NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

REUSABLE MONTEREY PHOENIX CODE LIBRARIES FOR BEHAVIOR MODELS AND MODEL SEGMENTS

by

Rusita H. Desai

September 2021

Thesis Advisor: Kristin M. Giammarco
Second Reader: Walter E. Owen

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2021	**3. REPORT TYPE AND DATES COVERED** Master's thesis

4. TITLE AND SUBTITLE
REUSABLE MONTEREY PHOENIX CODE LIBRARIES FOR BEHAVIOR MODELS AND MODEL SEGMENTS

5. FUNDING NUMBERS

6. AUTHOR(S) Rusita H. Desai

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release. Distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

System developers generate and simulate models for behavior analysis of the system at different phases of the system life cycle to aid stakeholders in understanding and informing decisions for systems or programs. Monterey Phoenix (MP), a Navy-developed modeling and simulation tool, is distinct in that it provides a unique modeling environment for exhaustively testing and finding desired and undesired behaviors. To create a model in MP, users manually browse the examples with no indexing or search capability to find matching code for their needs or construct the needed code from memory or another source. The MP tools provided no built-in mechanism to store and reuse snippets of MP code that have already been constructed. This thesis enables reuse of MP code segments by creating a formalized library that can be automatically accessed for repurpose. This research builds and catalogs code templates based on snippets from examples of prior MP models and frequently used expressions. It then develops, tests, and verifies a process for users to access model and snippet libraries through MP. The research conducted in this thesis enables the Department of Defense (DOD), Navy, and systems engineering community to repurpose validated code segments from one system to another. Having a repository with validated code templates significantly reduces the time to build a model by providing more efficient and convenient ways to build models.

14. SUBJECT TERMS model-based systems engineering, Monterey Phoenix, repository, reuse, library	**15. NUMBER OF PAGES** 87
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

REUSABLE MONTEREY PHOENIX CODE LIBRARIES FOR BEHAVIOR MODELS AND MODEL SEGMENTS

Rusita H. Desai
Civilian, Department of the Navy
BS, California State Polytechnic University, Pomona, 2011

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 2021

Approved by: Kristin M. Giammarco
Advisor

Walter E. Owen
Second Reader

Oleg A. Yakimenko
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

System developers generate and simulate models for behavior analysis of the system at different phases of the system life cycle to aid stakeholders in understanding and informing decisions for systems or programs. Monterey Phoenix (MP), a Navy-developed modeling and simulation tool, is distinct in that it provides a unique modeling environment for exhaustively testing and finding desired and undesired behaviors. To create a model in MP, users manually browse the examples with no indexing or search capability to find matching code for their needs or construct the needed code from memory or another source. The MP tools provided no built-in mechanism to store and reuse snippets of MP code that have already been constructed. This thesis enables reuse of MP code segments by creating a formalized library that can be automatically accessed for repurpose. This research builds and catalogs code templates based on snippets from examples of prior MP models and frequently used expressions. It then develops, tests, and verifies a process for users to access model and snippet libraries through MP. The research conducted in this thesis enables the Department of Defense (DOD), Navy, and systems engineering community to repurpose validated code segments from one system to another. Having a repository with validated code templates significantly reduces the time to build a model by providing more efficient and convenient ways to build models.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

DOD	Department of Defense
FEM	Finite Element Method
GUI	graphical user interface
MBSE	model-based systems engineering
MP	Monterey Phoenix
PCB	Printed Circuit Board
SE	systems engineering

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

System developers generate and simulate models for behavior analysis of the system at different phases of the system life cycle to aid stakeholders in understanding and informing decisions for systems or programs. Monterey Phoenix (MP), a Navy-developed modeling and simulation tool, is distinct in that it provides a unique modeling environment for exhaustively testing and finding desired and undesired behaviors. To create a model in MP, users must manually browse the examples with no indexing or search capability to find matching code for their needs or construct the needed code from memory or another source. Model developers need a mechanism to store and reuse snippets of MP code that have already been constructed. The objective of this thesis is to create a process to reuse MP code segments by creating formalized libraries that can be accessed through MP graphical user interface (GUI) for repurpose. To accomplish the objective of this thesis, a design-oriented systems engineering method was utilized, which was divided in three phases.

The first phase consisted of analyzing modeling tools from other engineering fields such as Cadence PSpice, used in electrical engineering, and Dassault Systems SolidWorks, used in mechanical engineering. The analysis of these tools specifically looked at model library management, organization, and access capabilities and how they support model reuse. This was followed by a comparison of both PSpice and SolidWorks with MP to identify gaps and determine the specific need to implement this capability within MP.

The second phase encompassed an MP-Gryphon architecture review to understand current MP-Gryphon components and how they are utilized within the system. Next, an analysis of user's utilization of the current MP-Gryphon GUI was done to identify where new additions to the GUI could be made for the library features. This led to the generation of use cases that illustrated the interaction between the user and the MP-Gryphon GUI with the new library features. Finally, the second phase culminates with a requirements generation for the new design to support model and snippet library features within MP-Gryphon.

Phase three of the research implements the library features within MP-Gryphon and tests the design for validity. First, in the implementation stage, the three library types are

characterized. Preloaded Examples Library contain complete example models provided by default when MP-Gryphon is installed, Preloaded Snippets Library contain excerpts of MP code that can be inserted into a larger MP model, and Personal Collection Library is a customizable library that user can utilize to create user-defined model or snippet libraries depending on user-specific projects or analysis. Next, changes to the MP-Gryphon GUI were completed to allow users to access the model and snippet libraries. Specifically, the GUI now allows users to import and export to and from libraries and provides the ability to search within libraries. Last, in the third phase, test cases were developed based on the requirements, followed by verification and validation of the new design via testing.

This research successfully demonstrates a process to access the model or snippet libraries for reuse. New features and capabilities added to the MP-Gryphon GUI to support the objectives of this thesis include libraries with distinct characteristics to store MP model and snippets, provides the ability to the user to expand model libraries through the availability of personal collection libraries, search functionality to find models within libraries and by further refining the search criteria by filtering by filename or texts within the file, import/ export capability of models and snippets, ability to reviewing and selecting the model or snippets contents prior to importing. The updated MP-Gryphon GUI provides the users of MP with the ability to store, organize, and manage model and snippet libraries and can enable reuse of entire models or snippets. The library feature implementation completed in this thesis is a starting point in this area and provides further opportunities to evolve and optimize this capability.

Four areas were found where further research can be conducted to expand on the model libraries and reuse the addition of a Behavior Patterns library, further optimization of the model search capability, sharing repositories with other MP users and maintenance of libraries. The research conducted in this thesis enables Department of Defense (DOD), Navy, and Systems Engineering community to repurpose validated code segments from one system to another. Having a repository with validated code templates significantly reduces the time to build a model by providing more efficient and convenient ways to build models.

ACKNOWLEDGMENTS

With deepest gratitude, I would like to thank my advisor Dr. Kristin Giammarco, who provided me with her invaluable expertise and knowledge for this thesis. Without her support and guidance, this thesis would not have been possible. A sincere thank you to Bruce Allen for his contribution to the Monterey Phoenix Gryphon Graphical User Interface updates, which greatly supported the outcome of this research and aided in demonstration of this topic. Thank you to Mike Collins for providing his expertise to a better understanding of Monterey Phoenix architecture, which brought a deeper insight into the architecture and informed this research greatly.

Thank you to my family, my friends, the faculty at Naval Postgraduate School, my fellow cohort students, my supervisors, and my coworkers. You have directly contributed to me completing this program by providing me with a support system, guidance, help, and friendship.

Lastly, I would like to thank my dearest husband and best friend, Spencer Brown, who has endured my absence physically and mentally for the past two years and has encouraged me through this journey. Thank you for being my rock and constant source of laughter.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

This chapter establishes the motivation and need for the research topic and provides background information on Monterey Phoenix (MP). Furthermore, it provides details on a research methodology and approach utilized to answer the research question. Finally, this chapter provides benefits of this research.

A. BACKGROUND AND MOTIVATION

System developers perform modeling and simulation of systems to better understand the overall system and its behavior before physically building it. This crucial exercise helps developers to better understand the system by assessing the behaviors and interactions to discover new system behaviors using executable models before investing in expensive testing. Developers can generate and simulate models for behavior analysis of the system at different phases of the system life cycle to aid stakeholders in understanding and informing decisions for systems or programs. The Department of Defense (DOD) recognizes the benefits of model-based systems engineering (MBSE) practices, and their recognition of these benefits motivates them to implement them into systems and the acquisition life cycle. Thus, in 2018, the DOD released its *Digital Engineering Strategy,* which calls to "formalize the development, integration, and use of models to inform enterprise and program decision making" (Office of the Deputy Assistant Secretary of Defense for Systems Engineering 2018, 4).

In MBSE, systems are represented by models, which are abstractions and not a real physical system. As such, one model can be reused to describe another similar system due to no interference of physical limitations. An existing established model from one project can be utilized in another project within new context and can be adapted to be used appropriately in the new project. The concept of model reuse is an important capability that must be incorporated within MBSE tools to drive forward digital engineering within the DOD. While there are variety of MBSE tools in the market with different approaches to MBSE there is only one that does scope-complete scenario generation, known as Monterey

Phoenix (MP), which is the focus of this thesis because of its presently untapped opportunity for model reuse.

MP "is a framework for software system architecture and business process (workflow) specification based on behavior models" (Auguston 2020, 5). MP provides modeling and simulation capability of variety of systems which aid to answer questions about their behavior, providing a unique modeling environment for exhaustively testing and finding desired and undesired behaviors up to a user-defined scope limit on iterations. To create a model in MP, a designer must model the system's behaviors and its interactions. Constructing known behaviors and interactions within the model is an important task for a designer as these specifications, when simulated, provide valuable insights of the system behavior. Next, the designer simulates the model, validates the results, and documents alternative and/or emergent behaviors.

B. RESEARCH STATEMENT

To assist users learning MP, many example models come preloaded with the MP tools, and users often copy and paste code snippets from these, modifying them as needed for their own model. Currently, users must manually browse the examples with no indexing or search capability to find matching code for their needs or construct the needed code from memory or another source. The frequent manual reconstruction of common code snippets by MP users has led to the idea for a reusable code template library to minimize redundant steps while modeling. Model developers need a mechanism to share and reuse snippets of MP code that have already been constructed to efficiently conduct common tasks like queries, annotations, tables, and graphs. This thesis creates and automates a process for the reuse of MP code segments by creating a formalized repository that can be automatically accessed for repurpose.

C. RESEARCH METHODOLOGY

This thesis utilizes a design-oriented research method. This thesis research includes three main phases. First phase conducts a literature review of past research or other related research materials that will aid in supporting this thesis, and it studies the current architecture of Monterey Phoenix tools and the systems with which they interface. Second

phase builds a catalog of code templates based on snippets from examples of prior MP models. Third phase develops the process for the user to access the template library through a MP tool, along with testing, verification, and analysis of the results.

The literature review phase of the research consists of a review of related work and analysis of previous research projects conducted in the area of code template cataloging and reuse. Additionally, the literature review covers modeling and simulation tools from other disciplines such as PSpice and SolidWorks in order to find concepts of applications potentially applicable to behavior models involving a broader range of systems. This phase also examines previous MP models to search for schema, expressions, statements, patterns, and other code snippets to employ in the library. Additionally, this phase reviews MP software tool architecture to understand how it currently functions to implement appropriate changes to the system. Lastly, this phase reviews the current user interface in preparation for creating a framework to automatically access the code template library.

The second phase, the repository creation phase, entails organizing MP code snippets found in the literature review as templates for cataloguing them in an appropriate location within a repository. This phase also establishes appropriate terms and naming conventions to aid in the search capability.

After the literature review and repository creation phases are completed, this last phase develops the process of users accessing the repository and implements that process as new user interface features. First, this phase creates an MP model to describe the process of a user accessing the repository through an MP tool. The simulated output from the MP model generates event traces, which can be considered use cases to guide the design of the new capability. After use case analysis of the user interactions with the library, design requirements will be generated to update MP-Gryphon Graphical User Interface so the user can access the libraries. New design implementation to MP-Gryphon requires assistance from a Naval Postgraduate School programmer with access to MP-Gryphon and Python programming language to make the required design changes. Lastly, this phase tests and verifies this implementation and documents the results. The use cases generated from MP model will be the specifications used as test plans to ensure all paths through the use cases are present in the implementation and no unexpected differences are found.

D. BENEFITS OF THIS RESEARCH

Model-based systems engineering (MBSE) is a rapidly growing area of interest within the Navy and DOD. There is a great deal of interest in utilizing and implementing MBSE processes and tools in all aspects of the system acquisition life cycle, to gain a greater understanding of system behaviors, identify unknown risks, and to find new requirements for systems. The research conducted in this thesis will enable the DOD, Navy, and systems engineering communities to repurpose validated code segments from one system to another or even cross system reuse, which will aid in an in-depth and detailed analysis of systems.

Currently, there is a need for systems engineers to establish efficient and convenient ways to build and use behavior models of a wide range of systems with MP. Having a repository with validated code templates will significantly reduce the time to build a model and enable future studies on usability and code reuse across system models in different domains. Such a model library containing code snippets and segments of behaviors will give a designer flexibility to quickly edit parameters and incorporate them into the model. It is also expected to aid with creating models with reduced bugs due to the reuse of existing code segments that are known to work, thus saving debug time. Changes made to the Monterey Phoenix user interface will create an additional feature for the user to access the repository along with a search function to find code snippets within the repository.

E. THESIS ORGANIZATION

Chapter I discussed the background and motivation for this thesis which led to the research statement, methodology, and benefits. Chapter II conducts literature review by analyzing material that supports this research and establishing the need for this research. Chapter III provides use case scenarios that describe user and MP-Gryphon interaction which inform the requirements for the design updates. Chapter IV discusses possible design concepts followed by the actual design implementation, testing, and evaluation. Chapter V provides conclusion, recommendations for this thesis, and future research.

II. LITERATURE REVIEW

This chapter begins by providing background of MP, followed by a review of various behavior modeling and simulation tools in the market today. These tools were studied to get an insight into the capabilities they provide that facilitate model reuse and further find gaps in the systems engineering domain where ideas and concepts from other domains inform requirements for MP. Additionally, this chapter reviews related work and previous research projects in the area of model cataloguing and reuse to further establish the need for this capability for MP.

A. MONTEREY PHOENIX INTRODUCTION

MP "is a Navy-developed framework composed of a language, approach and tool for system and process behavior modeling" (Giammarco 2019a, 3). It can be utilized to model behaviors and interactions in various fields such as systems, software, hardware, organizations, and environments. MP provides a language framework which allows users to input model descriptions which are then used to generate graphical outputs consisting of sequential flowcharts describing scenarios and interactions.

The MP language or code entered by the user is easy to learn and no prior programming knowledge is required. Knowledge of a basic set of commands is required for users to build an MP code to describe detailed and informative models. The MP code is simulated to generate event traces, which are flow charts describing all possible scenario variations within a user defined limit called scope. The user then reviews all generated event traces for validity either manually or with automated queries.

MP is the only tool in market "that generates scope-complete sets of event traces. That is, it automatically produces every possible combination of behavior from models of interactions among different systems, up to a user-defined scope limit" (Giammarco 2019a, 5). MP excels as predicting emergent behavior of modeled systems and processes. Emergent behaviors are often found when a system is deployed where separate behaviors interact to produce new and unforeseen behaviors. MP uniquely "models behaviors and

interactions separately, then combines them" (Giammarco 2019a, 6) to generate emergent behavior scenarios.

B. ANALYSIS OF MODELING AND SIMULATION TOOLS

The following subsections evaluate two modeling and simulation tools, OrCAD Cadence PSpice and Dassault Systems SolidWorks, utilized in electronics and mechanical engineering domain to understand how models are managed and reused within these tools. Specifically, the below subsections investigate model libraries, organization of models within these libraries, model access and search capabilities, naming convention, and reuse within the simulation tools.

1. OrCAD Cadence—PSpice

OrCAD Cadence is an industry recognized company that provides electronic engineers with a complete suite of solutions to develop designs with confidence and ease to meet evolving technology challenges. PSpice is one of the many solutions offered by OrCAD Cadence. PSpice is a modeling and simulation tool, which provides users with circuit modeling, behavior simulation, verification, and an analysis environment, which allow electronics engineers to refine designs prior to committing to fabrication. Additionally, PSpice Designer Plus is a second offering in conjunction with PSpice to conduct advanced analysis of designs. It provides capability for "simulation to maximize design performance, yield, cost-effectiveness, and reliability" (OrCAD Cadence PCB Solutions 2018).

a. PSpice Modeling and Simulation Requirements

To create a PSpice project, interfaces with other programs within the OrCAD Cadence product offerings and PSpice internal tools is required. Figure 1 shows major components that interface with PSpice to aid in creation of a project. This includes OrCAD Capture, Model Editor, and Model libraries. PSpice utilizes OrCAD Capture or Design Entry HDL for circuit schematic creation or design entry in preparation for simulation. A circuit schematic diagram normally includes graphical symbols representing various electronic components or parts, representative connection between parts, component

values, and other attributes that describe a design of equipment. Model Editor provides the user with the capability to create and edit model definitions of a part. Model Libraries are organized, searchable part repositories containing models and device definitions.

