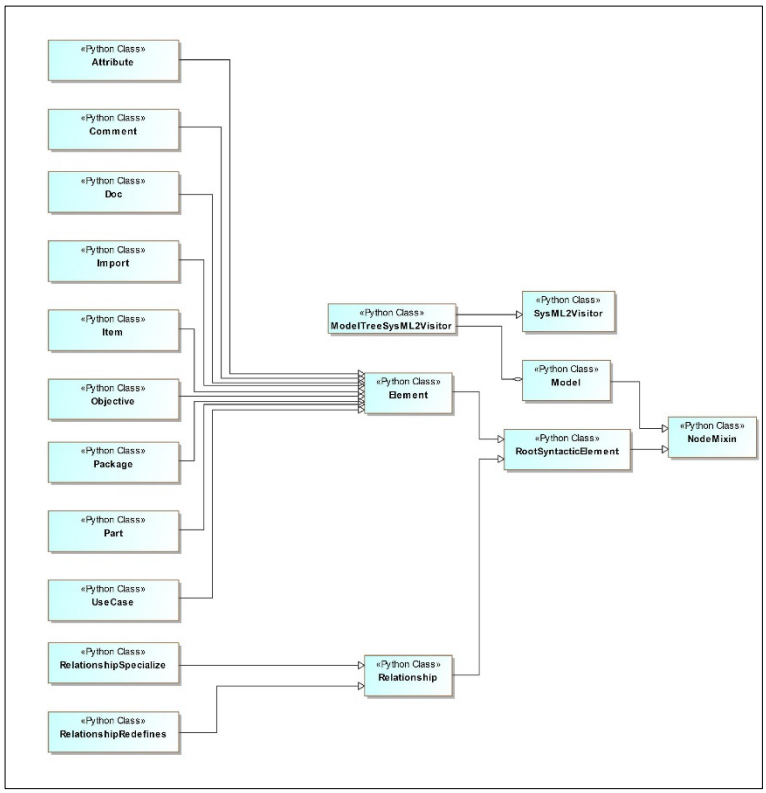


Figure 7 - PySysML v2 Inheritence Structure

Figure 7 shows the inheritance structure of the primary, user-facing PySysML2 classes: *Model, Element,* and *Behavior*. As mentioned, all three inherit from the tree data structure 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. This class, then, has all the fields that *Elements* and *Behavior* 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 will only support basic elements of the language, enough to build useful but still simple models. Future iterations, though, will gradually incorporate the rest of the specification.

Before writing any code that would parse the SysML v2 textual language, a SysML v2 model is required for validation. Since only basic elements of the language will be incorporated at first, a simple system will be modeled to begin. Figure 8 shows a concept schematic for a notional integrated system called the Tabletop Roleplaying Game (TTRPG) eToken. The primary use case for this system will be to display a user defined image on a small, round LCD screen that can be used as a customizable game token in a tabletop game. It will be driven by a micro-CPU controller with Wi-Fi and/or Bluetooth capability and battery powered. The user will be able to 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. Figure 9 shows a virtual implementation of this concept on a Virtual Tabletop (VTT) that supports remote gaming. As the COVID-19 worldwide pandemic begins to subside, more players return to in person games. The eToken is intended to provide a similar feel to the VTT experience, while providing the flexibility that players have come to enjoy when gaming in a virtual space.

This system strikes a suitable balance between simplicity and substance to serve as an effective test model for PySysML v2. Table 4 provides a side-by-side comparison of the TTRPG eToken SysML v2 textual model with the grammar specification shown in Table 2. Figure 10 presents a view of the parse tree once the model is read into PySysML2, and Figure 11 presents a graphical representation of the SysML v2 model built from that parse tree. Note that for the sake of space and readability, only the Structure package is shown.

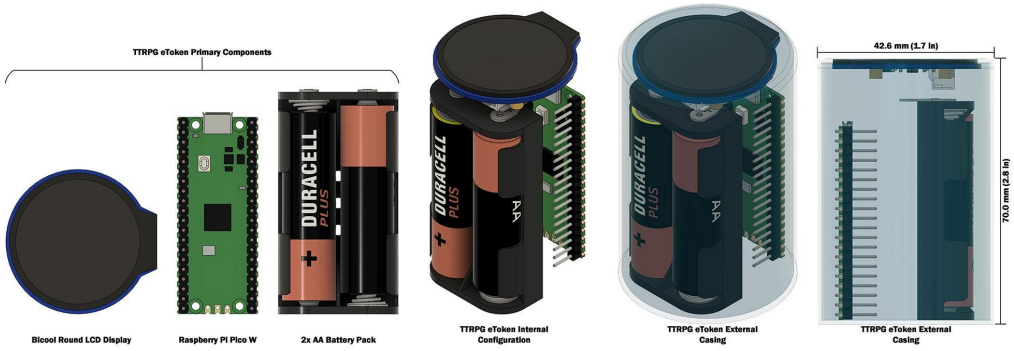


Figure 8 - Notional System: TTRPG eToken

Drawings down in Autodesk Fusion 360 with assets from GrabCAD.com

Figure 9 - Screenshot from Virtual Tabletop RPG Game

PROCESS

Next, the PySysML2 processing chain is described.

1. The user accesses PySysML2 through a command line interface. The first step is to point the application to a SysML v2 source file, in this case the TTRPG eToken model, as shown in Table 4
2. The application parses the model source through the process shown in Figure 5 under Use Case 1, using the *Lexer* and *Parser* code referenced in Figure 6 that was generated from the grammar shown in Table 2
3. Next, the model is built in memory from the PySysML2 classes shown in Figure 7
4. A *Model* object is instantiated, and its root node defined
5. *Model* then interacts with the TTRPG eToken SysML v2 model through the *SysML2Visitor* class, which walks the parse tree
6. *Element* and *Behavior* objects are instantiated by *SysML2Visitor* and saved in a temporary array in *Model*
7. After walking the parse tree, *Model* loops through the array of *Elements* and *Behaviors*, building out the model tree with nodes pointing to their children and back to their parents. This tree structure is shown in Figure 10
8. Now that the *Model* object is fully instantiated, the TTRPG eToken model, shown as a SysML v2 diagram in Figure 11, may now be accessed and manipulated as the user sees fit in their data analysis environment. Note that the diagram only shows the contents of the Structural package due to space constraints
9. 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 suitable for data science and analysis. The end of this document outlines plans for future built-in data analysis capabilities in addition to these basic ones, but the following table lists the currently implemented data transformations in PySysML2
10. Once PySysML2 completes a run, multiple output files are generated: a tabular version of the model, a JSON 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 application was developed and validated using Test Driven Development (TDD) based on the following workflow. This process was validated through test cases based on the notional TTRPG eToken model. Specifically, that 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 form, to the original textual code. Lastly, the visual graph was inspected and compared to the model hierarchy.

Table 3 - Data Structure Transformations

Multidimensional Array	*Numpy Array*. The uses of multidimensional arrays are many, ranging from statistical, engineering, and numerical analysis to many types of modeling and simulation. This data structure is the primary interface to the majority of Python based data science tools
Dataframe	*Pandas Dataframe*. This is a specially formatted multidimensional array that implements spreadsheet functionality. There are many use cases, but the one currently implemented in PySysML2 is the transformation of a SysML v2 model into a relational table that can be written and read as CSV file. This provides a powerful, consolidated model view, and opens the door to many potential use cases
Tree	*Anytree*. The utility of trees with respect to grammar parsing has already been established, but they are also ideal for any number of data science tasks. One particularly useful capability is the ability to store a model in a hierarchical data format like HDF5, a data storage file format in wide use across academia and industry [13]. This also provides a potential integration with MATLAB, as its .mat file formats are based on the HDF5 standard
Nested Dictionary	*Pure Python*. This is handy for any number of uses, not least of which is serialization to formats like JSON or XML. This can also be readily used by many tools. NetworkX, for example, performs graph theory analysis, and it utilizes nested dictionaries natively due to their efficiency in handling large data sets [14]

Table 4 - SysML v2 Model Text with Grammar Rules

Line	SysML v2 Textual Code	Grammar Rule Name
1	// References ----------	NOTE
2	// Intro to the SysML v2 Language-Textual Notation.pdf	NOTE
3	// https://github.com/Systems-Modeling/SysML-v2-Release/tree/master/doc	NOTE
4	import ISQ::*;	import_package
5	import ISQSpaceTime;	import_package
6	import ScalarValues::*;	import_package
7		WS
8	package TTRPGeToken{	sysml2_package, *namespace 0 begins…*
9	doc overview /*	doc_named
10	* The TTRPGeToken is a device used for displaying NPC/PC	*…continued*
11	* avatars in a physical token that can be used on a tabletop	*…continued*
12	*/	*…continued*
13	doc /* TODO: include links to remotely hosted images */	doc_unnamed
14	doc /* TODO: include links to public facing documentation */	doc_unnamed
15	comment RevComment_1 /* TO: Maatlock: Please evaluate your vigorous use of	comment_named
16	* commenting in the TTRPGeToken model. Comment variety	*…continued*
17	* in the language seems... excessive. Don't get	*…continued*
18	* carried away!	*…continued*
19	*/	*…continued*
20	package Structure{	import_package, *namespace 1 begins…*
21	doc overview /* Structural elements of model */	doc_named
22	// This is an example of composite structures, [1] pg. 16	NOTE
23	part def 'WiFi Component';	part_def
24	part def 'Bluetooth Component';	part_def
25	part def 'Integrated Wireless Chip' {	part_def, *namespace 2 begins…*
26	attribute name : String;	feature_attribute_def
27	part wifiComponent : 'WiFi Component' {	part_def, *namespace 3 begins…*
28	attribute 'WiFi Frequency' : Real;	feature_attribute_def
29	attribute 'WiFi Protocol' : String;	feature_attribute_def
30	}	*…namespace 3 ends*
31	part btComponent : 'Bluetooth Component'{	part_def, *namespace 4 begins…*
32	attribute 'BT Protocol' : String;	feature_attribute_def
33	}	*…namespace 4 ends*
34	}	*…namespace 2 ends*
35	part def 'Controller Board' {	part_def, *namespace 5 begins…*
36	part def 'wChip' specializes 'Integrated Wireless Chip';	part_def_specializes
37	attribute 'RAM_kb' : Integer;	feature_attribute_def
38	attribute 'Primary Interface' : String;	feature_attribute_def
39	attribute 'Secondary Interface' : String;	feature_attribute_def
40	attribute 'Bluetooth Capable' : Boolean;	feature_attribute_def
41	}	*…namespace 5 ends*
42	part def 'LCD Display' {}	part_def, *namespace 6 complete*
43	part def 'Battery' {	part_def, *namespace 7 begins…*
44	attribute isRechargeable : Boolean;	feature_attribute_def
45	attribute 'Battery Type' : String;	feature_attribute_def
46	attribute name : String;	feature_attribute_def
47	attribute avg_voltage_V : Real;	feature_attribute_def
48	attribute avg_capacity_mAh : Real;	feature_attribute_def
49	}	*…namespace 7 ends*
50	part def 'Raspberry Pi Pico Wireless' specializes 'Controller Board' {	part_def_specializes, *namespace 8 begins…*
51	doc info /*https://en.wikipedia.org/wiki/Raspberry_Pi*/	doc_named
52	part def wChip_PiPicoW :> wChip{	part_def_specializes, *namespace 9 begins…*
53	attribute redefines name : String = "Infineon CYW43439";	feature_attribute_redefines
54	part wifiComponet_PiPicoW :> wifiComponent{	part_def_specializes, *namespace 10 begins…*
55	attribute redefines 'WiFi Frequency': Real = 2.4;	feature_attribute_redefines
56	attribute :> 'WiFi Protocol': String = "IEEE 802.11 b/g/n";	feature_attribute_redefines
57	}	*…namespace 10 ends*
58	part btComponet_PiPicoW :> btComponent{	part_def_specializes, *namespace 11 begins…*
59	attribute redefines 'BT Protocol' : String="Bluetooth 5.2";	feature_attribute_redefines
60	}	*…namespace 11 ends*
61	}	*…namespace 9 ends*
62	attribute :> 'RAM_kb' = 264;	feature_attribute_redefines
63	attribute :> 'Bluetooth Capable' = true;	feature_attribute_redefines
64	attribute :> 'Primary Interface' = "USB 1.1";	feature_attribute_redefines
65	attribute redefines 'Secondary Interface' = "SPI";	feature_attribute_redefines
66	}	*…namespace 8 ends*
67	part def 'AA Battery Duracell Quantum' specializes 'Battery' {	part_def_specializes, *namespace 12 begins…*
68	attribute :> isRechargeable : Boolean = false;	feature_attribute_redefines
69	attribute :> 'Battery Type' : String = "AA";	feature_attribute_redefines
70	attribute :> name : String = "Duracell Quantum";	feature_attribute_redefines
71	attribute :> avg_voltage_V : Real = 1.5;	feature_attribute_redefines
72	attribute :> avg_capacity_mAh : Real = 2350.0;	feature_attribute_redefines
73	}	*…namespace 12 ends*
74	// Specialize using the "subset" symbol	NOTE
75	part def 'Bicool Round LCD IPS Display GC9A01' :> 'LCD Display' {}	part_def_specializes, *namespace 13 complete*
76		WS
77	part def 'TTRPG eToken System' {	part_def, *namespace 14 begins…*
78	part 'controller board' : 'Controller Board';	feature_part_specializes
79	part 'lcd display' : 'LCD Display';	feature_part_specializes
80	part 'battery' : 'Battery'[1..2];	feature_part_specializes
81	}	*…namespace 14 ends*
82	part def 'TTRPG eToken System Prototype' {	part_def, *namespace 15 begins…*
83	part 'controller board' : 'Raspberry Pi Pico Wireless';	feature_part_specializes
84	part 'lcd display' : 'Bicool Round LCD IPS Display GC9A01';	feature_part_specializes
85	part 'battery' : 'AA Battery Duracell Quantum';	feature_part_specializes
86	}	*…namespace 15 ends*
87	}	*…namespace 1 ends*
88	package Behavior{	sysml2_package, *namespace 16 begins…*
89	part def 'User' {}	part_def, *namespace 17 complete*
90	use case def 'Change displayed image on eToken'{	use_case_def, *namespace 18 begins…*
91	actor 'user' : 'User';	feature_actor_specializes
92	objective {	objective_def, *namespace 19 begins…*
93	doc /*	doc_unnamed
94	* The user changes the displayed image on the eToken to one	*…continued*
95	* that is currently stored in storage	*…continued*
96	*/	*…continued*
97	}	*…namespace 19 ends*
98	}	*…namespace 18 ends*
99	use case def 'Remove existing image form eToken'{	use_case_def, *namespace 20 begins…*
100	objective {	objective_def, *namespace 21 begins*
101	doc /*	doc_unnamed
102	* The user deletes an image from the eToken currently	*…continued*
103	* stored in storage	*…continued*
104	*/	*…continued*
105	}	*…namespace 21 ends*
106	actor 'user' : 'User';	feature_actor_specializes
107	}	*…namespace 20 ends*
108	use case def 'Load new image to eToken'{	use_case_def, *namespace 22 begins…*
109	objective {	objective_def, *namespace 23 begins…*
110	doc /*The user uploads a new image to the eToken's storage*/	doc_unnamed
111	}	*…namespace 23 ends*
112	actor 'user' : 'User';	feature_actor_specializes
113	}	*…namespace 22 ends*
114	use case def 'Use eToken as game piece'{	use_case_def, *namespace 24 begins…*
115	objective {	objective_def, *namespace 25 begins…*
116	doc /*The user places the eToken on the board to use as a	doc_unnamed
117	*game piece	*…continued*
118	*/	*…continued*
119	}	*…namespace 25 ends*
120	actor 'user' : 'User';	
121	}	*…namespace 24 ends*
122	}	*…namespace 16 ends*
123	}	*…namespace 0 ends*
124		
125		
126		
127		
128		
129		
130		