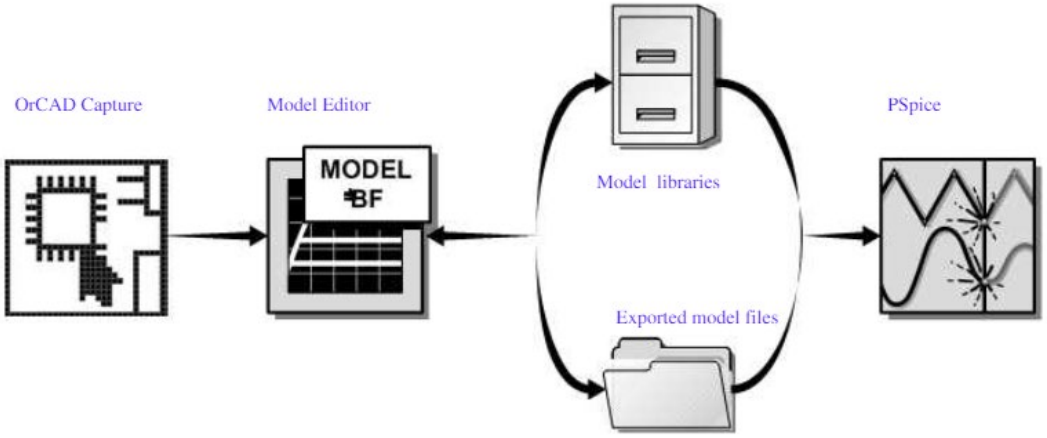

Figure 1. Relationship of PSpice with OrCAD Capture, Model Editor, and Model Libraries. Source: Cadence Design Systems (2019).

To simulate a design successfully in PSpice the following information is required:

(1) A schematic with components or parts and corresponding connections.

(2) Type of analysis

(3) Test stimulus definitions.

(4) Simulation models corresponding to parts in the circuit.

PSpice has the capability to perform various types of analysis such as DC, AC, time-based, and advanced analysis which encompasses temperature, parametric, sensitivity or worse case analysis, and statistical analysis such as Monte Carlo. The type of analysis to be conducted needs to be included in preparation for simulation. A stimulus definition is an input waveform, analog or digital, which defines "the shape of time-based signals used to test circuit's response during simulation" (Cadence Design Systems 2019, 44). Each part within a circuit schematic is associated with a model file which contains a behavior description of the part. Figure 2 shows a simple schematic of an operational

amplifier with all the necessary parameters such as supporting components, connections, values, inputs, and outputs along with the simulation output showing the behavior of the operational amplifier circuit under specified analysis conditions.

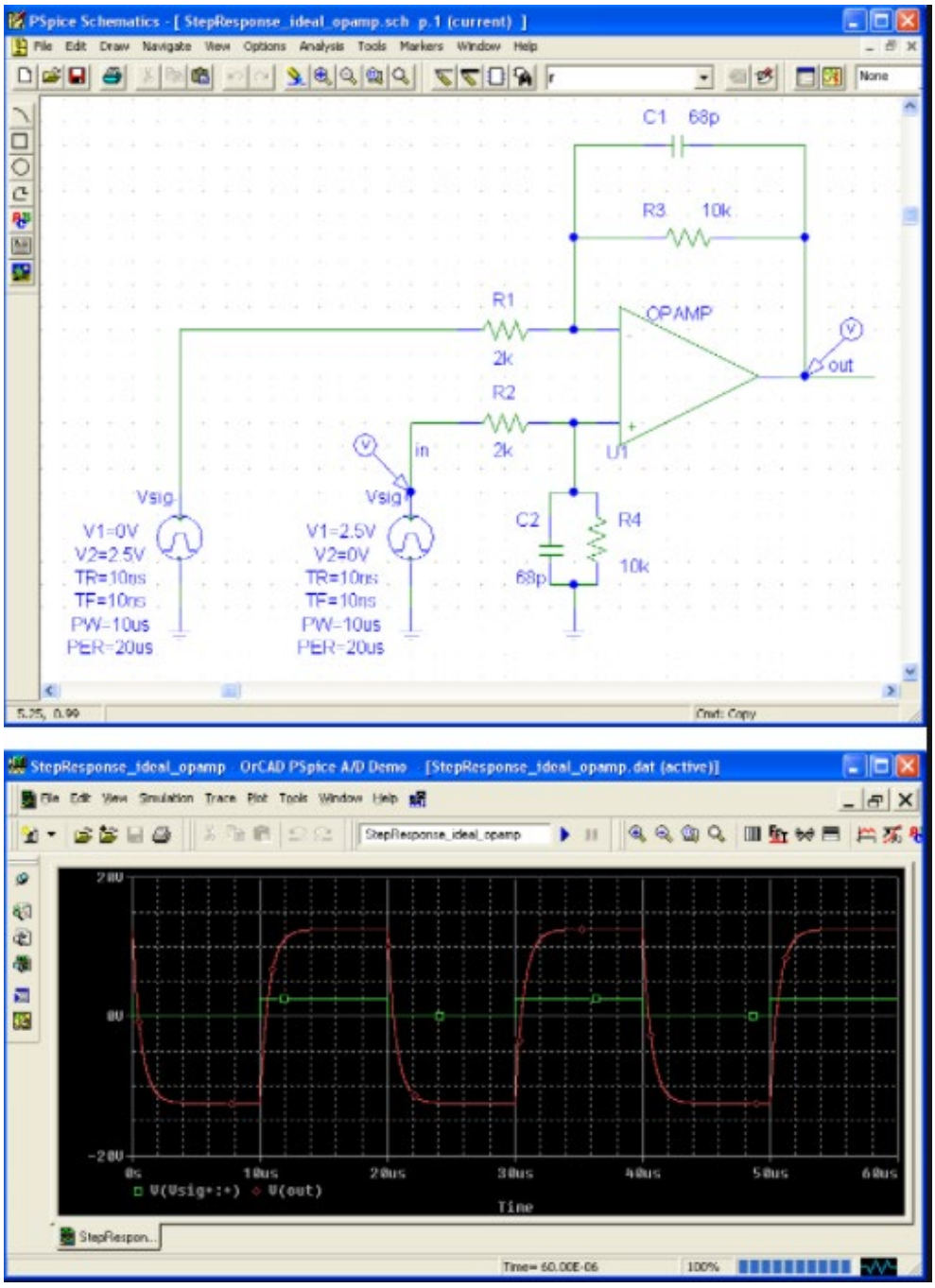

Figure 2. Operational Amplifier Schematic and Simulation Output. Source: onemilimeter (2011).

b. Part and Model Libraries

Thousands of electronic parts are available in market today and growing. This includes various part types that vary in function and each part type can have variants that are produced by multiple manufacturers. This leads to a need for a sophisticated approach to organizing, classifying, and managing parts and models within PSpice. Various types of electronic parts can be stored within the PSpice parts libraries, and each part type has varying characteristic that is applicable to specific areas of application. A part can contain information such as symbol, packaging, behavior description, and more. Symbol is useful for creating schematics, whereas part packaging data is useful during Printed Circuit Board (PCB) layout, and behavior description is needed to conduct simulation. Therefore, the type of information encapsulated within a part needs to be known prior to using the part. Basic parts are provided built in by default within PSpice. User can also add parts within libraries or add full libraries provided by parts manufacturer. PSpice also contains Model Editor which provides the capability to edit parts and models. PSpice provides two types of part libraries, Standard PSpice libraries and PSpice Advanced Analysis libraries. Standard PSpice libraries contain "over 16,000 analog and 1,600 digital and mixed-signal models of devices manufactured in North America, Japan, and Europe" (Cadence Design Systems 2019, 141). PSpice Advanced Analysis libraries consists of parameterized and standard parts.

> Parameterized parts include tolerance, distribution, optimizable and smoke parameters. Parameterized parts are associated with template based PSpice models. An important advantage of using the template based PSpice models is that you can pass model parameters as properties from the design entry tool. For example, if a template-based model is associated with a part, the model parameters that you specify on an instance of the part in you design will be passed to the model. There is no need to edit the model itself to change a parameter value. This is unlike the standard PSpice parts that are associated with device characteristic curve-based PSpice models, where you need to edit the model to change a simulation parameter. (Cadence Design Systems 2019, 34)

PSpice part libraries contain four types of parts suitable for simulation, vendor-supplied parts, passive parts, breakout parts, and behavioral parts. Vendor-supplied part libraries are parts provided by manufacturer. Passive parts are basic electronic parts

built-in PSpice parts libraries. Breakout parts "are passive and semiconductor parts with default model definition that define a basic set of model parameters" (Cadence Design Systems 2019, 152). Behavioral parts allow a user "to define how a block of circuitry should work without having to define each discrete component" (Cadence Design Systems 2019, 154). It is important to note that not all parts can be simulated. Parts without model definition associated to them cannot be simulated; however, they could exist in the library for other modeling purposes. A part that contains model definition or that can be simulated is saved in files called Model Libraries and has three properties. It has a model to describe the part's electrical behavior, the part must have electrical connections within the design, and it must translate from design part to netlist for PSpice to read.

c. Part Placement and Library Access

PSpice provides an easy and intuitive way to place parts into project. Figure 3 shows the OrCAD Capture user interface. On the right the Place Part pane contains list of all parts within a selected library. It also contains a list of applicable libraries for the project. User can add multiple libraries that will only be used for project under creation, these are called local libraries. User can select a library and all parts within the selected library will be listed under Part. Place Part pane also shows the symbol of the selected part and its value. When the part is placed into the schematic a reference designator is automatically assigned for the part. The same part can be placed within the design and a different value and reference designator can be assigned to part to differentiate it.

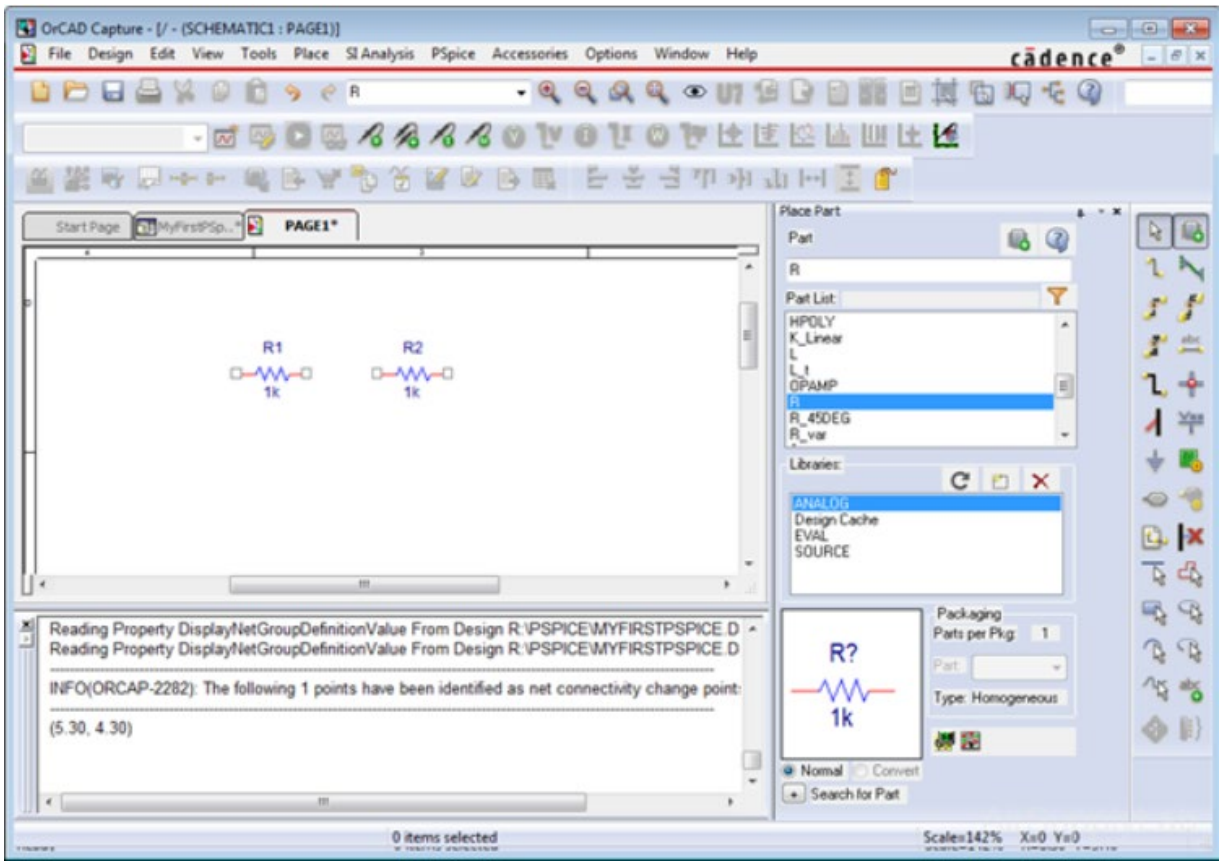

Figure 3. OrCAD Capture User Interface Illustrating Place Part. Source: Lin (n.d.).

d. Parts Naming Convention and Search

With extensive part libraries, parts management is an important capability within PSpice which allows users to handle the correct parts needed for designs. The naming convention of parts is critical to organizing parts and storing them at the appropriate library location. There are various types of electronic part types available with multiple manufacturers making the same part type. Within PSpice libraries the part names "usually reflect the manufacturer's part names. If multiple vendors supply the same part, each part name includes a suffix that indicates the vendor that supplied the model. For example, the PSpice libraries include several models for the OP-27 operational amplifier as shown in Figure 4. Notice that there is a generic OP-27 part provided by PSpice, the OP-27/AD from Analog Devices, Inc., and the OP-27/LT from Linear Technology Corporation" (Cadence

Design Systems 2019, 143). In addition to naming conventions, each part contains a part description to aid the user in recognizing the correct part. To find a part, a user can search the libraries by "restricting the parts lists to those names that match a specified wildcard text string" (Cadence Design Systems 2019, 144). "With easy access to parametric component data and part information, users can quickly identify and design with preferred components, reducing the amount of time spent researching parts. Parts can be queried based on their electrical, physical, or corporate characteristics and automatically retrieved for use" (OrCAD Cadence PCB Solutions 2014).

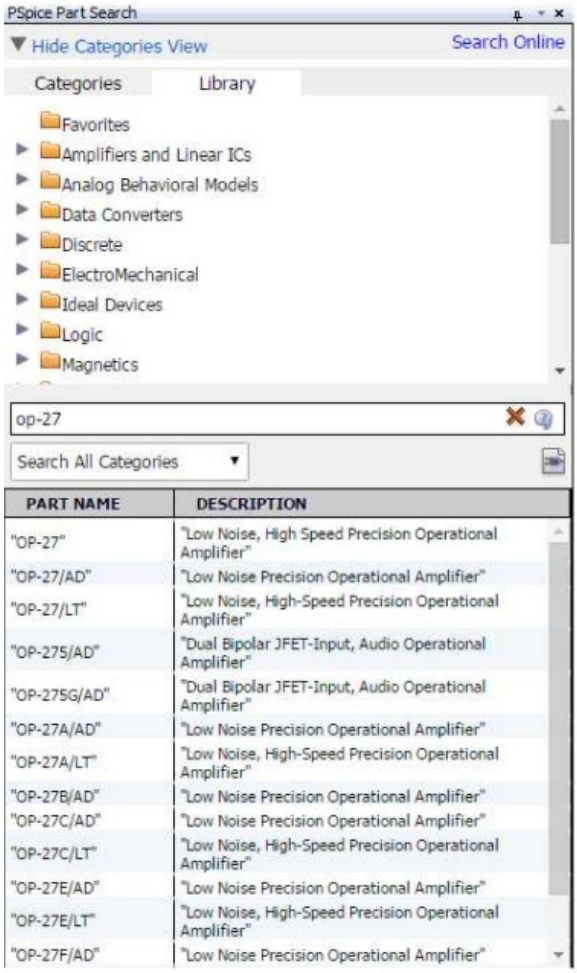

Figure 4. PSpice Part Search Feature. Source: Cadence Design Systems (2019, 144).

The capabilities provided by PSpice allows the user to create reliable products with confidence afforded by the libraries containing known design information. More importantly, it allows users to create designs with ease and provides an environment that enables reuse of models and designs through its architecture and user interface. It allows for users to maintain, manage, and search for parts within an extensive part and model library. It provides a default library when installed and provides capability for users to add more models and add edited models with the ability to change the behavior model of the parts in multiple ways. Through parametrized parts and Model Editor, a user can easily change the behavior definition of the model, which can be saved for reuse.

2. Dassault Systems—SolidWorks

SolidWorks is a solid modeling program released by Dassault Systems. It is a mechanical design tool that allows mechanical engineers to quickly create designs, examine and evaluate geometric features and dimensions, and produce three-dimensional models and detailed design drawings. Integrated within SolidWorks is SolidWorks Simulation. This is a design analysis system which provides simulation and analysis capability to predict a designs physical behavior to find design errors and mitigating them prior to its construction and release to field thus reducing time to market and cost. SolidWorks Simulation provides a whole array of analysis types such as linear and nonlinear static analysis, frequency analysis, thermal analysis, topology studies, harmonic analysis, random vibration analysis and more. Figure 5 shows stress analysis results on a part with external force applied to the part to find the location that is most at risk of failing based on the stress applied.

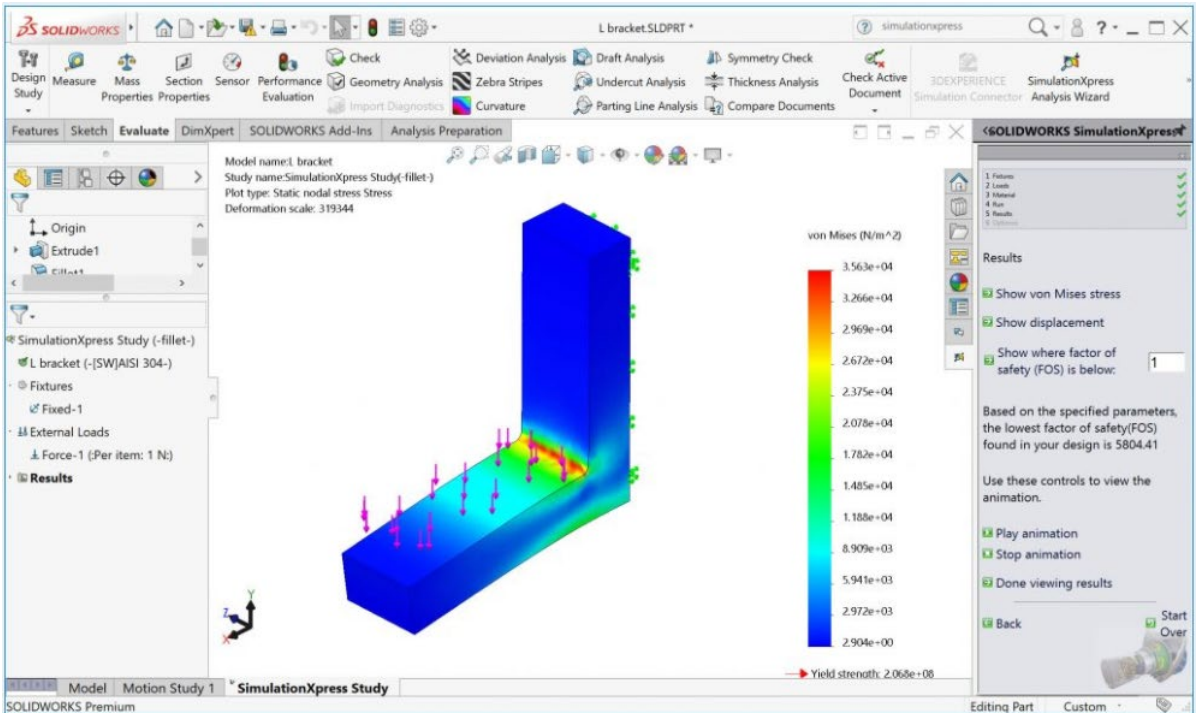

Figure 5. SolidWorks Simulation Results of a Bracket. Source: Javelin A TriMech Company (2017).

a. SolidWorks Simulation Approach

Prior to simulating a design in SolidWorks Simulation, a user needs to define the analysis type, parameters such as model dimensions, material property, external loads, component contacts and connectors. Parameters in SolidWorks are linked to numerical variables that can be changed or modified based on evaluation conditions. SolidWorks Simulation uses Finite Element Method (FEM), a numerical technique to simulate designs. FEM

> divides the model into small pieces of simple shapes called elements effectively replacing a complex problem by many simple problems that need to be solved simultaneously. The software formulates the equations governing the behavior of each element taking into consideration its connectivity to other elements. These equations relate the response to known material properties, restraint, and loads. Next, the program organizes the equations into a large set of simultaneous algebraic equations and solves for the unknown. (Dassault Systems 2021b)

b. SolidWorks Libraries

SolidWorks provides with multiple types of libraries which is are databases of many different types of data pertinent to creating and simulating models. SolidWorks provides design library, material library, and analysis feature library. Design library contains subdirectories with reusable parts or frequently used parts, assemblies, and drawings. Custom parts can also be added to design library. Figure 6 shows the Design Library pane which displays the parts contained in selected library. The material library is a database containing different types of materials that can be applied to parts. Custom materials can be created and added to the material libraries. Figure 7 shows the materials library and parameters that can be entered to customize materials. The Analysis Libraries Feature, "frequently uses analysis features (such as load/restraint, contact condition, etc.) that users can create once and save in a library for future use" (Dassault Systems 2021a). This provides a capability to define commonly used analysis features and store them based on application such as environment or operating conditions and allows for reuse of analysis features from one model to other similar models. The analysis libraries by default include subdirectories called Environment, Examples, Loads, and Supports. Since the models are parametrized, a user can edit parameters and save the edited model with new parameters for future use.

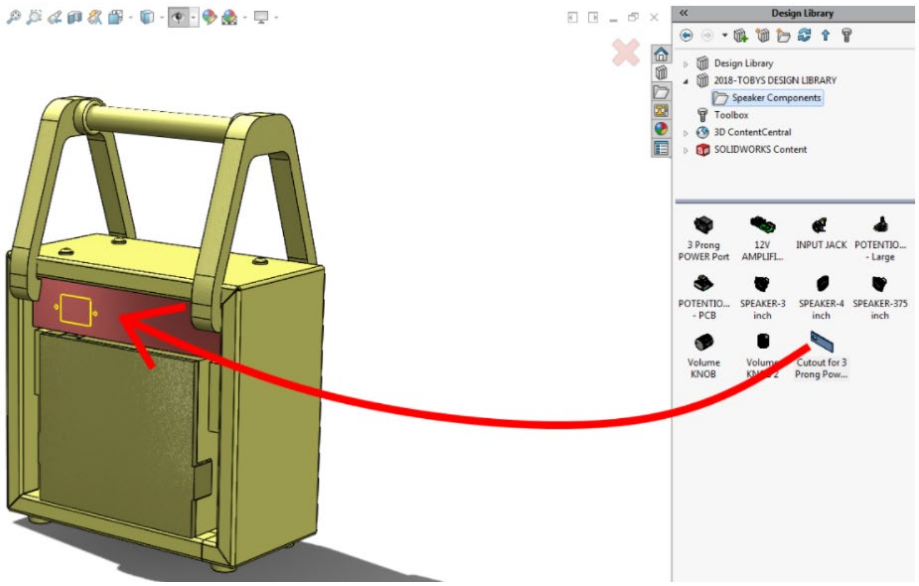

Figure 6. SolidWorks Design Library Showing Parts Available in a Selected Library Source: Schnaars (2018).

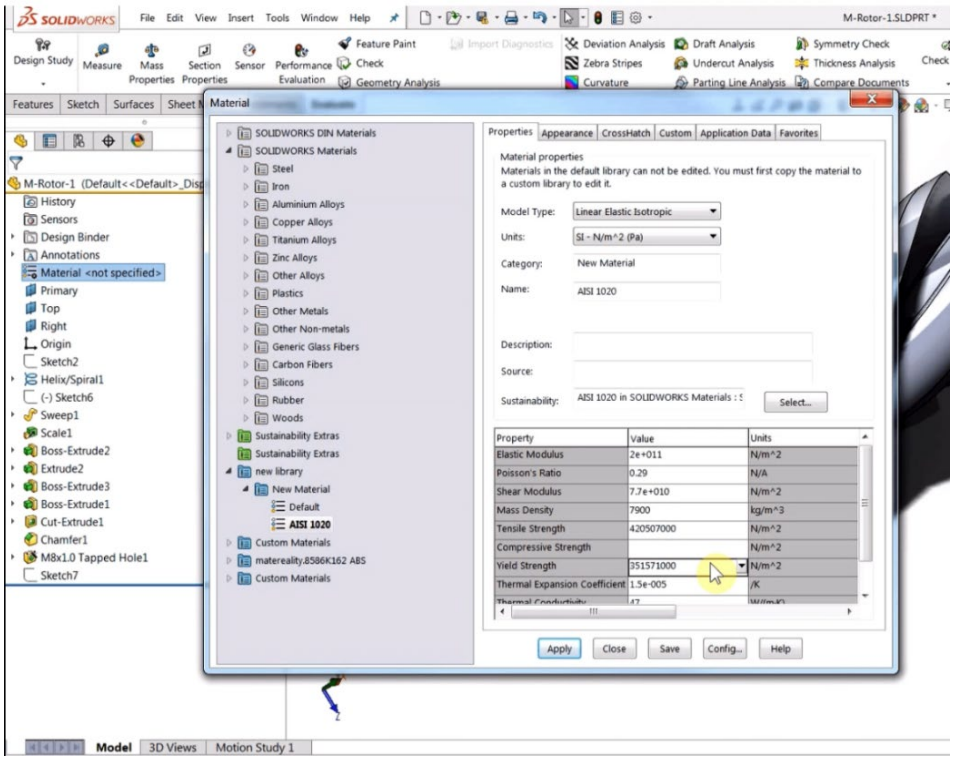

Figure 7. SolidWorks Materials Library Listing of Material Types and Editable Parameters. Source: Marrs (2017).

c. Access to Libraries and Search Capabilities

SolidWorks employs a visual approach to access and save parts and models within libraries. To save a part, a user can drag the part from graphics area to library area and add a description to the part being saved for future identification. The user can select the desired part from library area and simply drag and drop the part into the graphics area to add to design. The user can visually see the parts available in the library in tabular form and can easily pick it up for use. Figure 8 shows the design library of the SolidWorks user interface and dragging an analysis feature into the design. All available library types are listed within design library. Figure 8 also shows that analysis library called "mySimulationFeatures" is included within design library. The user can custom parts directories within the design library section. The SolidWorks Graphical User Interface (GUI) displays all the models available within the selected directory.

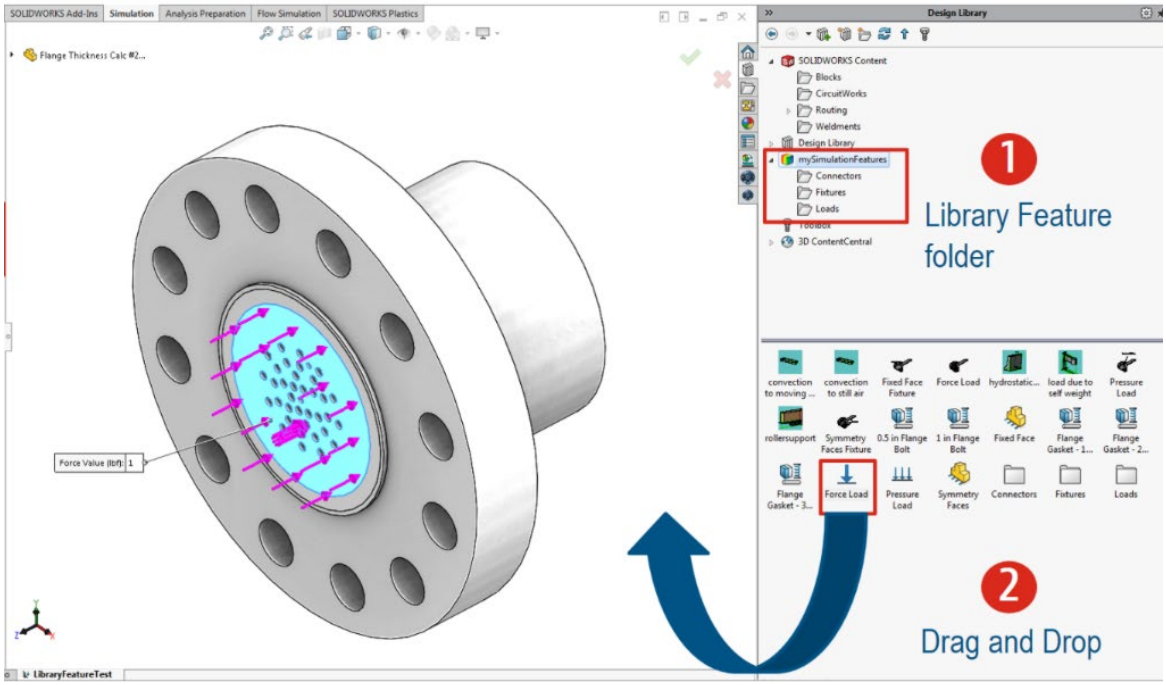

Figure 8. Screen Capture of SolidWorks User Interface with Illustration of Design Library and Component Selection. Source: Lakshmipathy (2019).

SolidWorks provides users with the capabilities to define behavior for mechanical parts using the FEM technique. These behavior models can be saved into analysis libraries, which can be reused in the future and applied to similar models. SolidWorks also provides the user with the capability to manage and organize libraries. It gives the user the ability to create and add new libraries.

C. NEED FOR A MODEL REPOSITORY WITHIN MONTEREY PHOENIX

This section summarizes findings from the review conducted of PSpice and SolidWorks and compares them to MP to understand similarities and differences between the three tools. From the comparison of the three tools, it identifies related missing features in the MP user interfaces. Finally, it highlights the opportunities for enhancement to MP user interfaces to make them comparable to the industrial tools reviewed with respect to code reuse.

1. Comparison of Modeling Tools

Although PSpice and SolidWorks modeling tools are utilized in two separate fields with varying applications the intention of both tools is to provide insight into the behavior of the designs, discover new requirements, and find emergent behaviors similar to MP's use in the systems engineering domain. PSpice and SolidWorks were selected to be analyzed and compared to MP behavior modeling tools because these tools provide behavior analysis of electronic circuits and mechanical systems, respectively. Table 1 compares the three tools based on the library features and capabilities provided by each tool.

Table 1. Comparison of Modeling and Simulation Tools

Comparison of Modeling Tools			
	MP	**PSpice**	**SolidWorks**
Library Content	Behavior Logic Models	Physical Object Models, Behavior Logic Models	Physical Object Models, Behavior Logic Models
Low Level Library Elements	Code snippets for behavior compositions within a schema	Component/part models	Component/part models
High Level Library Elements	Full schemas containing behavior patterns for interacting systems	Full designs	Full designs
Model Description	MP Language	Characteristic curve-based definitions, Template-based or parametrized	Finite Elements Method (FEM), Parametrized
Model Libraries	Default example library, more libraries proposed for development	Standard libraries, Advanced analysis libraries, Custom	Design libraries, Materials libraries, Analysis libraries, Custom
Model Search	Proposed for development	Yes	Yes
Reusable Models	Proposed for In development	Yes	Yes

Each tool handles model behavior descriptions differently because behavior of a part in its associated field is described by domain-specific measures. For example, the behavior of an electronic part can be described by an IV curve, which can be translated numerically into behavior. Therefore, within PSpice, behavior of a part can be described based on the characteristic curve of the part and through passing parameters via a template-based approach. Similarly, in SolidWorks, behavior is described by FEM or through parameters. Both PSpice and SolidWorks provide a parameterized approach to quickly change behavior of a model.

Both PSpice and SolidWorks by default provide extensive parts and model libraries that aid users in applying these models in a broad range of designs. Libraries aid users with the ability to organize and manage hundreds of parts. Along with providing the libraries, they provide the ability to easily create new libraries and edit models with detailed

descriptions such that models can be searched through libraries. Having built-in capabilities for parts and model storage and management provides the user with capability to reuse models from one design to other similar designs.

MP provides a built-in example library of broad range of models from various disciplines. In addition to example library, the user can save user-created models locally to create a personal model library. Other than the example model library, MP does not provide template-based or parametrized model storage that can be reused from one project to another. Additionally, it would be beneficial to have a capability to save models with descriptions so the user can search through a library when needed for future similar designs. At the time of this comparison, MP did not provide the user with the ability to search through the existing example library. The user needed to open each example to look for the desired behavior if it could not be gleaned from the example model file name.

2. Need for Model Libraries and Model Reuse

Having model libraries available for users creates opportunity for automating the process of incorporating known validated models within a design. Models from libraries can be customized and easily conformed to match different, but similar requirements for another system. This creates opportunities for sharing models and collaborating with other systems that can be varied to output multiple different outcomes. Having model libraries encourages reuse of models by enabling harvesting from existing system models. This will reduce the time it takes to build a model in MP along with providing the user flexibility to quickly edit code parameters and incorporating them into the model. Lastly, incorporating reusable proven models or snippets within the design reduce code errors or bugs which will further save time for the user.

III. LIBRARY USE CASES AND REQUIREMENTS

This chapter analysis the current architecture of the MP-Gryphon GUI to develop context for where changes in GUI architecture are to be made in support of the research. This chapter lays out the high-level architecture of the MP tool suite. This chapter also analysis the interaction between MP user and current MP-Gryphon GUI followed by generation of use cases describing user and updated MP-Gryphon GUI with new library features. Lastly, this chapter provides the requirements for implementing the new proposed changes.

A. MP ARCHITECTURE ANALYSIS

MP architecture was studied to understand the mechanics of how the MP tool suite functions. This information aids in understanding where changes need to be made within the architecture to implement the new library features. The design elements described in this chapter leverage the existing features of the MP GUI to quickly implement the code snippet library function.

Figure 9 shows a hierarchical decomposition of all MP tool suite entities. The first level of decomposition of MP consists of MP-Firebird, MP-Gryphon, and Trace Generator. MP-Firebird and MP-Gryphon are two different GUI applications for MP. MP-Firebird is a web-based application accessible through the public domain and requires no application installation by the user. MP-Gryphon is an open-source application available for local installation of MP. The Trace Generator System takes MP schemas from MP-Firebird or MP-Gryphon, "produces exhaustive set of all valid behaviors for a given scope" (Auguston 2020), and returns those results back to MP-Firebird or MP-Gryphon. The modeling experience from a user's perspective is identical in both MP-Firebird and MP-Gryphon with the only exception being location of certain tools within the GUI. This thesis is directed specifically at MP-Gryphon because it is locally accessible, and the entire MP community will not be affected by the changes due to developmental efforts of this thesis. The objectives of this thesis can be implemented, tested, and validated in a localized environment of MP-Gryphon; therefore, MP-Gryphon is further decomposed. The second level of decomposition consists of User/ Modeler/Designer, MP-GUI, and Trace Generation Manager.