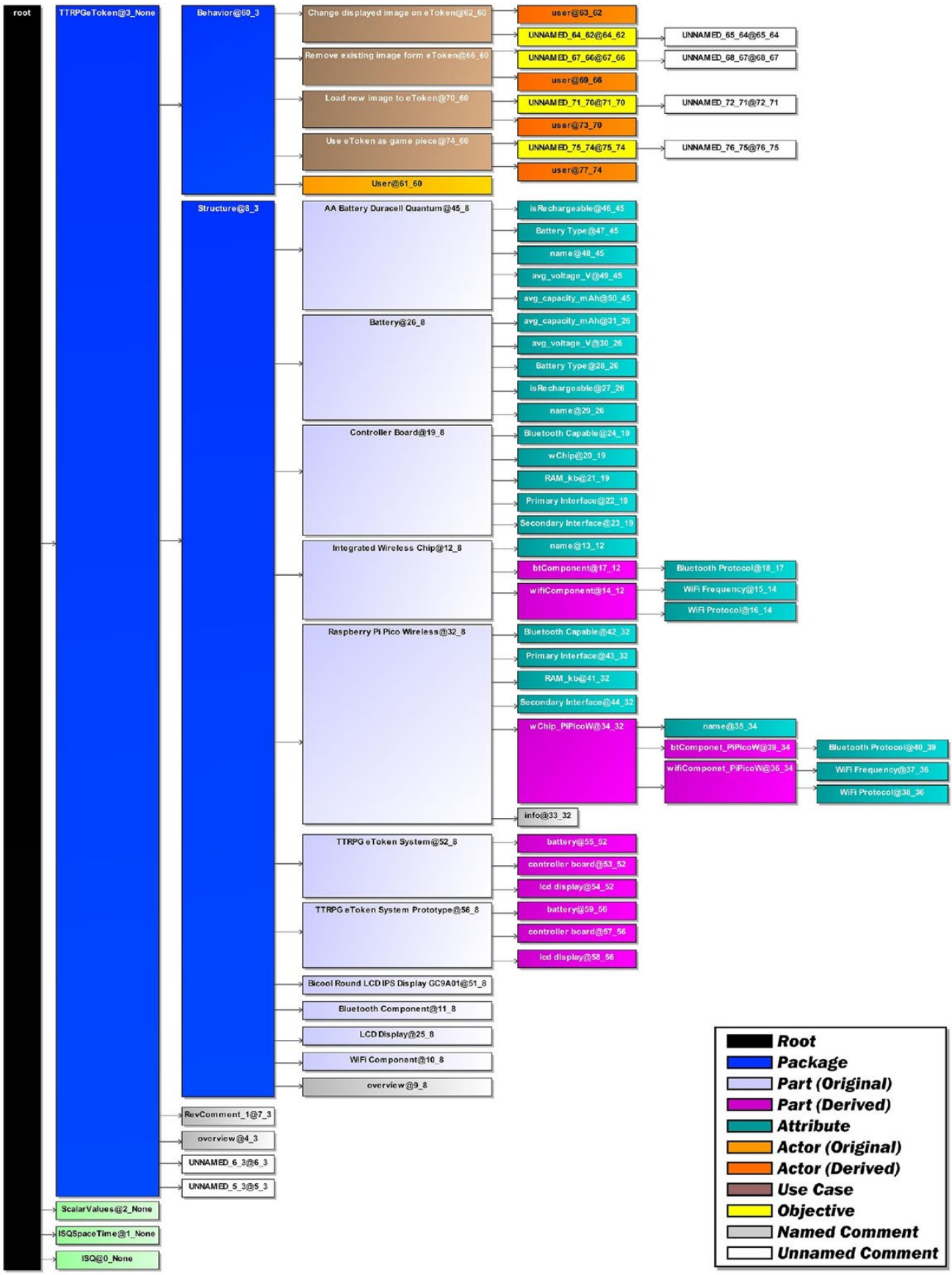


Figure 10 - Hierarchical Tree Structure of Elements for TTRPG eToken Model

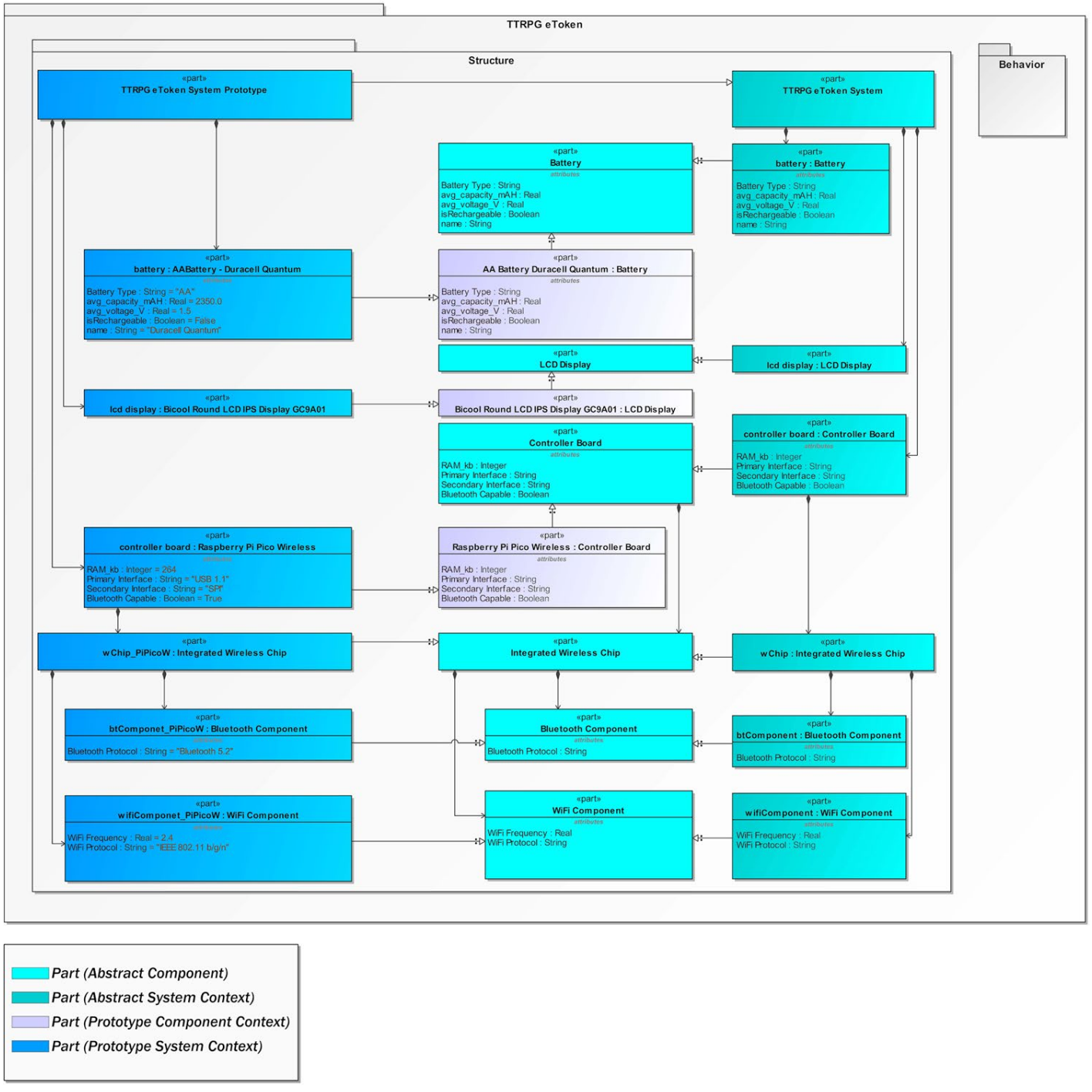


Figure 11 - TTRPG eToken SysML v2 Graphical Model

USE CASE 3: SUPPORT SYSML V2 MODEL SERIALIZATION AND PORTABILITY

With the ability to read, parse, build, and manipulate models in memory now implemented, the ability to serialize models into a portable file format will round out the foundational capabilities of PySysML2. That use case is explored further in this section.