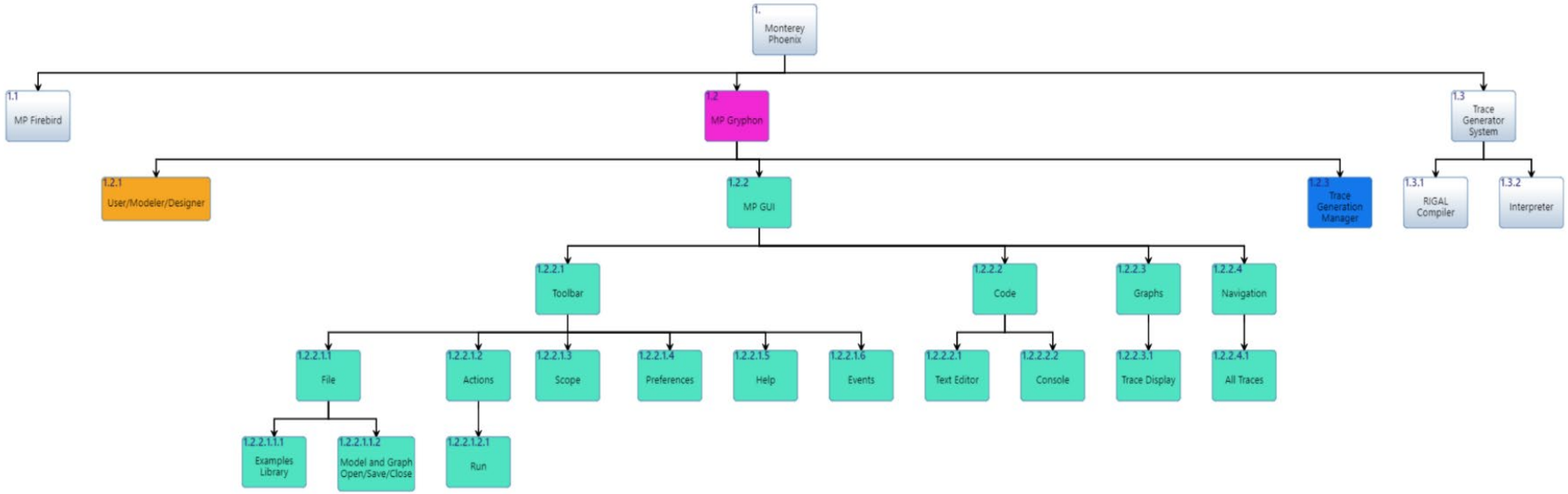

Figure 9. MP-Gryphon Architecture Decomposition

1. User/Modeler/Designer

The User, Modeler, or Designer is the individual utilizing MP-Gryphon to construct a system model followed by simulating the model to review traces or scenarios generated by MP-Gryphon.

2. MP-Gryphon Graphical User Interface (GUI)

The MP-Gryphon GUI allows interaction between User and the Trace Generation Manager to create simulated behavior scenarios of the modeled system. The MP-Gryphon GUI is decomposed further to five levels. MP-Gryphon GUI contains Toolbar, Code, Graphs, and Navigation features.

a. Toolbar

Toolbar provides users with control elements consisting of File, Action, Preferences, Help, and Events menus. Figure 10 shows a screenshot of the MP-Gryphon GUI highlighting the main components of the GUI. The File menu consists of input and output control elements that enable the user to access the MP Example Library provided by default, save models and graphs, and open models saved at a local location. The Action menu contains Run and Scope controls. When Run is selected, the model is sent to the Trace Generation Manager for compilation of traces. Scope allows user to set the number of maximum loop iterations for the model.

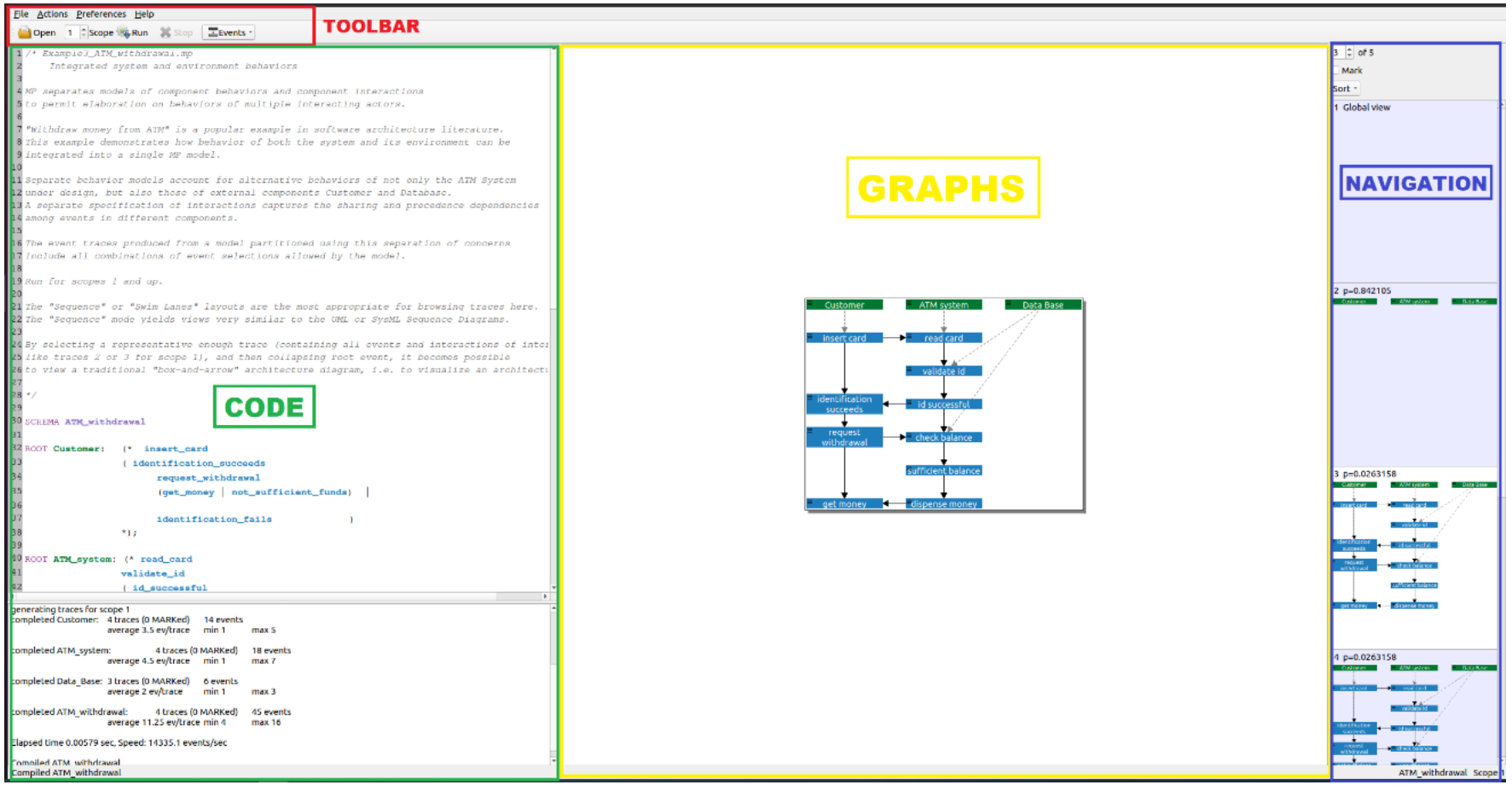

Figure 10. Monterey Phoenix Gryphon Graphical User Interface Widgets

b. Code

The code area contains Text Editor area which is where the user can type an MP schema (behavior model) or load an example of locally saved model. The Code area also consists of a console panel at the bottom which "provides statistics about the trace derivation process, such as the average, min, and max number of events per trace, and elapsed time. When trace generation is complete, the console window will display how many traces were generated" (Giammarco 2019b, 7).

c. Graphs

Graphs area displays the traces generated by MP in a sequenced format, "which positions the root events at the top and stacks included events directly below its respective root"(Giammarco 2019b, 8).

d. Navigation

Navigation "displays a thumbnail view of generated traces under the total number of traces generated for that run" (Giammarco 2019b, 8). It also includes the sort option used to organize the generated traces.

3. Trace Generation Manager

Trace Generation Manager collects the MP schema from the code area along with scope information and provides it to the Trace Generator System. It handles sending and receiving information between GUI and Trace Generator System. The RIGAL Compiler and Interpreter are subsystems of the Trace Generator System. The RIGAL Compiler and Interpreter consist of tools that process MP code – human-readable language – and translates it into a programming language that gets converted into traces. Some of the functions include syntax analysis, identifying allowable sequences, writing an abstract syntax tree, compiling C/C++ files, and executing C/C++ files to output JavaScript Object Notation (JSON) file. The JSON file is sent back to GUI, which contain the user readable traces. Figure 11 shows a screen capture of the MP architecture model, illustrating the behavior of the Trace Generator System and its interaction with the MP GUI.

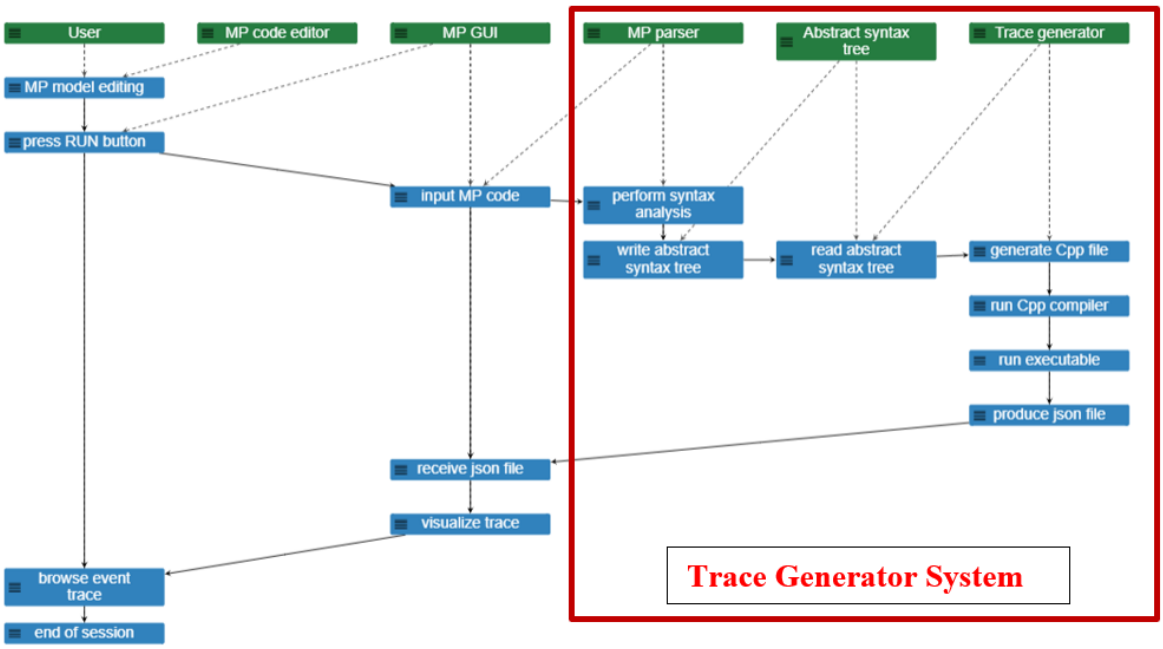

Figure 11. MP Compiler/Trace Generator Architecture Model. Adapted from Monterey Phoenix (2021).

B. USE CASES OF MP-GRYPHON GUI

Use of MP-Gryphon GUI was further evaluated by analyzing the current use of the GUI and by creating use cases to show different scenarios in which the GUI could be used with new feature implementation. The use cases illustrate interaction between the MP user and MP-Gryphon GUI. First, the interaction between the user and the current version of MP-Gryphon GUI is illustrated by a flow chart diagram. Next, an activity diagram is generated using MP-Gryphon simulation that illustrates how a user could utilize the model libraries and insert snippets of code within the existing model. Lastly, multiple scenarios show user and GUI interaction with new library features if implemented within the GUI. This aids in the visualization between differences in the use of the GUI before and after the implementation of new GUI features that includes model libraries.

1. Current MP-Gryphon GUI and User Interaction

The interface between user and MP-Gryphon GUI is illustrated by Figure 12. The list below summarizes actions that need to take place to create models and simulate them.

- To begin writing the MP code to describe a system's behavior, a user will either import a user-created model stored at a local location, import an example provided along with the software, or write code from scratch.
- A user can review and edit the model or adjust an example model to current required simulation conditions and enter changes accordingly.
- Once MP code is written, the user sets the scope and clicks Run to execute and send the model for trace generation.
- If the Run is successful, the GUI will display the traces in the graphs and navigation area for the user to review. If the Run is not successful, then the user should check for syntax errors or notes returned of failures in the Console. User reviews code and makes corrections in the code accordingly.
- After reviewing the traces, the user can either increase the scope of the analysis and rerun the code to evaluate results of the increased scope criteria or export the traces, model, and/or diagram.

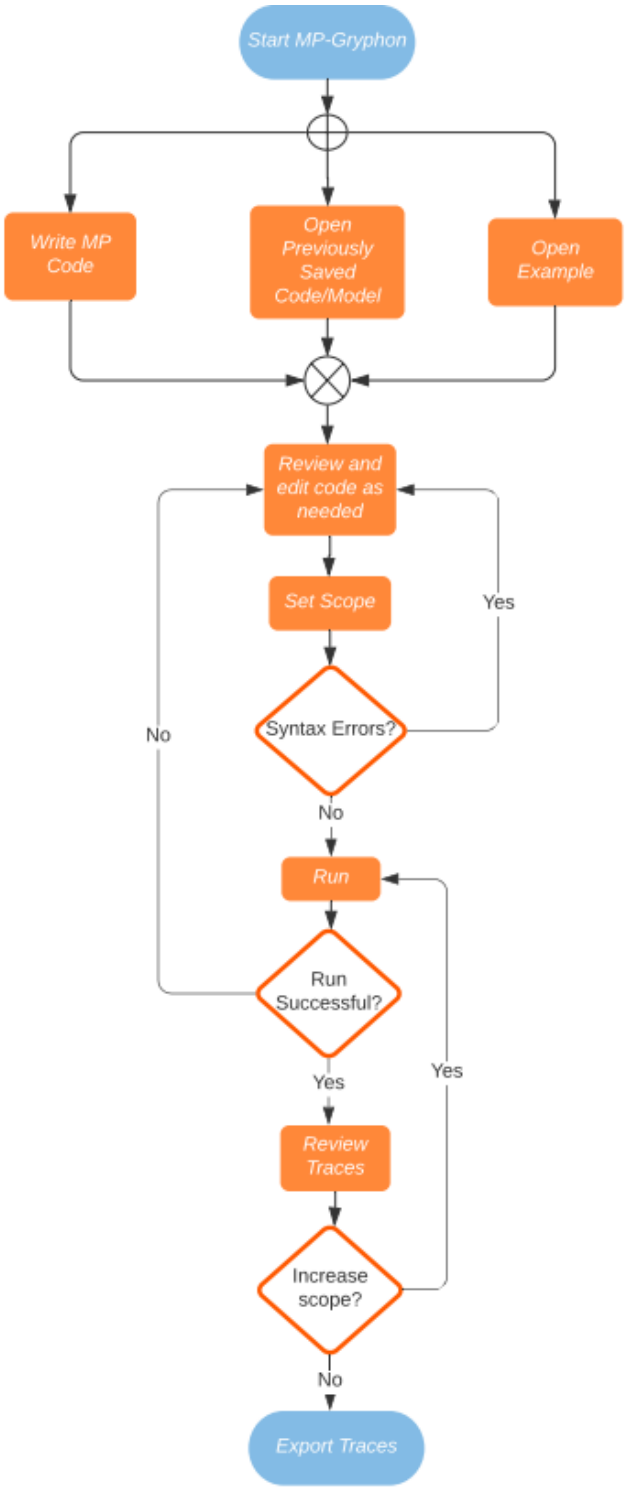

Figure 12. User and MP-Gryphon Interaction Flow Chart

2. User and MP-Gryphon GUI Use Cases

MP-Gryphon was utilized to generate its own use case scenarios to understand the interaction between the user and the new features that will be added to support model libraries. This will aid stakeholders in understanding how this new capability will change the flow of simulating models while reusing MP code. The below steps summarize the flow of a user's actions to simulate in MP-Gryphon with the new features. Figure 13 shows a corresponding activity diagram of the user's behavior.

- To describe a system, a user will either start by loading a preexisting code from example libraries or other source or write new MP code. User reviews and edits the code accordingly.
- A user could choose to incorporate snippets of MP code available in libraries to add to the existing code, followed by a review of the MP code to ensure no errors.
- The user runs the code, which is passed to the Trace Generation System.
- If the Run is successful, the GUI will display the traces in the graphs and navigation area for the user to review. If the Run is not successful, then the user should check for syntax errors or notes returned of failures in the Console. User reviews code and makes corrections in the code accordingly.
- After reviewing the traces, the user can either increase the scope of the analysis and rerun the code to evaluate results of the increased scope criteria or export the traces, model, and/or diagram.

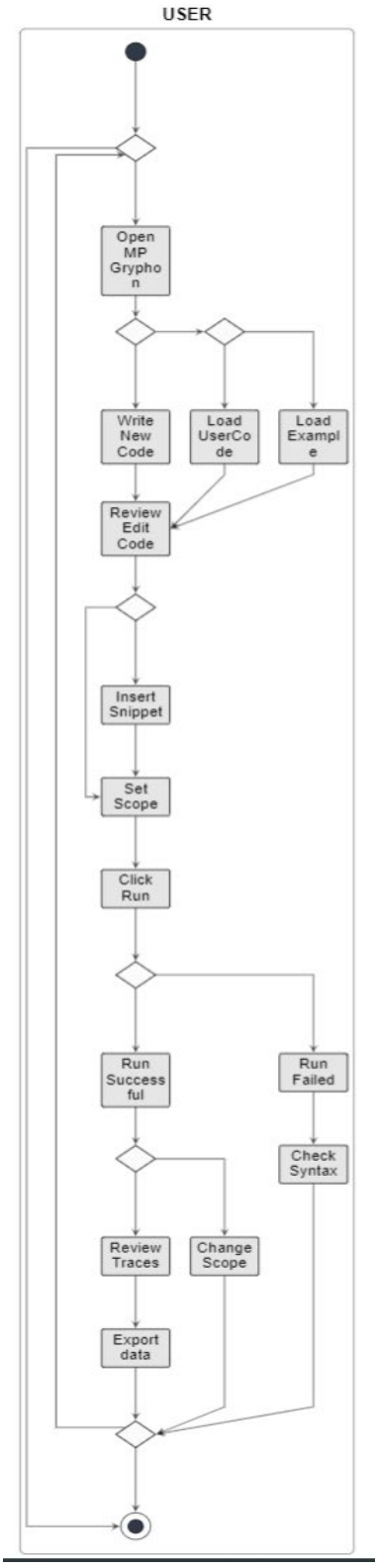

Figure 13. MP-Gryphon User Activity Diagram Illustrating the User and MP-Gryphon GUI Interaction with New GUI Features

Furthermore, MP-Gryphon model yields various use case scenarios that illustrate the interaction between user and MP-Gryphon GUI. Figure 14 shows six different use case scenarios that could occur when a user chooses to write an MP code, with three scenarios where an MP snippet could be added from the libraries, followed by running the code. If the run is successful, then the user reviews the code and exports the resulting traces or increases the scope and reruns the entire code for further analysis. If the run is not successful, then the user is notified to check syntax or details on the error is provided in the console display. Similarly, Figure 15 and Figure 16 show six use case scenarios if the user chooses to import examples or import a previously saved user code from libraries.

The modeled use cases of how a user interacts with the MP-Gryphon GUI informs the requirements and are also used as high-level test cases for the new implementation of adding model libraries. It gives insight into specific details such as the controls users will need to access libraries, which will lead to additional requirements towards the design of the new feature.

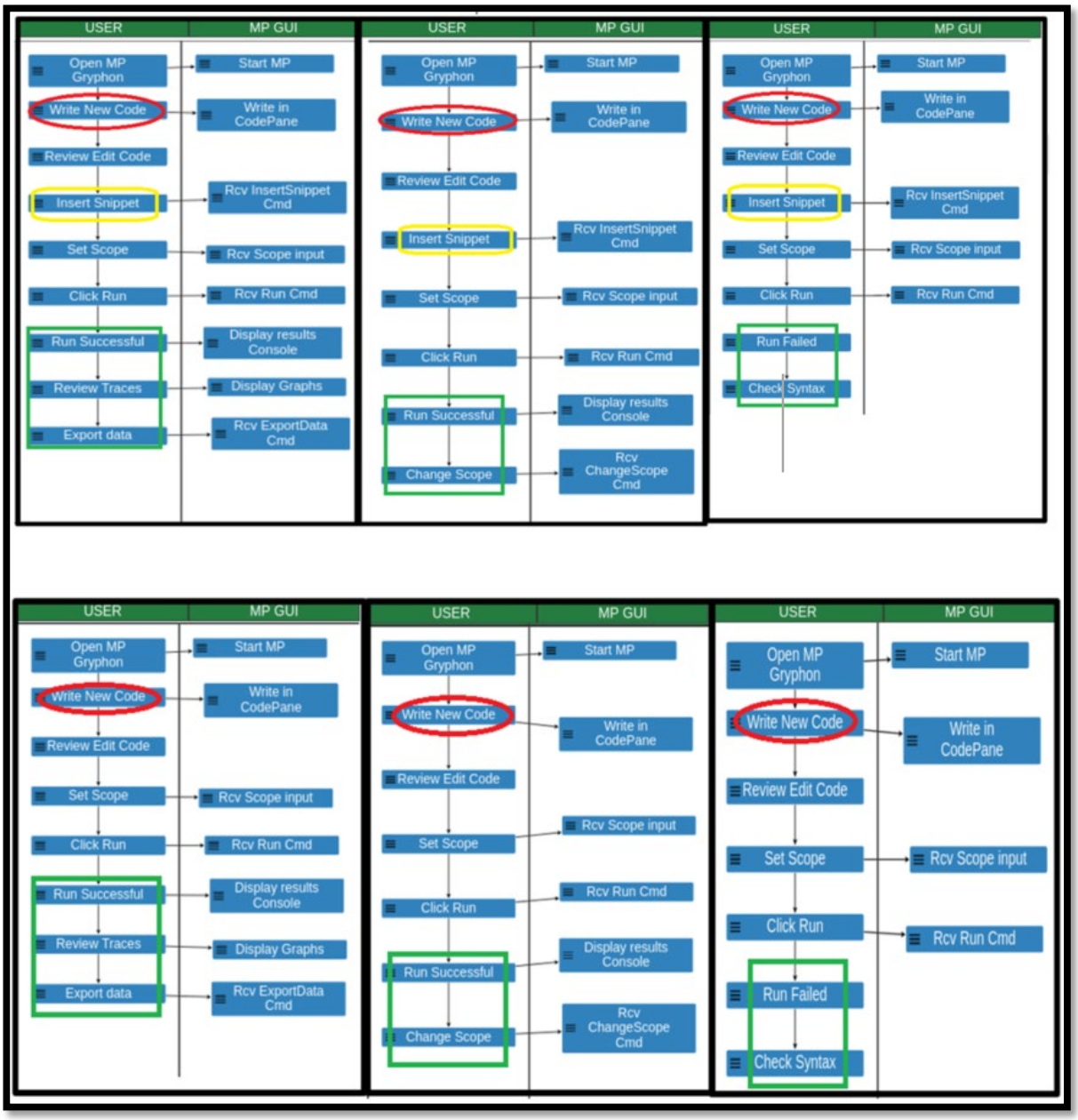

Figure 14. Event Traces for Use Case Scenarios if User Writes New MP Code

Figure 15. Event Traces for Use Case Scenarios if User Imports MP Code from Existing Libraries

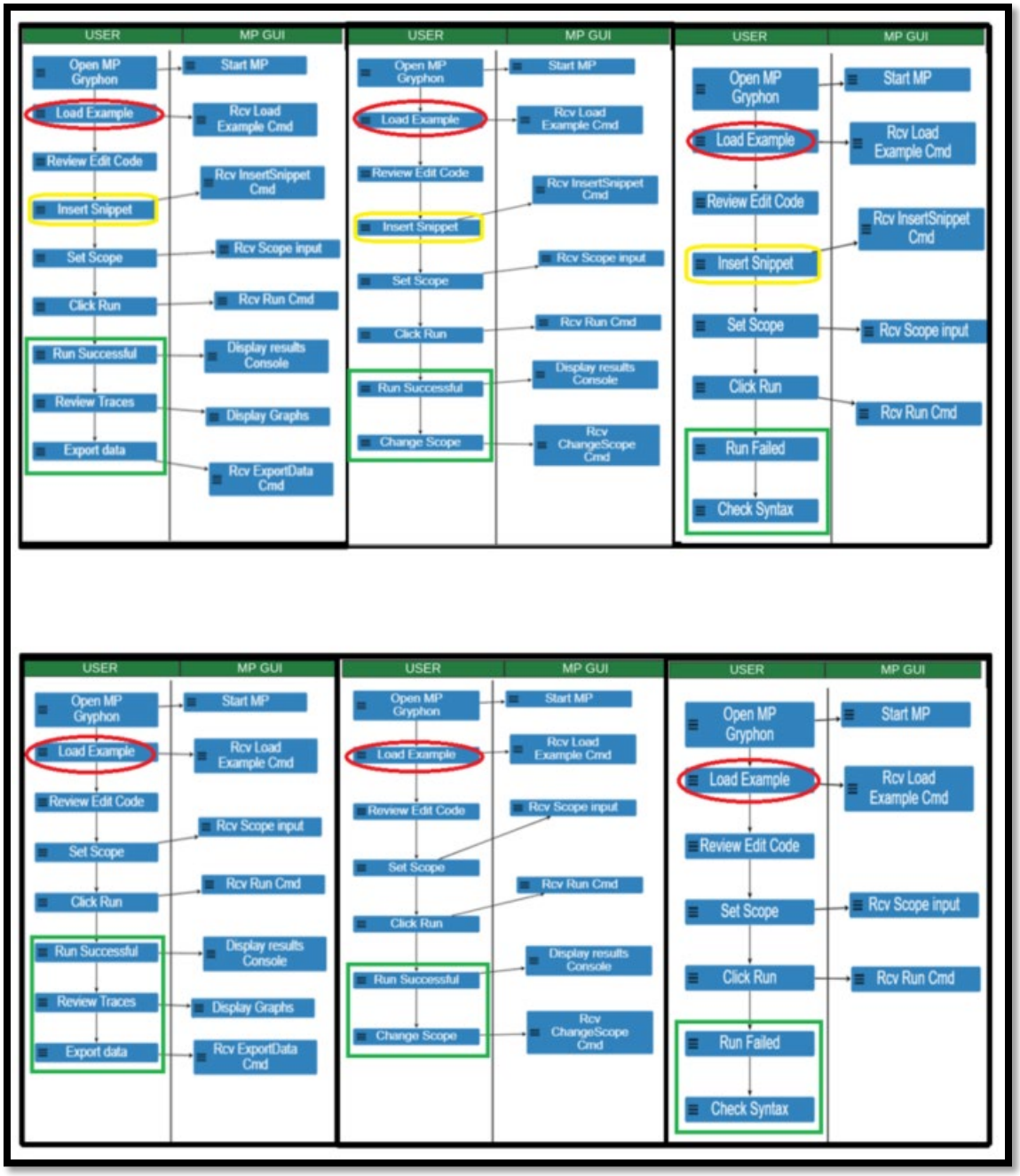

Figure 16. Event Traces for Use Case Scenarios if User Imports MP Code from Examples Library

C. REQUIREMENTS FOR MP-GRYPHON UPDATES

This section lists the requirements for updating the MP-Gryphon GUI to support libraries of reusable models. Functional requirements consisting of Input, Output, and functions requirements are listed within Table 2. The requirements generated cover features that support providing MP libraries of different types, access to the libraries, and searching within the libraries.

Table 2. MP-Gryphon Requirements for Implementation of Libraries

1. Functional Requirements
1.1. Input Requirements
1.1.1. MP-Gryphon GUI shall accept user request to import model code from a preloaded library.
1.1.2. MP-Gryphon GUI shall accept user request to import snippet code from a preloaded library.
1.1.3. MP-Gryphon GUI shall accept user request to save MP model code to a user-selected library.
1.1.4. MP-Gryphon GUI shall accept user request to save MP snippet code to a user-selected library.
1.1.5. MP-Gryphon GUI shall accept user request to search model code within libraries
1.1.6. MP-Gryphon GUI shall accept user request to search snippet code within libraries
1.2. Output Requirements
1.2.1. MP-Gryphon GUI shall provide a notification when model code is imported successfully from libraries.
1.2.2. MP-Gryphon GUI shall provide a notification when snippet code is imported successfully from libraries.
1.2.3. MP-Gryphon GUI shall provide a notification when model code is saved successfully to libraries.
1.2.4. MP-Gryphon GUI shall provide a notification when snippet code is saved successfully to libraries.
1.2.5. MP-Gryphon GUI shall provide user-requested search results of the library database.
1.3. Functions Requirements
1.3.1. MP-Gryphon GUI shall store MP model code.
1.3.2. MP-Gryphon GUI shall store MP snippet code.
1.3.3. MP-Gryphon GUI shall provide access to model libraries for user to save models.
1.3.4. MP-Gryphon GUI shall provide access to model libraries for user to open models.

1.3.5. MP-Gryphon GUI shall provide access to snippet libraries for user to save snippets.
1.3.6. MP-Gryphon GUI shall provide access to snippet libraries for user to open snippets.
1.3.7. MP-Gryphon GUI model libraries shall be searchable by model name.
1.3.8. MP-Gryphon GUI model libraries shall be searchable by keywords.
1.3.9. MP-Gryphon GUI snippet libraries shall be searchable by snippet name.
1.3.10. MP-Gryphon GUI snippet libraries shall be searchable by keywords.
2. Non-Functional Requirements
2.1. MP-Gryphon GUI shall provide flexibility in organization of the libraries.
2.1.1. Model libraries provide the users with the ability to add subdirectories within libraries based on model types.
2.1.2. Snippet libraries provide the users the ability to add subdirectories within libraries based on snippet types.

IV. MP-GRYPHON LIBRARY DESIGN AND TESTING

This chapter discusses the design process for this thesis which includes conceptualization of the design, the final design implementation, test and evaluation with analysis of the test results, and an illustration of the newly implemented features within MP-Gryphon GUI.

A. CONCEPTUAL DESIGN

Multiple design concepts are evaluated to ensure the required features are implemented to meet the thesis objectives. First the library types to be incorporated in the design are identified and proposed. Library type description and contents are discussed. Concepts for library access are also discussed, including import, export, and search capabilities.

1. Proposed Library Types

The following subsections describe different types of libraries that can be added within MP-Gryphon. There are three types of libraries described in the below sections: Preloaded Example Models Library, Preloaded Example Snippets Library, and Personal Collection Library. Each library type has distinct characteristics as to what should be stored in them.

a. Preloaded Example Models Library

The Preloaded Example Models library contains example executable MP models that are currently provided along with the MP-Firebird and MP-Gryphon tools. This is an extensive library containing proven example models from various interdisciplinary fields. Currently, MP-Gryphon is installed with 64 example models and growing. These are basic models of various applications that contain detailed description of the contents and the what the model does. These example models are intended to be used as baseline to build a user specific model or can be used for training purposes to learn MP language.

b. Preloaded Example Snippets Library

The Preloaded Example Snippets Library contain snippets or excerpts of MP code that can be plugged into a larger model. These are not entire system or process models; these could be frequently used proven code snippets that can be quickly integrated into a larger executable model. For example, a user might not have the syntax memorized for a Coordinate statement or the various ways it can be utilized. Having a collection of all the different ways to write a Coordinate statement readily available to import is beneficial for the user and aids in quick development of the model code.

c. Personal Collection

The Personal Collection is a customizable library where a user can save specific types of models or snippets unique to the users need. Libraries can be locally created by user and organized based on function type. For example, a user can create exclusive libraries to store executable models created to conduct risk analysis in a library specified for risk related models. Another example can be a user can create a library consisting of snippets specific to customer system types. Having a user-defined library provides flexibility within the MP-Gryphon tool for the user to create a custom library and organize it to the users liking.

2. Library Access Concepts

Multiple ways of accessing the libraries were conceptualized. First the current MP-Gryphon GUI was reviewed to see if any existing features can be leveraged for the new design. The current GUI allows users to open or close personal model files or example model files from File menu. Figure 17 provides a mockup of updates that can be made by expanding the File menu to include access for the proposed library types. The second concept includes implementing a dialogue box when a user requests import or export of models or snippets, where a user can select a library type to save model to or open model from. The dialogue box also includes a search feature which allows user to search within selected libraries based on user-specified text. Figure 18 displays a mockup of the dialogue box containing the library selection, search criteria entry location, and search result area. Lastly, the import and export of snippets need to occur such that the snippet can be inserted

at a user-defined location within an existing code file, or the user can select a snippet of code to export from an existing larger model code file. The easiest way to do this quickly is by providing the user with the option to place cursor or select snippets and right click with options for snippet library access. Figure 19 illustrates selection of snippets to be saved and when right clicked the menu will contain import or export option.