USE CASE REFINEMENT

One of the most important goals of PySysML2 is better 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 will 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 in their own right. For example, serializing a model to a relational table would allow its direct examination in a spreadsheet for any number of purposes. Use Case 3, then, is refined to:

Build two serialization formats, one optimized for general portability between applications and one optimized to support applications built around relational tables, like databases and spreadsheets.

TRADESPACE ANALYSIS

Serialization of SysML v2 models could be accomplished somewhat by the textual language itself. The main problem with relying on the textual language for serialization, though, is that there is not a precise, one-to-one correspondence from a model back to the textual source from which it was compiled. In other words, the same model can be generated from multiple configurations of the textual source. This is because there are so many different ways to represent the same concept through the language's 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 the 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 will require 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 files (or even Excel) through the Pandas Dataframe data structure, which is already a supported PySysML2 data transformation. Lastly, it will be useful to have a simplified string representation of a model. While this is not serialization per se, it is related in that it will transform a model into a human readable string. This will be 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 for serialization, the JSON and tabular formats will be prioritized for the first version of PySysML2, along with a string format suitable for quick model viewing. A reverse compiler for the language would certainly be useful for other applications and will be explored in future work.

"PRETTY PRINT" STRING FORMAT

```
[0]: ISQ@0_None
├── [1]: ISQSpaceTime@1_None
├── [2]: ScalarValues@2_None
└── [3]: TTRPGeToken@3_None
    ├── [4]: overview@4_3
    ├── [5]: PySysML2_GENERATED_NAME_5_3@5_3
    ├── [6]: PySysML2_GENERATED_NAME_6_3@6_3
    ├── [7]: RevComment_1@7_3
    ├── [8]: Structure@8_3
    │   ├── [9]: overview@9_8
    │   ├── [10]: WiFi Component@10_8
    │   ├── [11]: Bluetooth Component@11_8
    │   ├── [12]: Integrated Wireless Chip@12_8
    │   │   ├── [13]: name@13_12
    │   │   ├── [14]: wifiComponent@14_12
    │   │   │   ├── [15]: WiFi Frequency@15_14
    │   │   │   └── [16]: WiFi Protocol@16_14
    │   │   └── [17]: btComponent@17_12
    │   │       └── [18]: Bluetooth Protocol@18_17
    │   ├── [19]: Controller Board@19_8
    │   │   ├── [20]: wChip@20_19
    │   │   ├── [21]: RAM_kb@21_19
    │   │   ├── [22]: Primary Interface@22_19
    │   │   ├── [23]: Secondary Interface@23_19
    │   │   └── [24]: Bluetooth Capable@24_19
```

Figure 12 - Excerpt of "Pretty Print" Model String

The 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. Figure 12 shows a snippet of the Pretty Print[4] string generated from the TTRPG eToken model. Note the similarities between it and the standard containment tree view of a SysML v1 model in a tool like Cameo or Rhapsody. Although it is static text rather than a navigable containment tree, it provides a clear, concise view of the model's structure. Utilizing the built-in tree walking functions of the Anytree module, the implementation for the "Pretty Print" formatted view comes almost for free, requiring a straightforward override of the default *__str__*[5] function of *Model*.

This view proved extremely useful for debugging and validation, but it can also be used for analysis purposes as well. Whenever a new update to the code was made (for example to the grammar itself or to one of the *Visitor* functions that implemented the rules of the grammar), this view of the model weas automatically generated. Mistakes resulting from erroneous model configurations were readily detected, resulting in faster and more accurate code development.

DATA SCHEMAS FOR SERIALIZATION

A "data schema" is a blueprint that describes how data is organized. It is a detailed specification of a particular data set that facilitates interoperability between producers and consumers of that data. It is defined at a 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 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 syntax 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.

[4] "Pretty Print" is a term of art that refers to a string format optimized for readability and style.

[5] All Python objects have a built-in, default private feature called *__str__*, which can be overridden with tailored code for producing formatted strings.

JSON SCHEMA


```
{
  "children": [
    {
      "archtype": "element",
      "constants": [
        null
      ],
      "context_type": "Import_packageContext",
      "element_text": null,
      "fully_qualified_name": "ISQ",
      "fully_qualified_name_tagged": "ISQ@0_None",
      "idx": 0,
      "idx_parent": null,
      "idx_related_element": null,
      "keywords": [
        "import",
        "package"
      ],
      "multiplicity": null,
      "name": "ISQ@0_None",
      "related_element_name": null,
      "sysml2_layer": "Root Syntactic Element",
      "sysml2_type": "import",
      "tree_level": 0,
      "uuid": "05382729-55a0-4c56-9a9d-088d520eeda8",
      "uuid_parent": null,
      "value_types": null
    },
```


Figure 13 - Excerpt from JSON String Serialized Model Object

The JSON schema is straightforward and automatic. It is derived from the class definitions of *Model*, *Element*, and *Behavior*, and 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. An excerpt of the TTRPG eToken model's JSON serialization is shown in Figure 13.

TABULAR CSV SCHEMA

In contrast to the JSON schema, The CSV schema was designed to support model portability between applications. PySysML2 specific fields necessary for complete JSON serialization were removed, leaving only fields common to many modeling and analysis tools. In the current version, all the omitted fields may be derived from the tabular fields, so the CSV format is suitable for roundtrip read-write for the time being. This may not continue, however, as the primary purpose of the tabular format is data interchange between applications, and the JSON format already addresses roundtrip read-write for PySysML2. The exported TTRPG eToken CSV table is shown in Table 1 on the next page. Example rows are highlighted for readability.

The tabular fields were chosen based on their criticality to the model's organization and descriptiveness of model elements. The goal is that the tabular export will provide 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 UIDs for parent and related elements, along with other information about relationships captured by the keywords used to define them. Future development of PySysML2 will include the implementation of the RESTful API. This draft tabular format can serve is the initial schema for a database view, which will facilitate this connection.