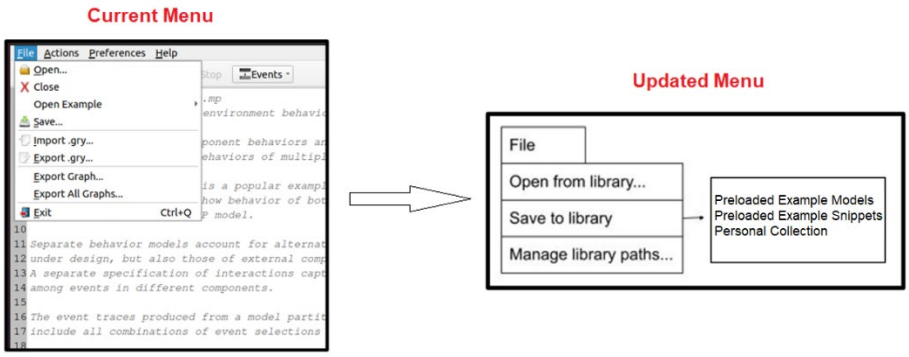

Figure 17. Mockup of Adding Updates to Existing File Menu to Access New Library Types

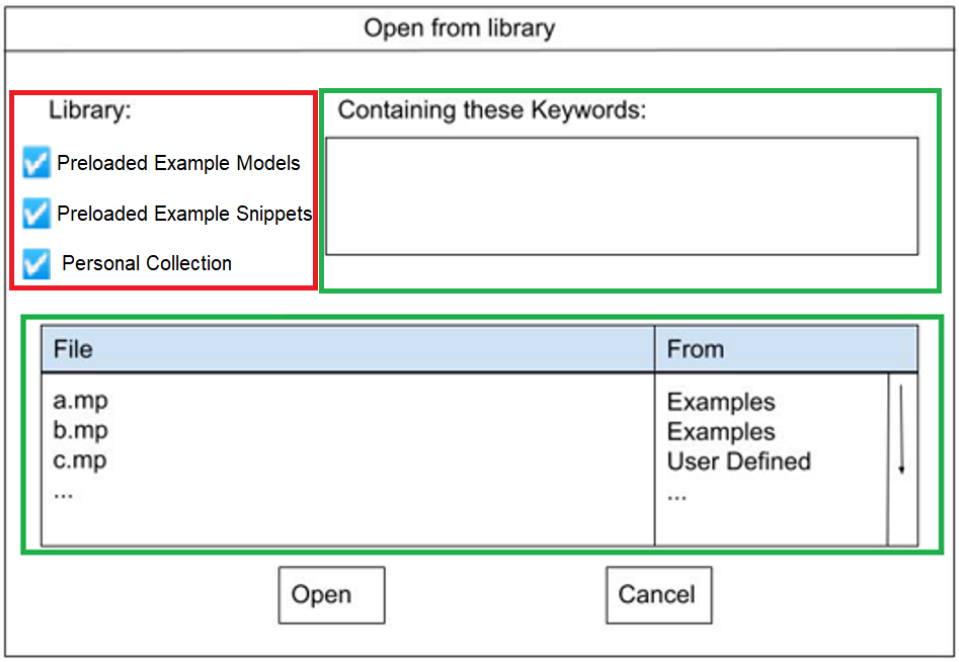

Figure 18. Mockup of Dialogue Box Containing Library Type Selection and Search Feature

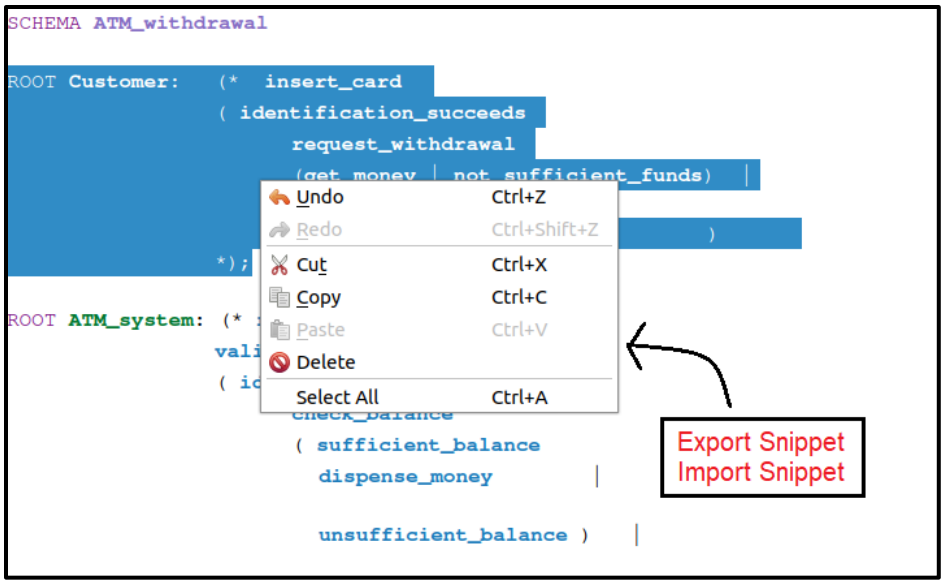

Figure 19. Mockup of Import/Export of Snippets from Text Editor

B. DESIGN IMPLEMENTATION

The design updates developed by this thesis research were implemented in the MP-Gryphon GUI Python source code by Bruce Allen, Faculty Associate- Research, at Naval Postgraduate School. The below subsections discuss the final design implementation to support library features within MP-Gryphon. The updated architecture is discussed to understand where within the MP-Gryphon architecture the design changes have been made. This discussion is followed by a discussion on how to access the libraries through the MP-Gryphon GUI. Specifically, the subsections discuss where the users can find access points to the libraries within the GUI along with how to import data into libraries and export data from the libraries. Finally, the addition of the search capability is discussed.

1. MP-Gryphon Updated Architecture

The three new library types can be accessed from File or Edit menu within the MP-Gryphon GUI. Figure 20 shows the MP-Gryphon architecture and the new additions to the GUI to visually see where the libraries can be accessed from the GUI.

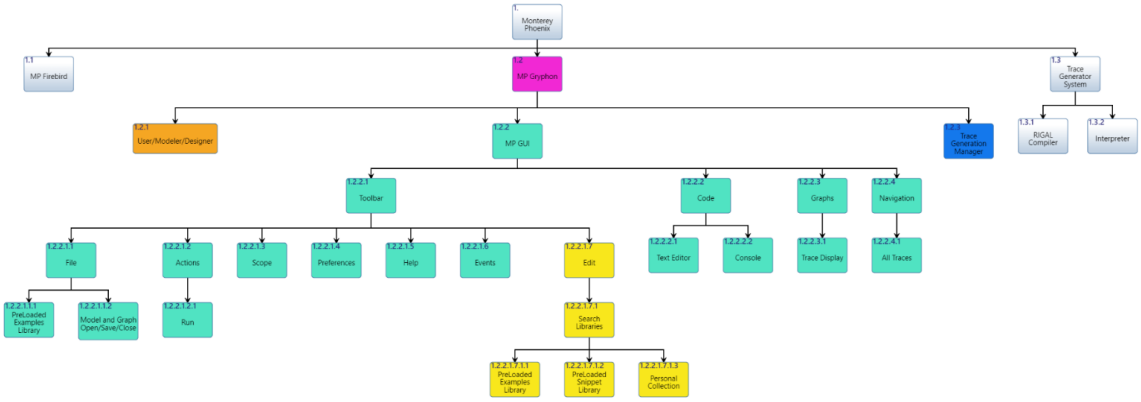

Figure 20. MP-Gryphon Architecture with Addition of Libraries Highlighted in Yellow

2. Import and Export from Libraries

A user has multiple options to import whole models or snippets into the Text Editor area of the code pane. The below subsections discuss how users can open and save to libraries.

a. Import From File Menu

Preloaded example models and full models from the personal collection can be imported from the File menu by a selection of File followed by Open .mp or Open Preloaded Example Models (Figure 21). A dialogue box opens where user can select library type and file to import the entire MP model into Text Editor with Code pane.

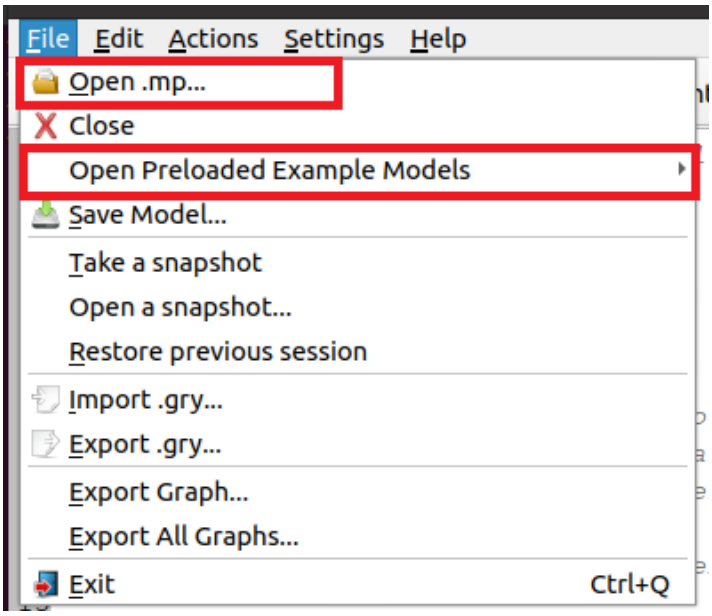

Figure 21. Illustration of Import from File Menu

b. Import from Edit Menu

To import code from the preloaded example model library, preloaded example snippet library, or personal collection, a user selects Search from Edit menu. When Search is selected a dialogue box will open where user can select the library type and library path to import MP code from. The top left portion of Figure 22 lists the three library types for user to select a model or snippet from within the implemented dialog. Once the user selects library type and the model or snippet to import, a View window opens to display the contents of the selected model or snippet file (Figure 23). Within the View window the user can either select the entire content of the file or select excerpts to import to the Text Editor when the user clicks Open. If a snippet is being imported, then the user should insert the cursor at the location where the snippet will be imported within the text editor prior to searching for a snippet.

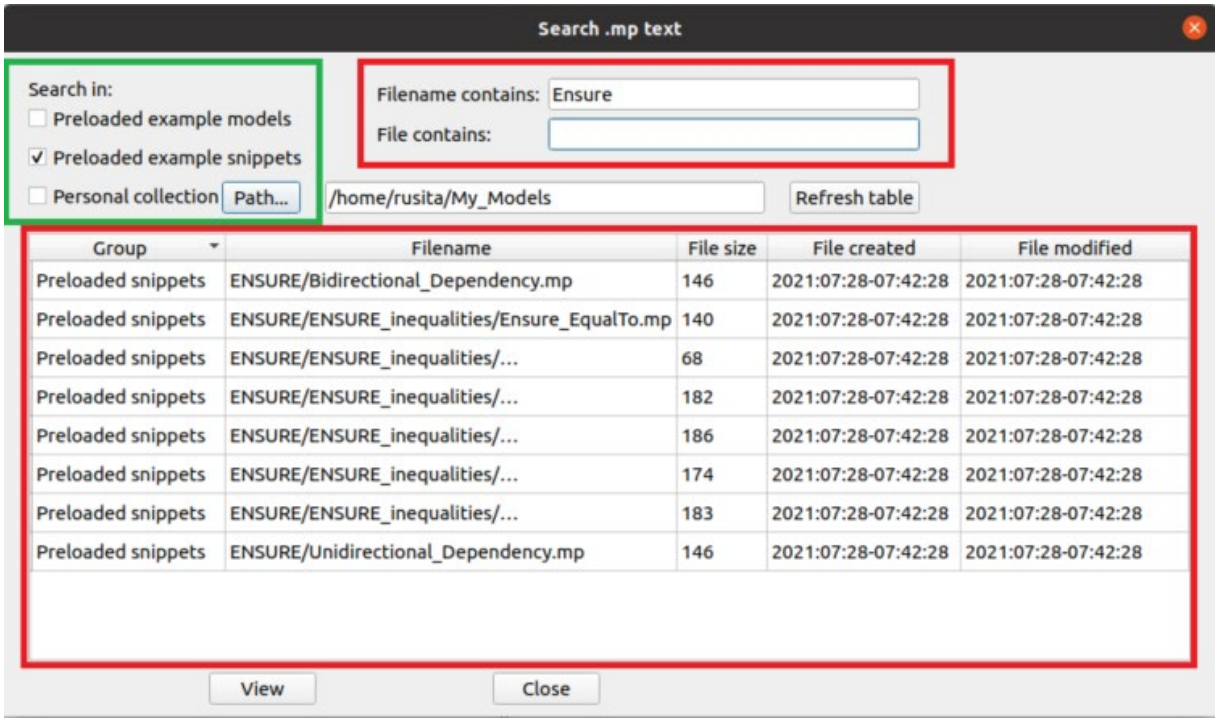

Figure 22. Design of Search Dialogue Box from Edit Menu

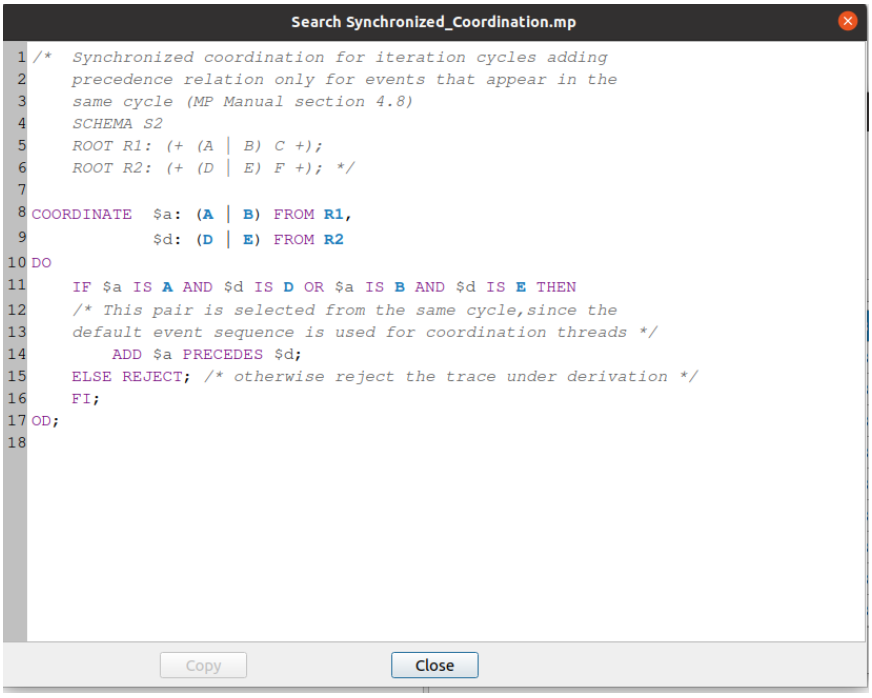

Figure 23. Design of the View Window Displays the Selected .mp File for User to Review Prior to Copying to Text Editor

c. Export from File Menu

To export entire code to a model library, a user selects Save Model from the File menu (Figure 24). When Save Model is selected a dialogue box opens where a user can select the library type and library path to export the MP code. The path is remembered and is recommended when user selects to Save code again.

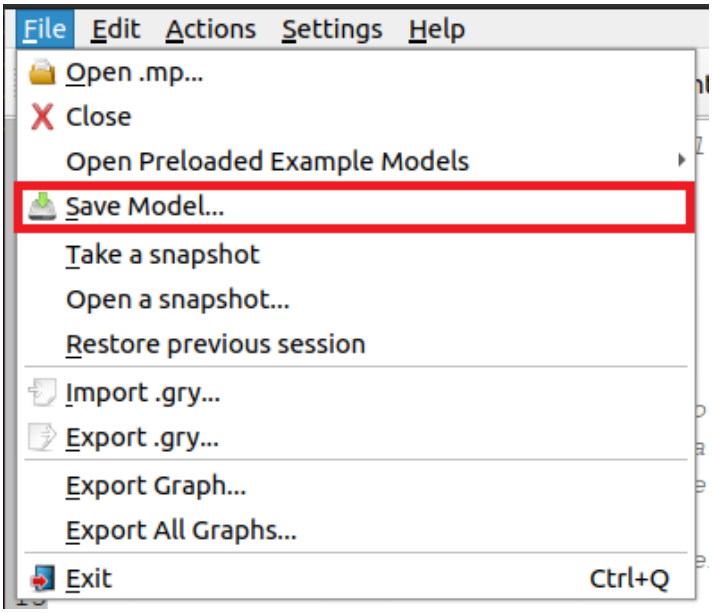

Figure 24. Illustration of Export from File Menu

d. Export Snippets from Text Editor

To save a snippet to a snippet library the user must select the desired excerpt in the text editor, right click, and select Save Selection. A dialogue box will open where the user can select a location as the export destination of the snippet. Figure 25 illustrates the snippet selection from Text Editor and the Save Selection option when right-clicked in the Text Editor.

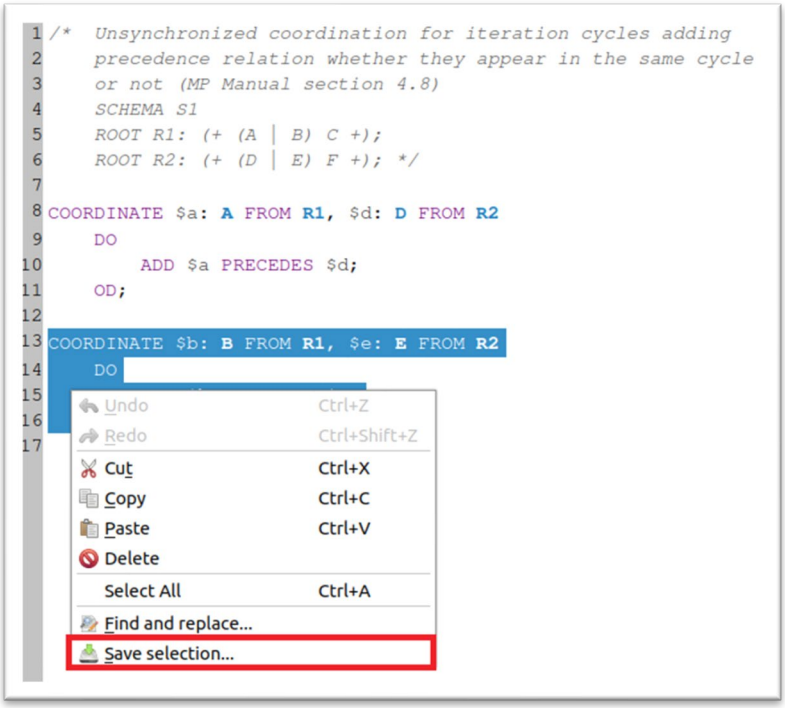

Figure 25. Selecting Excerpt and Right-Clicking to Save Selection as Snippet

3. Searching For Models and Snippets

To aid users to easily find models containing certain text of interest, two free text boxes are embedded within the Search dialogue box which allows a user to search for models or snippets within the selected libraries or all libraries. Expanding on the original concept to search keywords, search capability was refined to find models or snippets by entering filename or text contained within the file. Once search criteria is entered in the Filename Contains and File Contains text boxes, the results are listed in the table within the dialogue box. Furthermore, the user can organize the search results based on Group (library type), Filename, File size, File created, or File Modified in ascending or descending order. Figure 22 shows the search features in red boxes within the Search dialogue box.

C. TEST AND EVALUATION

This section lists the test cases generated to verify the library functions implemented within MP-Gryphon. In addition to test cases, this section also discusses the outcome of the testing done on the new MP-Gryphon GUI.

1. Test Cases

To test the new features implemented within MP-Gryphon GUI to support model and snippet library functions, test cases were generated. Table 3 lists all the test cases generated derived from the requirements to test all the newly added features. In addition to displaying the test cases, Table 3 also displays the expected results for each test case along with a test number associated with each test. Requirement numbers associated with the test case are also listed for traceability and confirmation that each requirement is satisfied.

Table 3. Test Cases for MP-Gryphon Library Design

Requirement No.	Test No.	Test Case	Expected Results
1.1.1	1	Verify user can request to open a MP model code from preloaded model library	Open dialogue box
1.1.2	2	Verify user can request to open a MP model code from preloaded snippet library	Open dialogue box
1.1.3	3	Confirm user can request to save an MP model code to preloaded model library	Open dialogue box
1.1.4	4	Confirm user can request to save an MP snippet code to preloaded snippet library	Open dialogue box
1.1.5	5	Confirm user can request to search models	Open dialogue box
1.1.6	6	Confirm user can request to search snippets	Open dialogue box
1.2.1	7	Confirm user is notified when model code is imported to text editor	Console message
1.2.2	8	Confirm user is notified when a snippet code is imported to text editor	Console message
1.2.3	9	Confirm user is notified when model code is saved to libraries	Console message
1.2.4	10	Confirm user is notified when a snippet code is saved to libraries	Console message
1.2.5	11	Verify results are returned based on search criteria	Search output in dialogue box
1.3.1	12	Ensure model libraries are provided such that users can access them via MP-Gryphon GUI.	Within dialogue box list of the model library types along with option to select them

Requirement No.	Test No.	Test Case	Expected Results
1.3.2	13	Ensure snippet library is provided such that users can access them via MP-Gryphon GUI	Within dialogue box list of the snippet library along with option to select them
1.3.3	14	Ensure a model library is provided to save models to	Option to select model libraries when saving
1.3.4	15	Ensure a model library is provided to open models from	Option to select model libraries when opening
1.3.5	16	Ensure a snippet library is provided to save models to	Option to select snippet libraries when saving
1.3.6	17	Ensure a snippet library is provided to open models from	Option to select snippet libraries when opening
1.3.7	18	Verify model libraries are searchable based on model names.	Enter search criteria by model name and display results
1.3.8	19	Verify model libraries are searchable based on keywords.	Enter search criteria by keyword and display results
1.3.9	20	Verify snippet libraries are searchable based on snippet names.	Enter search criteria by snippet name and display results
1.3.10	21	Verify snippet libraries are searchable based on keywords.	Enter search criteria by keyword and display results
2.1.1	22	Confirm user can add subdirectories within model libraries	Add subdirectories within existing model libraries
2.1.2	23	Confirm user can add subdirectories within snippet libraries	Add subdirectories within existing snippet libraries

2. Test Results

The updated MP-Gryphon GUI was tested based on the test cases in Table 3 to ensure the library function was implemented per the requirements. Table 4 displays the actual test results for each test case along with the expected results from Table 3.

Table 4. Test Cases and Results for MP-Gryphon Library Design

Requirement No.	Test No.	Test Case	Expected Results	Test Results
1.1.1	1	Verify user can request to open a MP model code from preloaded model library	Open dialogue box	Pass
1.1.2	2	Verify user can request to open a MP model code from preloaded snippet library	Open dialogue box	Pass
1.1.3	3	Confirm user can request to save an MP model code to preloaded model library	Open dialogue box	Pass
1.1.4	4	Confirm user can request to save an MP snippet code to preloaded snippet library	Open dialogue box	Pass
1.1.5	5	Confirm user can request to search models	Open dialogue box	Pass
1.1.6	6	Confirm user can request to search snippets	Open dialogue box	Pass
1.2.1	7	Confirm user is notified when model code is imported to text editor	Console message	Pass
1.2.2	8	Confirm user is notified when a snippet code is imported to text editor	Console message	Pass
1.2.3	9	Confirm user is notified when model code is saved to libraries	Console message	Pass
1.2.4	10	Confirm user is notified when a snippet code is saved to libraries	Console message	Pass
1.2.5	11	Verify results are returned based on search criteria	Search output in dialogue box	Pass
1.3.1	12	Ensure model libraries are provided such that users can access them via MP-Gryphon GUI.	Within dialogue box list of the model library types along with option to select them	Pass
1.3.2	13	Ensure snippet library is provided such that users can access them via MP-Gryphon GUI	Within dialogue box list of the snippet library along with option to select them	Pass

Requirement No.	Test No.	Test Case	Expected Results	Test Results
1.3.3	14	Ensure a model library is provided to save models to	Option to select model libraries when saving	Pass
1.3.4	15	Ensure a model library is provided to open models from	Option to select model libraries when opening	Pass
1.3.5	16	Ensure a snippet library is provided to save models to	Option to select snippet libraries when saving	Pass
1.3.6	17	Ensure a snippet library is provided to open models from	Option to select snippet libraries when opening	Pass
1.3.7	18	Verify model libraries are searchable based on model names.	Enter search criteria by model name and display results	Pass
1.3.8	19	Verify model libraries are searchable based on keywords.	Enter search criteria by keyword and display results	Pass
1.3.9	20	Verify snippet libraries are searchable based on snippet names.	Enter search criteria by snippet name and display results	Pass
1.3.10	21	Verify snippet libraries are searchable based on keywords.	Enter search criteria by keyword and display results	Pass
2.1.1	22	Confirm user can add subdirectories within model libraries	Add subdirectories within existing model libraries	Pass
2.1.2	23	Confirm user can add subdirectories within snippet libraries	Add subdirectories within existing snippet libraries	Pass

The test results of the updated MP-Gryphon GUI containing the new library features listed in Table 4 show that all the intended changes have been successfully made to the GUI. The new design of the MP-Gryphon GUI contains the three libraries recommended in this thesis which are Preloaded examples, Preloaded snippets, and Personal collection. The testing also confirms and verifies that access to the libraries is implemented as intended. Testing shows that importing and exporting of models and snippets was successful from the various locations within the GUI such as File menu, Edit

menu, and from Text Editor. The search capability is also implemented successfully within the MP-Gryphon GUI. The search capability can be accessed successfully from the Edit menu where users can search for files based on name and contents of the file. Additionally, the user can also search within selected libraries or all libraries.

D. DESIGN ILLUSTRATION

This section provides a step-by-step sequential illustration of how the new design within MP-Gryphon GUI is utilized. The following table and illustrations demonstrate the steps required to 1) import an entire model from the preloaded example models library, followed by 2) import code snippets within a previously imported example model. Table 5 lists the required steps along with a quick description of each step. Figures 26 to 32 display screen captures of each step of the process.

It should be noted that users can repeat the process of adding snippets to a model multiple times as required. Additionally, the inserted snippets must be edited to match the context of the entire model. For example, if a Coordinate statement is imported, then the events being coordinated must be entered per the declared events in the existing model.

Table 5. Sequential Steps and Descriptions of Process to Import Models and Snippets from Libraries

Process Step No.	Description
1	To import from preloaded examples model library, select Edit, Search .mp files.
2	After the Search .mp files dialogue box opens, select preloaded example models. The results section of the dialogue box will filter and display all files within preloaded examples models library.
3	Select a model (in this case, Autonomous Car.mp) to import.
4	Click View to open a second window displaying the contents of the selected model file.

Process Step No.	Description
5	Review the file contents and Select All.
6	Click Copy, this will copy the selected text to clipboard.
7	In Code Pane, right click and paste the copied content within Text Editor. User reviews the example model and runs the code to view the generated traces. To add additional constraints user can selected from preloaded example snippets library to import snippets easily.
8	To import from preloaded example snippets library, select Edit, Search .mp files.
9	The Search .mp files dialogue box opens; select preloaded example snippets.
10	To further filter the results, enter a known filename in the Filename Contains.
11	The results section of the dialogue box filters and displays all files within the preloaded example snippets library containing the specified filename in the search field. Select the desired snippet file to import.
12	Click View to open a second window displaying the contents of the selected model file.
13	Review the file contents and Select the snippet to import.
14	Click Copy, this will copy the selected text to the clipboard.
15	In the Code Pane, place the cursor at the desired location to place the snippet and right click and paste the copied content within Text Editor. The user reviews and edits the new snippet added to model and runs the code to view the generated traces.