Table 5 - PySysML2 Tabular View of TTRPG eToken

sysml2_layer	archtype	sysml2_type	tree_level	name	idx	uuid	parent	idx_parent	uuid_parent	related_element_name	idx_related_element	multiplicity	value_types	constants	context_type	keywords	element_text
Root Syntactic Element	element	import	0	ISQ@0_None	0	05382729-51a0-4c56-9d6d-088d520eeda8	root							[None]	Import_packageContext	['import', 'package']	
Root Syntactic Element	element	import	0	ISQSpaceTime@1_None	1	598658f23-1f9a-4e55-9ab3-76581295bedc	root							[None]	Import_packageContext	['import', 'package']	
Root Syntactic Element	element	import	0	ScalarValues@2_None	2	a4e6677df-9a0d-41e2-a894-9006eac5e7ba	root							[None]	Import_packageContext	['import', 'package']	
Kernel Element	element	package	0	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693	root							[None]	Sysml2_packageContext	['package']	
Root Syntactic Element	element	doc	1	overview@4_3	4	a0498613-f828-476a-9444-c7662213540e	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Doc_namedContext	['doc']	The TTRPGeToken is a device used for displaying NPC/PC avatars in a physical token that can be used in a Table Top RPG game.
Root Syntactic Element	element	doc	1	PySysML2_GENERATED_NAME_5_3@5_3	5	be6c4d0f-b09a-4a24-8f96-51d5ace3c7b7	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Doc_unnamedContext	['doc']	TODO: include links to remotely hosted images
Root Syntactic Element	element	doc	1	PySysML2_GENERATED_NAME_6_3@6_3	6	713b6bdbc-e221-4639-81e4-d82784e8037c	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Doc_unnamedContext	['doc']	TODO: include links to public facing documentation
Root Syntactic Element	element	commnent	1	RevComment_1@7_3	7	e0b92afc-1275-4746-8377-0f0ba2f83e13	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Comment_namedContext	['comment']	TO: Maatlock: Please evaluate your vigorous use of commentng in the TTRPGeToken model. * Comment variety in the language seems... excessive. Don't get carried away!
Kernel Element	element	package	1	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Sysml2_packageContext	['package']	
Root Syntactic Element	element	doc	2	overview@9_8	9	63f2b27f-2c07-4026-87bf-69244e0279e9	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Doc_namedContext	['doc']	The structure package will contain all structural elements of the model
Systems Element	element	part	2	WiFi Component@10_8	10	32e67c0e-ce2e-43ef-9c59-3142d4310a02	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	part	2	Bluetooth Component@11_8	11	4b2fbd07a-e7a0-4756-b43b-6f802014b53e	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	part	2	Integrated Wireless Chip@12_8	12	c1d82227-4709-451a-9f2a-102885b7f6f2	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	attribute	3	name@13_12	13	e42e6c06-7bda-47c9-84d9-354fc0b4e6e0	Integrated Wireless Chip@12_8	12	c1d82227-4709-451a-9f2a-102885b7f6f2				String	[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	specializes	3	wifiComponent@14_12	14	72d629f7-8910-4d21-aec4-4b5b11fb861	Integrated Wireless Chip@12_8	12	c1d82227-4709-451a-9f2a-102885b7f6f2	WiFi Component@10_8	10			[None]	Part_specializesContext	['part', 'specializes']	
Systems Element	element	attribute	4	WiFi Frequency@15_14	15	4f140b19-e f43-4c64-b0cb-b4367c1c41b9	wifiComponent@14_12	14	72d629f7-8910-4d21-aec4-4b5b11fb861			Real		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	4	WiFi Protocol@16_14	16	ff927c94-9219-4b4a-b399-81e3f1f51e9d	wifiComponent@14_12	14	72d629f7-8910-4d21-aec4-4b5b11fb861			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	specializes	3	btComponent@17_12	17	07736911-b396-407d-ab04-1546e03a8aa9	Integrated Wireless Chip@12_8	12	c1d82227-4709-451a-9f2a-102885b7f6f2	Bluetooth Component@11_8	11			[None]	Feature_part_specializesContext	['part', 'specializes']	
Systems Element	element	attribute	4	Bluetooth Protocol@18_17	18	b581b7d7-7150-4085-97c3-775d415bb8be	btComponent@17_12	17	07736911-b396-407d-ab04-1546e03a8aa9			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	part	2	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbc71	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	specializes	3	wChip@20_19	20	947e1477-410f-4f4a-b2b5-a1f4af25812c	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbe71	Integrated Wireless Chip@12_8	12			[None]	Part_defContext	['def', 'part', 'specializes']	
Systems Element	element	attribute	3	RAM_kb@21_19	21	f12e6600-d477-48c9-8c78-b2b02886bab9	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbe71			Integer		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	Primary Interface@22_19	22	bb3f9231b-b322-4671-82ab-d51b7e044558	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbe71			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	Secondary Interface@23_19	23	593dc5b9-0b77-4f1c-ab2c-deb8164179d7	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbe71			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	Bluetooth Capable@24_19	24	42bef687-2e8a-4bc2-bece-8084a1bfa4ac	Controller Board@19_8	19	90da177c-279b-464f-9db6-4406941dbe71			Boolean		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	part	2	LCD Display@25_8	25	baff5f88-440f-4f61-b67d-b9dadcb6f151	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	part	2	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Systems Element	element	attribute	3	isRechargeable@27_26	27	385ab7b7-569b-4f6d-8821-07ab3f5d2e55	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189			Boolean		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	Battery Type@28_26	28	43b00534-0980-4109-99d3-9061aa69827f	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	name@29_26	29	2e56a97d-857e-4008-a255-6659b9760f41	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189			String		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	avg_voltage_V@30_26	30	8aaed7a0-b2ac-4341-8b5c-d9ccf91cf1a	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189			Real		[None]	Feature_attribute_defContext	['attribute', 'def']	
Systems Element	element	attribute	3	avg_capacity_mAh@31_26	31	69656a22-8a58-443a-b18b-30011e4c4a52	Battery@26_8	26	73d6c339-4e9c-426f-9898-622d05734189			Real		[None]	Feature_attribute_defContext	['attribute', 'def']	
Root Syntactic Element	relationship	specializes	2	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342	Controller Board@19_8	19			[None]	Part_defContext	['def', 'part', 'specializes']	
Root Syntactic Element	element	doc	3	info@33_32	33	0b103b1b-e2d0-4d23-9de4-713cdfbd7571	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4					[None]	Doc_namedContext	['doc']	https://en.wikipedia.org/wiki/Raspberry_Pi
Root Syntactic Element	relationship	specializes	3	wChip_PiPicoW@34_32	34	79d87875-1e79-4ce2-9db5-f01be45db38a	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	wChip@20_19	20			[None]	Part_def_specializesContext	['def', 'part', 'specializes']	
Root Syntactic Element	relationship	redefines	4	name@35_34	35	927e2d08-1887-4ba8-b90b-0e28adfdbab1a5	wChip_PiPicoW@34_32	34	79d87875-1e79-4ce2-9db5-f01be45db38a	name@13_12	13	String		['Infineon CYW43439']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	specializes	4	wifiComponent_PiPicoW@36_34	36	63a1bd0b-4865-41d0-9164-4de7316785c8	wChip_PiPicoW@34_32	34	79d87875-1e79-4ce2-9db5-f01be45db38a	wifiComponent@14_12	14			[None]	Part_def_specializesContext	['def', 'part', 'specializes']	
Root Syntactic Element	relationship	redefines	5	WiFi Frequency@37_36	37	dadc742-fe54-4619-9db2-6ee3e4042b0e	wifiComponet_PiPicoW@36_34	36	63a1bd0b-4865-41d0-9164-4de7316785c8	WiFi Frequency@15_14	15	Real		['2.4']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	5	WiFi Protocol@38_36	38	75bbbd52-fe6a-4765-8271-2a4908dcd086	wifiComponet_PiPicoW@36_34	36	63a1bd0b-4865-41d0-9164-4de7316785c8	WiFi Protocol@16_14	16	String		['IEEE 802.11 b/g/n wireless LAN']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	specializes	4	btComponent_PiPicoW@39_34	39	78ae75eb-e584-43ac-820d-fbf3b087223f	wChip_PiPicoW@34_32	34	79d87875-1e79-4ce2-9db5-f01be45db38a	btComponent@17_12	17			[None]	Part_def_specializesContext	['def', 'part', 'specializes']	
Root Syntactic Element	relationship	redefines	5	Bluetooth Protocol@40_39	40	67ad78e6-4503-4731-840f-6073e0ed6f3	btComponet_PiPicoW@39_34	39	78ae75eb-e584-43ac-820d-fbf3b087223f	Bluetooth Protocol@18_17	18	String		['Bluetooth 5.2']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	4	RAM_kb@41_32	41	fe9b981e-54e0-4cfb-8618-1c22c6d7da5e	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	RAM_kb@21_19	21			['264']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	4	Bluetooth Capable@42_32	42	3b4351c0-c89f-4c41-917b-c869384eb538	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	Bluetooth Capable@24_19	24			['true']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	4	Primary Interface@43_32	43	cdbeb3bc-305b-44d4-8574-d04caab3f666	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	Primary Interface@22_19	22			['USB 1.1']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	3	Secondary Interface@44_32	44	b84b5b76-33e4-4ec7-945b-648907218868	Raspberry Pi Pico Wireless@32_8	32	4ac6fd02-7bb9-421f-ae14-3026bb376ff4	Secondary Interface@23_19	23			['SPI']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	specializes	2	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342	Battery@26_8	26			[None]	Part_def_specializesContext	['def', 'part', 'specializes']	
Root Syntactic Element	relationship	specializes	3	isRechargeable@46_45	46	353fcfa1-1356-4144-a3d2-5b7d84bdcf173	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	isRechargeable@27_26	27	Boolean		['false']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	3	Battery Type@47_45	47	adb238a0-e538-47e4-aed1-2466c761500c	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	Battery Type@28_26	28	String		['AA']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	3	name@48_45	48	7fae6bbe-d940-41ad-9eba-2507aabbdcdd	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	name@13_12	13	String		['Duracel Quantum']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	3	avg_voltage_V@49_45	49	98397776-b061-423b-b959-d6f12bb49394	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	avg_voltage_V@30_26	30	Real		['1.5']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Root Syntactic Element	relationship	redefines	3	avg_capacity_mAh@50_45	50	1497a63d-6bec-4049-b746-da0d2dfa465a	AA Battery Duracell Quantum@45_8	45	0c81ca49-0db3-4631-83c8-cdb3f4355e40	avg_capacity_mAh@31_26	31	Real		['2350.0']	Feature_attribute_redefinesContext	['attribute', 'redefines']	
Systems Element	element	specializes	3	Bicoal Round LCD IPS Display GC9A01@51_8	51	93a19ebe-bace-49b8-8a2a-185ee0228d3b	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342	LCD Display@25_8	25			[None]	Part_def_specializesContext	['def', 'part', 'specializes']	
Systems Element	element	part	2	TTRPG eToken System@52_8	52	f5e6cc46-09cc-4d13-aadb-f5a01fc18cf5	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Root Syntactic Element	relationship	specializes	3	controller board@53_52	53	7a8c1855-09ff-4e85-b475-f99c0af40d4d	TTRPG eToken System@52_8	52	f5e6cc46-09cc-4d13-aadb-f5a01fc18cf5	Controller Board@19_8	19			[None]	Feature_part_specializesContext	['part', 'specializes']	
Root Syntactic Element	relationship	specializes	3	lcd display@54_52	54	107753c9-87fe-4970-9973-4266da693813	TTRPG eToken System@52_8	52	f5e6cc46-09cc-4d13-aadb-f5a01fc18cf5	LCD Display@25_8	25			[None]	Feature_part_specializesContext	['part', 'specializes']	
Root Syntactic Element	relationship	specializes	3	battery@55_52	55	0f8947a1-2243-4967-a242-d299bf6db52d	TTRPG eToken System@52_8	52	f5e6cc46-09cc-4d13-aadb-f5a01fc18cf5	Battery@26_8	26 [1..2]			[None]	Feature_part_specializesContext	['part', 'specializes']	
Systems Element	element	part	2	TTRPG eToken System Prototype@56_8	56	c3e61748-6676-417d-ab85-9892b50b7b2e	Structure@8_3	8	04f589a7-5dfb-4bb2-8b6b-fd743e92b342					[None]	Part_defContext	['def', 'part']	
Root Syntactic Element	relationship	specializes	3	controller board@57_56	57	7b14eef0-2c1c-472c-a6a4-68c5f4821ccf	TTRPG eToken System Prototype@56_8	56	c3e61748-6676-417d-ab85-9892b50b7b2e	Raspberry Pi Pico Wireless@32_8	32			[None]	Feature_part_specializesContext	['part', 'specializes']	
Root Syntactic Element	relationship	specializes	3	lcd display@58_56	58	9df7d80d6-a6fe-4ac7-817a-47726f306dd0	TTRPG eToken System Prototype@56_8	56	c3e61748-6676-417d-ab85-9892b50b7b2e	Bicoal Round LCD IPS Display GC9A01@51_8	51			[None]	Feature_part_specializesContext	['part', 'specializes']	
Root Syntactic Element	relationship	specializes	3	battery@59_56	59	c3bb85b4-9d83-4d78-abdf-3ce4cd238af5	TTRPG eToken System Prototype@56_8	56	c3e61748-6676-417d-ab85-9892b50b7b2e	AA Battery Duracell Quantum@45_8	45			[None]	Feature_part_specializesContext	['part', 'specializes']	
Kernel Element	element	package	1	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447	TTRPGeToken@3_None	3	108095bf-66fb-4831-bb6a-0cdee5719693					[None]	Sysml2_packageContext	['package']	
Root Syntactic Element	element	part	2	User@61_60	61	6b7ce10d-5ee1-465c-9d7f-70d9f4e4fa0	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447					[None]	Part_defContext	['def', 'part']	
Root Syntactic Element	element	usecase	2	Change displayed image on eToken@62_60	62	b8747ea9-15fa-4f72-81cc-816344450a8d	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447					[None]	Use_case_defContext	['case', 'def', 'use']	
Root Syntactic Element	relationship	specializes	3	user@63_62	63	32e17502-6d9c-4490-923d-fd8d282b3e38	Change displayed image on eToken@62_60	62	b8747ea9-15fa-4f72-81cc-816344450a8d	User@61_60	61			[None]	Feature_actor_specializesContext	['actor', 'specializes']	
Systems Element	element	objective	3	PySysML2_GENERATED_NAME_64_62@64_62	64	a70a5040-9324-4656-b02a-17c41a6ebca0	Change displayed image on eToken@62_60	62	b8747ea9-15fa-4f72-81cc-816344450a8d					[None]	Objective_defContext	['def', 'objective']	
Root Syntactic Element	element	doc	4	PySysML2_GENERATED_NAME_65_64@65_64	65	2a883e87-3703-4a6d-b889-320075f4f9a3	PySysML2_GENERATED_NAME_64_62@64_62	64	a70a5040-9324-4656-b02a-17c41a6ebca0					[None]	Doc_unnamedContext	['doc']	The user changes the displayed image on the eToken to one that is currently stored in storage
Root Syntactic Element	element	usecase	2	Remove existing image form eToken@66_60	66	528ecc1e-c73b-42f0-a2b3-122e59b5b3af	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447					[None]	Use_case_defContext	['case', 'def', 'use']	
Systems Element	element	objective	3	PySysML2_GENERATED_NAME_67_66@67_66	67	50e28e5c-8f4e-4ff8-a3fb-9abda871be86	Remove existing image form eToken@66_60	66	528ecc1e-c73b-42f0-a2b3-122e59b5b3af					[None]	Objective_defContext	['def', 'objective']	
Root Syntactic Element	element	doc	4	PySysML2_GENERATED_NAME_68_67@68_67	68	54d0f518-4ba0-49d9-b24b-865fa394ddc2	PySysML2_GENERATED_NAME_67_66@67_66	67	50e28e5c-8f4e-4ff8-a3fb-9abda871be86					[None]	Doc_unnamedContext	['doc']	The user deletes an image from the eToken currently stored in storage
Root Syntactic Element	relationship	specializes	3	user@69_66	69	7cb09d84-f183-43a6-b211-1e971681f0aa	Remove existing image form eToken@66_60	66	528ecc1e-c73b-42f0-a2b3-122e59b5b3af	User@61_60	61			[None]	Feature_actor_specializesContext	['actor', 'specializes']	
Systems Element	element	usecase	2	Load new image to eToken@70_60	70	b502303b-0969-46a7-b5b4-91014a0f385	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447					[None]	Use_case_defContext	['case', 'def', 'use']	
Systems Element	element	objective	3	PySysML2_GENERATED_NAME_71_70@71_70	71	76e65279-9695-44c7-b594-910147a0f385	Load new image to eToken@70_60	70	b502303b-0969-46a7-baad-23993fbb2649					[None]	Objective_defContext	['def', 'objective']	
Systems Element	element	doc	4	PySysML2_GENERATED_NAME_72_71@72_71	72	3884f200-14b2-491d-97ae-b1a2ed51ee26	PySysML2_GENERATED_NAME_71_70@71_70	71	76e65279-9695-44c7-b594-910147a0f385					[None]	Doc_unnamedContext	['doc']	The user uploads a new image to the eToken's storage
Root Syntactic Element	relationship	specializes	3	user@73_70	73	08ee37a4-f1a5-4b30-8997-f2017521268b	Load new image to eToken@70_60	70	b502303b-0969-46a7-baad-23993fbb2649	User@61_60	61			[None]	Feature_actor_specializesContext	['actor', 'specializes']	
Systems Element	element	usecase	2	Use eToken as game piece@74_60	74	461cacca-ecd5-4389-b11d-9109ff4172bd	Behavior@60_3	60	2c8a12b1-b776-4e2b-8503-687237a27447					[None]	Use_case_defContext	['case', 'def', 'use']	
Systems Element	element	objective	3	PySysML2_GENERATED_NAME_75_74@75_74	75	e8a5cb90-59c4-4145-be40-8184a25471f4	Use eToken as game piece@74_60	74	461cacca-ecd5-4389-b11d-9109ff4172bd					[None]	Objective_defContext	['def', 'objective']	
Root Syntactic Element	element	doc	4	PySysML2_GENERATED_NAME_76_75@75_75	76	85a837ea-237e-4db0-903f-d1e94d1e1d37	PySysML2_GENERATED_NAME_75_74@75_74	75	e8a5cb90-59c4-4145-be40-8184a25471f4					[None]	Doc_unnamedContext	['doc']	The user places the eToken on the board to use as a game piece
Root Syntactic Element	relationship	specializes	3	user@77_74	77	a28ba3ec-01cd-469d-a46a-ebc389753b1b	Use eToken as game piece@74_60	74	461cacca-ecd5-4389-b11d-9109ff4172bd	User@61_60	61			[None]	Feature_actor_specializesContext	['actor', 'specializes']	

Index 3

sysml2_layer	Kernel Element
archtype	element
sysml2_type	package
tree_level	0
name	TTRPGeToken@3_None
idx	3
uuid	108095bf-66fb-4831-bb6a-0cdee5719693
parent	root
idx_parent	
uuid_parent	
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Sysml2_packageContext
keywords	['package']
element_text	

Index 5

sysml2_layer	Root Syntactic Element
archtype	element
sysml2_type	doc
tree_level	1
name	PySysML2_GENERATED_NAME_5_3@5_3
idx	5
uuid	be6c4d0f-b09a-4a24-8f96-51d5ace3c7b7
parent	TTRPGeToken@3_None
idx_parent	3
uuid_parent	108095bf-66fb-4831-bb6a-0cdee5719693
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Doc_unnamedContext
keywords	['doc']
element_text	TODO: include links to remotely hosted images

Index 10

sysml2_layer	Systems Element
archtype	element
sysml2_type	part
tree_level	2
name	WiFi Component@10_8
idx	10
uuid	32e67c0e-ce2e-43ef-9c59-3142d4310a02
parent	Structure@8_3
idx_parent	8
uuid_parent	04f589a7-5dfb-4bb2-8b6b-fd743e92b342
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Part_defContext
keywords	['def', 'part']
element_text	

Index 38

sysml2_layer	Root Syntactic Element
archtype	relationship
sysml2_type	redefines
tree_level	5
name	WiFi Protocol@38_36
idx	38
uuid	75bbbd52-fe6a-4765-8271-2a4908dcd986
parent	wifiComponet_PiPicoW@36_34
idx_parent	36
uuid_parent	63a1bd0b-4865-41d0-9164-4de7316785c8
related_element_name	WiFi Protocol@16_14
idx_related_element	16
multiplicity	
value_types	String
constants	['IEEE 802.11 b/g/n wireless LAN']
context_type	Feature_attribute_redefinesContext
keywords	['attribute', 'redefines']
element_text	

Index 55

sysml2_layer	Root Syntactic Element
archtype	relationship
sysml2_type	specializes
tree_level	3
name	battery@55_52
idx	55
uuid	0f8947a1-2243-4967-a242-d299bf6db52d
parent	TTRPG eToken System@52_8
idx_parent	52
uuid_parent	f5e6cc46-09cc-4d13-aadb-f5a01fc18cf5
related_element_name	Battery@26_8
idx_related_element	26
multiplicity	[1..2]
value_types	
constants	[None]
context_type	Feature_part_specializesContext
keywords	['part', 'specializes']
element_text	

Index 74

sysml2_layer	Systems Element
archtype	element
sysml2_type	usecase
tree_level	2
name	Use eToken as game piece@74_60
idx	74
uuid	461cacca-ecd5-4389-b11d-9109ff4172bd
parent	Behavior@60_3
idx_parent	60
uuid_parent	2c8a12b1-b776-4e2b-8503-687237a27447
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Use_case_defContext
keywords	['case', 'def', 'use']
element_text	

Index 75

sysml2_layer	Systems Element
archtype	element
sysml2_type	objective
tree_level	3
name	PySysML2_GENERATED_NAME_75_74@75_74
idx	75
uuid	e8a5cb90-59c4-4145-be40-8184a25471f4
parent	Use eToken as game piece@74_60
idx_parent	74
uuid_parent	461cacca-ecd5-4389-b11d-9109ff4172bd
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Objective_defContext
keywords	['def', 'objective']
element_text	

Index 76

sysml2_layer	Root Syntactic Element
archtype	element
sysml2_type	doc
tree_level	4
name	PySysML2_GENERATED_NAME_76_75@76_75
idx	76
uuid	85a837ea-237e-4db0-903f-d1e94d1e1d37
parent	PySysML2_GENERATED_NAME_75_74@75_74
idx_parent	75
uuid_parent	e8a5cb90-59c4-4145-be40-8184a25471f4
related_element_name	
idx_related_element	
multiplicity	
value_types	
constants	[None]
context_type	Doc_unnamedContext
keywords	['doc']
element_text	The user places the eToken on the board to use as a game piece

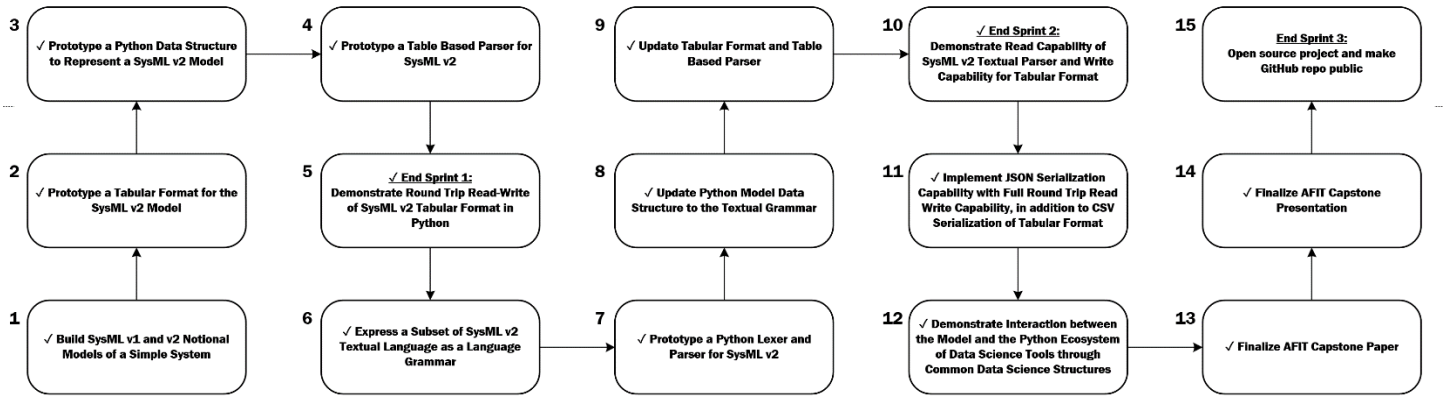


Figure 14 - Development Increment 1

The initial version of PySysML2 was developed over a series of three-month long sprints, shown above in Figure 14. The remaining task is to finalize the open-source release of PySysML2, which will completed upon submission and acceptance of this report.

After PySysML2 is released under the Apache Commons license, the first step will be to 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 will be vital going forward. In addition to the open-source community, the defense sector is also very active in this space. Many organizations, ranging from large contractors to Small Business Innovation Research (SBIR) firms, are pursuing solutions to the interoperability problems outlined at the beginning of this paper.

Regarding next features of the application, upcoming work will seek to continue building out the grammar definition, gradually filling out the rest of the SysML v2 specification.

Additionally, the tabular CSV schema will be refactored for compatibility with the open HUDS format, currently under development by the Aerospace Corporation, in addition to a corresponding XML serialization schema.

Further on the horizon, new capabilities will include the implementation of the RESTful API and integration with the SysML v2 model repository, in addition to the development of a reverse compiling feature capable of generating SysML v2 textual code.

The development of a basic simulation and analysis engine in PySysML2 would also be straightforward. While the current data transformation capability supports analysis in other tools, a suite of native analysis utilities would prove very useful, especially considering that the SST has not done much work in that space.

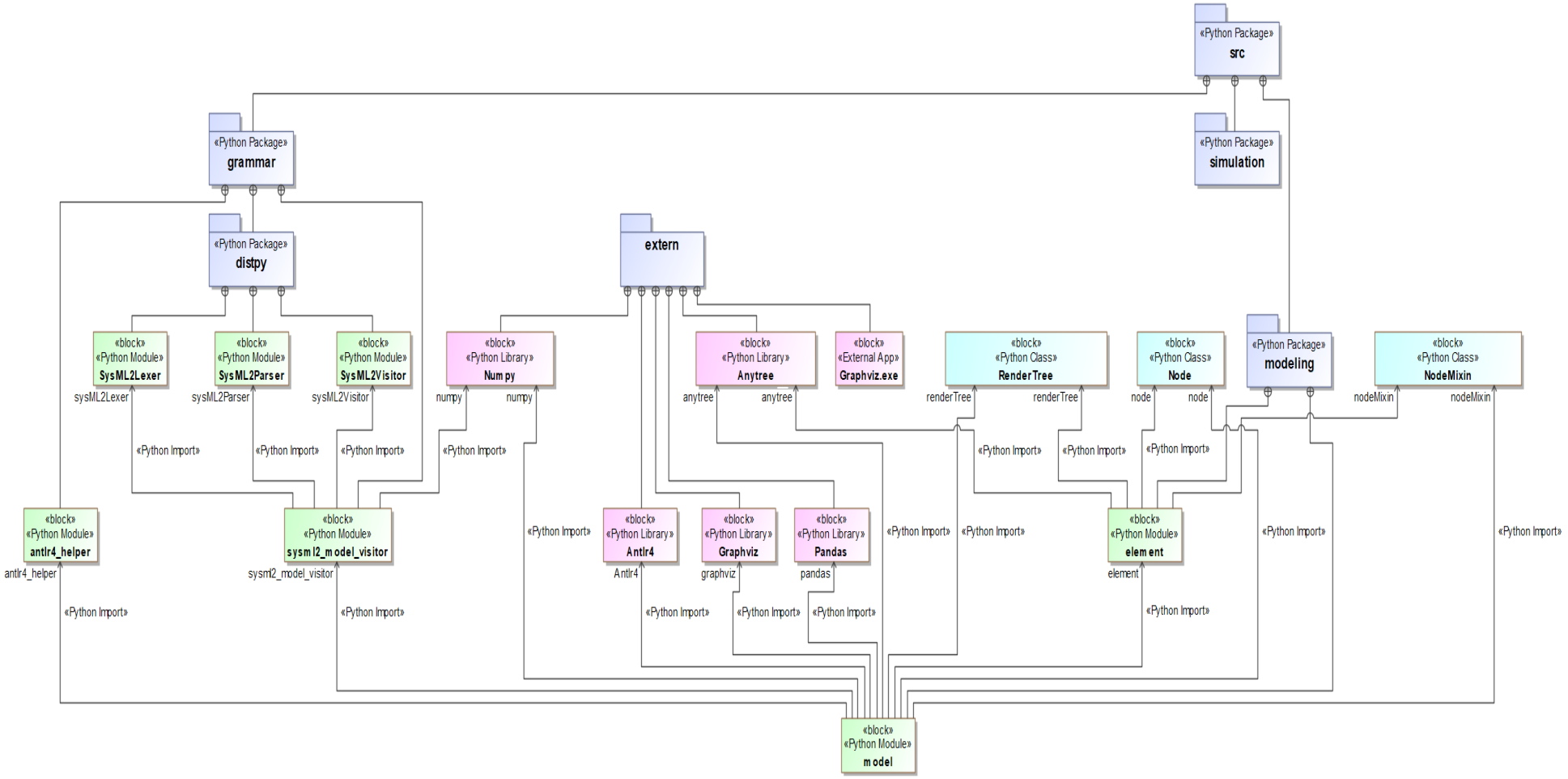
During the project some limited experimentation was performed using the publicly available BestBuy pricing API to programmatically retrieve cost information for components of the TTRPG eToken (e.g. the Raspberry Pi Pico Wireless and the round LCD screen). While this exploration was paused due to time constraints, a tradespace analysis capability utilizing automated API access could prove very useful.

CONCLUSION

The goal of this project was 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, will support eventual inclusion of the rest of the specification. Additionally, as new features of the SysML v2 textual language and its parent language KerML are released, the workflow will readily support both the adaption of existing rules to future changes and the inclusion of new rules.

More broadly, though, this pathfinder project demonstrates the possibilities that open-source development and open data standards can realize for MBSE usage in the DoD. Completing a project of this scope in three months was made possible only through the use of open-source technology, in addition to leveraging the work completed and made public by the SST, the consortium of developers supporting the SysML v2 Release. The decision of the OMG to prioritize model interoperability across tools in the SysML v2 RFP is truly a paradigm shift for MBSE standards and tool development. The proposed textual modeling language specification and accompanying RESTful API to support ease of model access across tools is evidence of that.

Ultimately, though, if the PySysML2 project accomplishes anything, the author's hope would be that it inspires future work in related areas. This capability was built from the ground up in three months, using computer programming and data science skills that most new engineering and computer science graduates will come standard with as they enter the work force. A dedicated team with access to higher tier expertise could accomplish so much more. As digital transformation in the DoD 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 a 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 silos locking away data from analysis. In that future, MBSE models become what they were intended to be in the first place: repositories of organized, integrated systems engineering data, easily accessible by the workforce across both engineering and functional support fields, and compatible with the vast array of powerful data science, simulation, and analysis tools available to the modern workforce.



Appendix A Figure 1 – PySysML2 Module Dependency Diagram

Appendix A Table 1

ANTLR4	The standalone ANTLR4 .command line application is used to generate the Python Lexer, Parser, and Visitor classes from the grammar. The ANTLR4 Python module is also imported, as that has supporting functions required by the generated language parsing code
Anytree	This is the tree data structure implementation underpinning the Model, Element, and Behavior classes. Although it could be replaced by a custom tree class, Anytree is widely used by the community, and its use reduces bugs that may be introduced from a custom tree implementation
Graphviz	Both the standalone command line application and the Python module are used. Graphviz routines are used for conversion of the model to .dot format, which is useful for graph theory analysis, and also for generating quick graphs of models using built in, automatic route optimization
Numpy	Numpy is the workhorse numerical analysis package for Python and includes the Numpy Multi-dimensional Array, a target data structure for model interoperability. It is also a dependency for Pandas.
Pandas	Pandas implements the Dataframe data structure, a target data structure for model interoperability, and it is also used for CSV serialization and writing to Excel

REFERENCES

[1] Office of the Deputy Assistant Secretary of Defense for Systems Engineering, "DoD Digital Engineering Strategy," Department of Defense, 2018.

[2] Object Management Group, "What is SysML?," 2022. [Online]. Available: https://www.omgsysml.org/what-is-sysml.htm.

[3] Object Management Group, "Systems Modeling Language (SysML) v2 Request for Proposal (RFP)," 2018.

[4] SysML v2 Submission Team (SST), "SysML v2 Release," Aug 2022. [Online]. Available: https://github.com/Systems-Modeling/SysML-v2-Release.

[5] SysML v2 Submission Team (SST), "SysML V2 Specification," 2022.

[6] SysML v2 Submission Team (SST), "Systems Modeling Application Program Interface (API) and Services," 2022.

[7] Apache Software Foundation, "Apache License, v 2.0," [Online]. Available: https://www.apache.org/licenses/LICENSE-2.0.

[8] GNU, Free Software Foundation, [Online]. Available: https://www.gnu.org/licenses/gpl-3.0.en.html.

[9] Aerospace Corporation, "HUDS Cameo Plugin GitHub Repo," [Online]. Available: https://github.com/the-aerospace-corporation/mtip-cameo-plugin.

[10] T. Parr, The Definitive ANTLR4 Reference, The Pragmatic Programmers, 2013.

[11] Model Driven Solutions, "Introduction to the SysML v2 Language Textual Notation," 2022.

[12] Department of Defense, "DoDAF Architecture Framework," [Online]. Available: https://dodcio.defense.gov/Library/DoD-Architecture-Framework/dodaf20_background/.