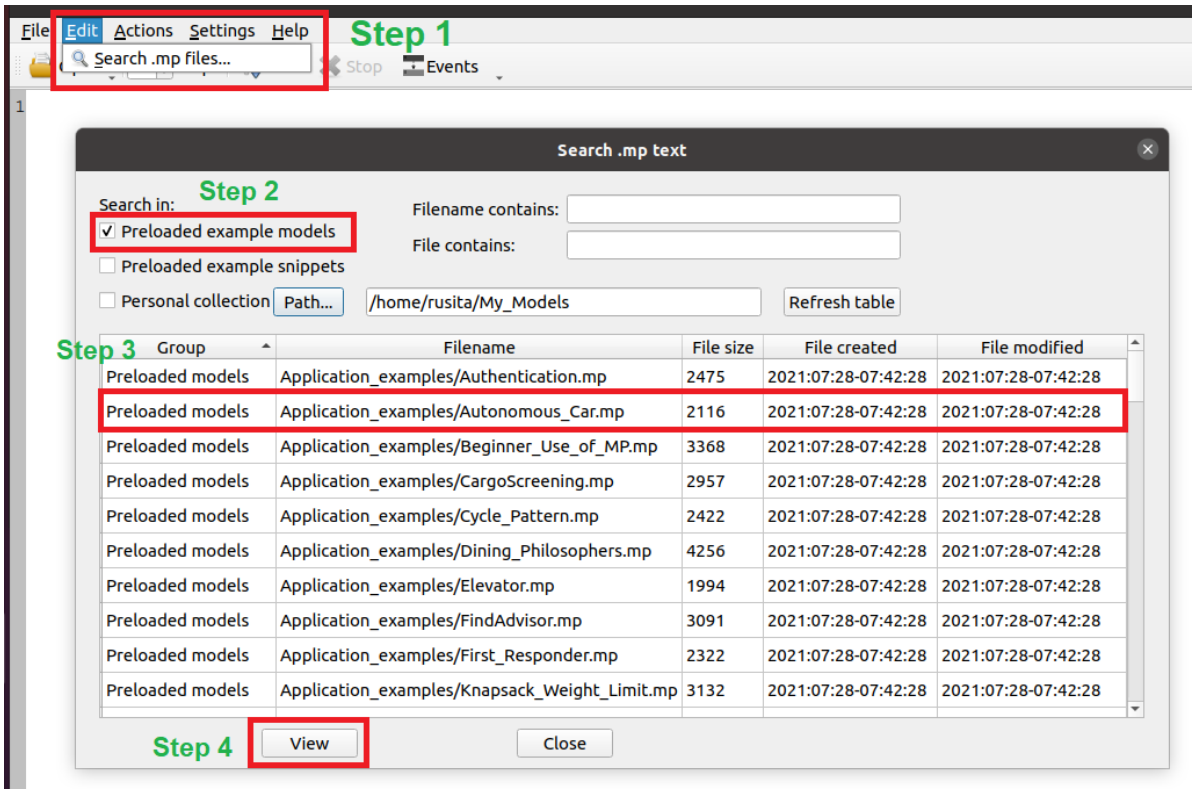

Figure 26. Screen Capture of Steps 1 to 4 of Import from Library Process

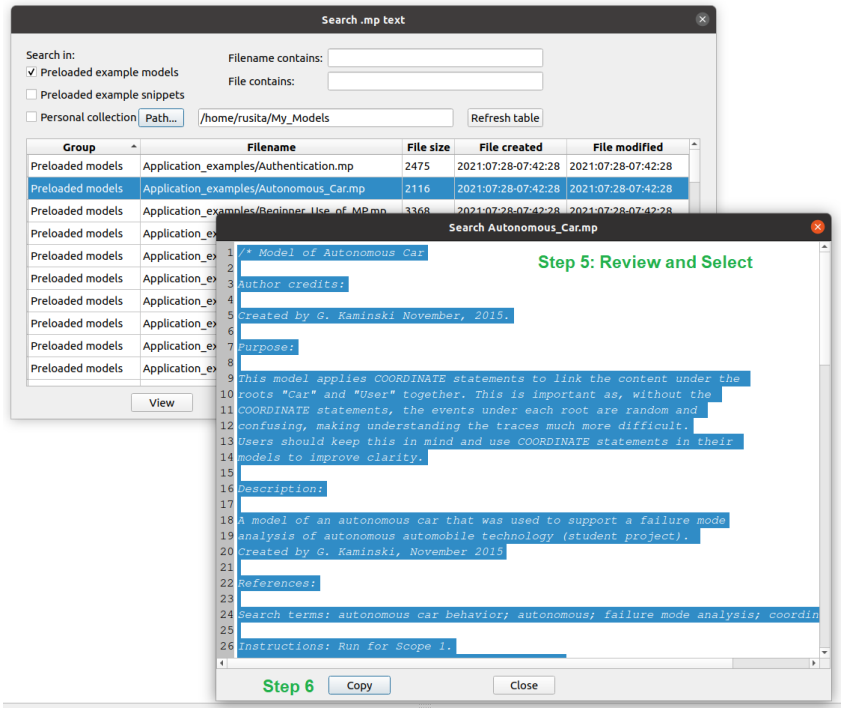

Figure 27. Screen Capture of Steps 5 to 6 of Import from Library Process

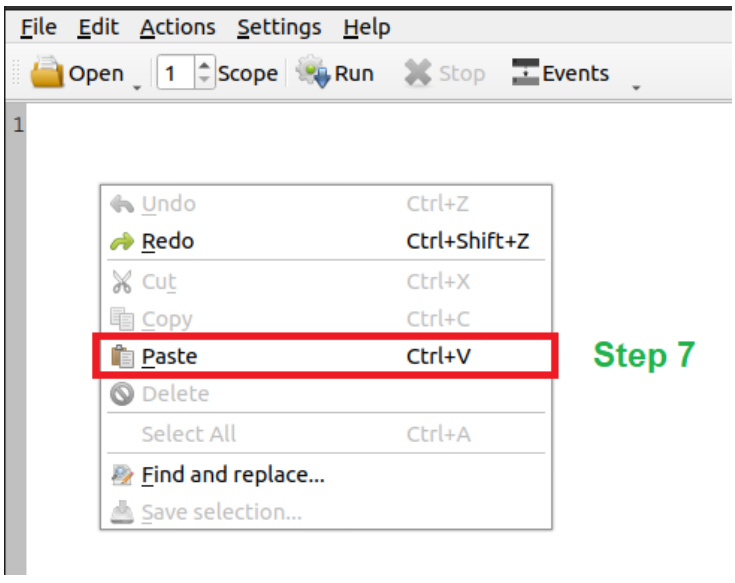

Figure 28. Step 7 illustrating Paste Code within Text Editor

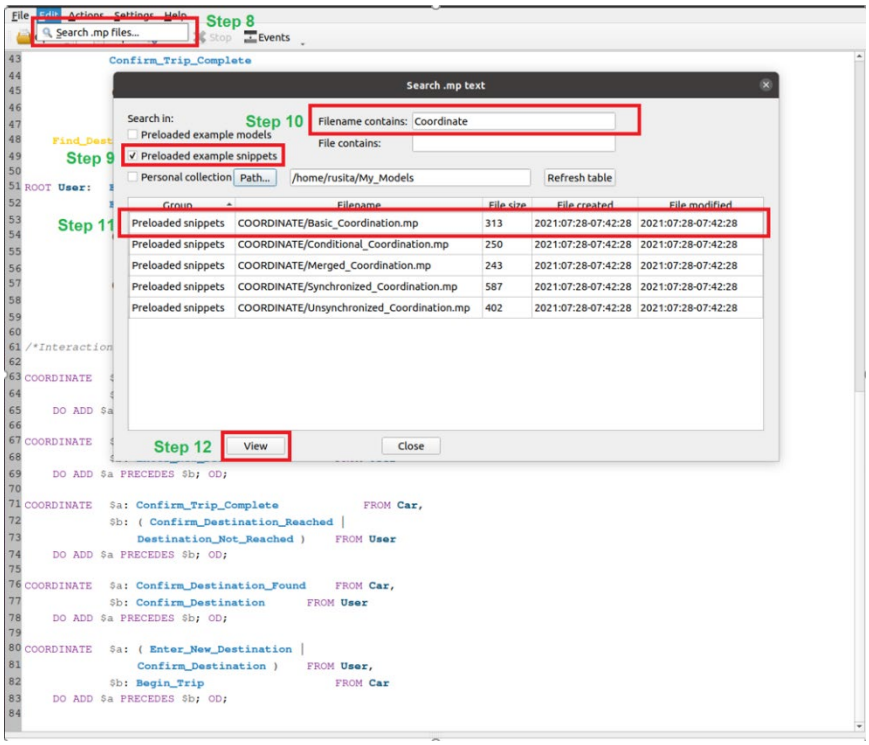

Figure 29. Screen Capture of Step 8 to 12 of Import from Library Process

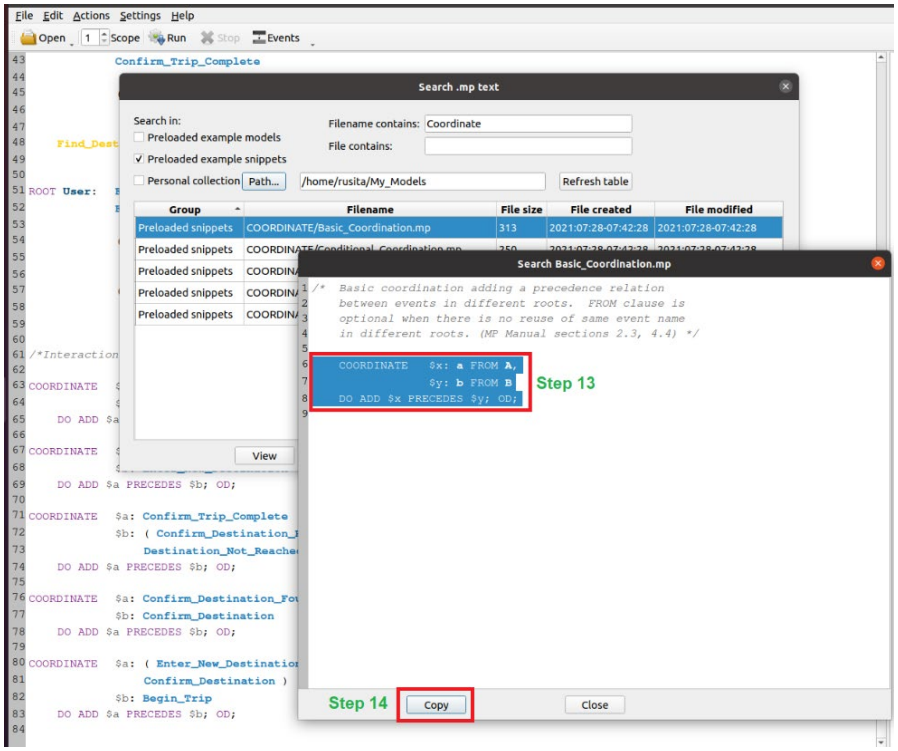

Figure 30. Screen Capture of Step 13 to 14 of Import from Library Process

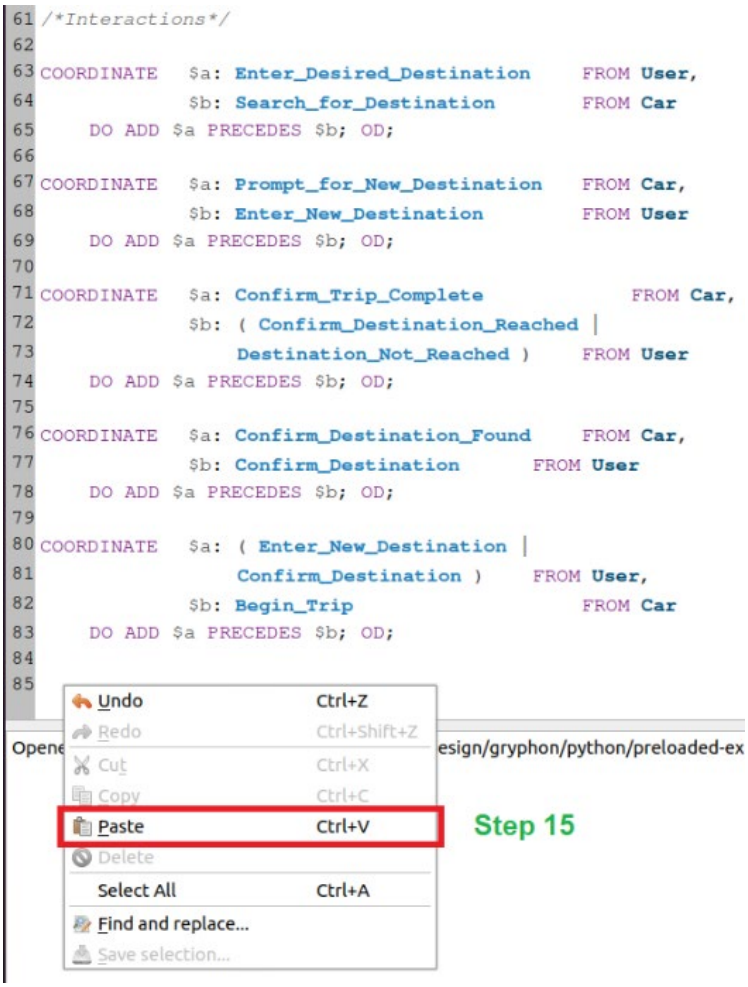

Figure 31. Step 15 illustrating to Paste Code Snippet within Text Editor

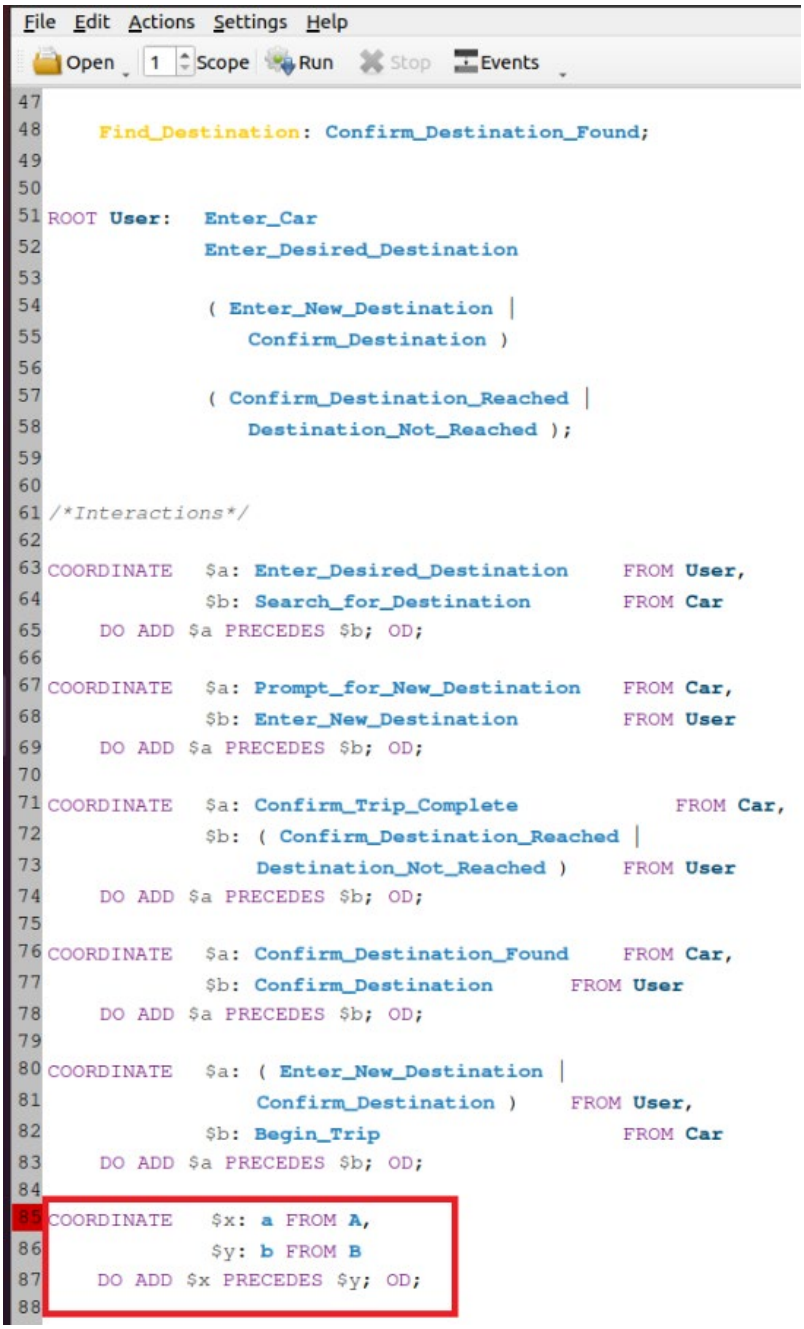

Figure 32. Shows the Imported Snippet within Existing Model

V. CONCLUSION AND FUTURE WORK

This chapter discusses the findings and conclusion of this thesis, following with recommendations on future research within the area of MP model and snippet reuse and libraries.

A. CONCLUSION

The objective of this thesis was to create a process to reuse MP code segments by creating formalized libraries that can be accessed through MP GUI for repurpose. To accomplish the objective of this thesis, a design-oriented systems engineering method was utilized which was divided in three phases. The first phase evaluated two modeling tools, Cadence PSpice used in electrical engineering, and Dassault Systems SolidWorks used in mechanical engineering. The first phase concluded with a comparison of PSpice, SolidWorks, and MP to identify capability gaps in the MP tool. The second phase consisted of MP-Gryphon architecture review, followed by an analysis of user's utilization of the current MP-Gryphon GUI and creation of use cases. The second phase concluded with the generation for requirements for the new design. Phase three of the research assesses multiple concepts for the design followed by a description of the final design implementation. Phase three concluded with testing and evaluation of the new design implementation and an illustration of the library access process.

This research successfully demonstrates a process to access the model and snippet libraries for reuse. New features and capabilities added to the MP-Gryphon GUI to support the objectives of this thesis include:

- Libraries with distinct characteristics to store MP models and snippets.
- Ability for the user to expand model libraries through the availability of a personal collection library.
- Search functionality to find models within libraries and further refining of the search criteria by filtering by filename or text within the file.

- Import/Export capability of models and snippets.
- Ability to review and select model or snippet contents prior to importing.

The updated MP-Gryphon GUI provide users of MP with the ability to store, organize, and manage model and snippet libraries and can enable reuse of entire models or snippets. Being able to leverage previously proven models or portions of the models from one design to another provides users with the ability to easily incorporate used models within new designs with confidence and with minimal effort. Editing and review of the imported model or snippet is required from one use to another for appropriate application, as it may or may not be an exact fit for different applications. However, a library with known models and outcomes provides foundation material and a starting point for building models for similar projects. Additionally, the snippets library further provides support to quickly add frequently used excerpts within larger models. The library feature implementation completed in this thesis is a starting point in this area and provides further opportunities to evolve and optimize this capability.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

This section discusses recommendations for future research studies based on the findings of this thesis. The outcome of this thesis has shown that having manageable model libraries enable users to reuse models. This capability can be further optimized to provide additional benefits to the user and to the outcome of the modeling and simulation results. Four areas were found where further research can be conducted to expand on the model libraries and reuse. The addition of a Behavior Patterns library, further optimization of the model search capability, sharing repositories with other MP users and maintenance of libraries are the topics for further research opportunities discussed in the below subsections.

1. Behavior Patterns Library

This thesis added three basic types of libraries within MP-Gryphon GUI, which can be further expanded to include model types with distinct characteristics. A behavior patterns library consists of models with repeatable patterns of behavior that can be applied from one application to another. For example, one system can have behavior that matches a behavior in

another system in another field. This library holds these behavior patterns that can be reused from one model to another. To capture behavior patterns is a challenge as capturing interoperability is difficult in addition to parametrizing these models to easily reuse these models.

2. Optimizing the Model Search Capability

This thesis added a basic search capability that will search the MP-Gryphon libraries based on names of model or keywords that will be searched within the entire model. As users create more models of varying and sometimes similar subjects an elegant search capability will be required that enables users to find the code function, they are looking for without necessarily knowing what the code looks like. The search capability can be further optimized to filter additional information that will include more detailed search criteria. Adding advanced filter criteria such as lines of code and complexity will aid users in finding more refined data.

3. Sharing Full MP Libraries with Other MP Users

As more and more users begin to utilize MP for their MBSE needs more models will be created. With the localized model libraries as the backbone, the next layer to the model library and reuse concept is to share model libraries between users. The MP-Gryphon GUI will need to be further refined to support share capabilities to other users by creating network of MP users and making libraries available via public domain for sharing. Alternatively, this feature can be considered for implementation to the MP-Firebird tool or as an NPS Gitlab project as the latter is a publicly available option and might provide for an easier integration of library sharing capabilities.

4. Maintenance of libraries

Users develop system models as part of ongoing effort for a project; however, when the system is deployed there is a lack of motivation to maintain models and they become outdated. To reuse models effectively, they must be maintained to the current state of the system. A process to maintain model libraries with the current details of the model can be developed. Having a maintenance process incorporated improves the quality of the models and provides the users with confidence when reusing models.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. EXAMPLE SNIPPET LIBRARY CONTENTS

A. COORDINATE STATEMENTS

1. Basic Coordination

```
/* Coordination adding a precedence relation between events
in different roots. FROM clause is optional when there is no
reuse of same event name in different roots. (MP Manual
sections 2.3, 4.4)
ROOT A: a;
ROOT B: b; */

COORDINATE   $x: a FROM A,
            $y: b FROM B
DO ADD $x PRECEDES $y; OD;
```

2. Merged Coordination

```
/* Merged coordination adding multiple precedence relations
at once (MP Manual section 4.4) */

COORDINATE  $x: a FROM A,
         $y: b FROM B,
         $z: c FROM C
 DO ADD  $x PRECEDES $y,
         $x PRECEDES $z; OD;
```

3. Conditional Coordination

```
/* Conditional coordination adding precedence relation only
if event B is present in a trace (MP Manual section 4.7)
 ROOT R1: (A | B | C);
 ROOT R2: D; */

IF #B > 0 THEN
 COORDINATE $x: B, $y: D
     DO ADD $x PRECEDES $y; OD;
FI;
```

4. Unsynchronized Coordination

```
/* Unsynchronized coordination for iteration cycles adding
precedence relation whether they appear in the same cycle
 or not (MP Manual section 4.8)
 SCHEMA S1
```

```
 ROOT R1: (+ (A | B) C +);
 ROOT R2: (+ (D | E) F +); */

COORDINATE $a: A FROM R1, $d: D FROM R2
 DO
     ADD $a PRECEDES $d;
 OD;

COORDINATE $b: B FROM R1, $e: E FROM R2
 DO
     ADD $b PRECEDES $e;
 OD;
```

5. Synchronized Coordination

```
/* Synchronized coordination for iteration cycles adding
 precedence relation only for events that appear in the
 same cycle (MP Manual section 4.8)
 SCHEMA S2
 ROOT R1: (+ (A | B) C +);
 ROOT R2: (+ (D | E) F +); */

COORDINATE  $a: (A | B) FROM R1,
         $d: (D | E) FROM R2
DO
 IF $a IS A AND $d IS D OR $a IS B AND $d IS E THEN
 /* This pair is selected from the same cycle, since the
 default event sequence is used for coordination threads */
     ADD $a PRECEDES $d;
 ELSE REJECT; /* otherwise reject the trace under derivation
*/
 FI;
OD;
```

B. ENSURE STATEMENTS

1. Basic Inequalities for Event Instances

```
/* Ensure that event A is present in all traces. */

ENSURE #A > 0;

/* Ensure that event A occurs a specific number of times.
(Set to 0 if the event should be absent from all traces. */
```

```
ENSURE #A == 2;

/* Ensure that the number of event A is greater than 1.
Guarantees at least two instances of A. Change 1 to 2
to guarantee at least three instances of A. */

ENSURE #A > 1;

/* Ensure that the number of event A is greater than or
equal to 1. Guarantees at least one instance of A. Change
1 to 2 to guarantee at least two instances of A. */

ENSURE #A >= 1;

/* Ensure that the number of event A is less than 3.
Guarantees at most two instances of A. Change 3 to 2
to guarantee at most one instance of A. */

ENSURE #A < 3;

/* Ensure that the number of event A is less than or
equal to 3. Guarantees at most three instances of A. Change
3 to 2 to guarantee at most two instances of A. */

ENSURE #A <= 3;
```

2. Unidirectional Dependency for Number of Events

```
/* Ensure that if event A appears exactly once, then event B
appears exactly once. Change the numbers as needed. */

ENSURE #A == 1 -> #B == 1;
```

3. Bidirectional Dependency for Number of Events

```
/* Ensure that if event A appears at least three times then
Event B appears exactly once, and if event B appears exactly
once then event A appears at least three times. (Traces with
other combinations
Of event instances are rejected.)  Change the numbers as
needed. */

ENSURE #A >= 3 <-> #B == 1;
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Auguston, Mikhail. 2020. “Monterey Phoenix System and Software Behavior Modeling Language (Version 4.0).” Documentation - Monterey Phoenix - NPS Wiki. March 3, 2020. https://wiki.nps.edu/display/MP/Documentation.

Cadence Design Systems. 2019. “PSpice User Guide.” October 2019. https://resources.pcb.cadence.com/i/1180526-pspice-user-guide/195?

Dassault Systems. 2021a. “Analysis Library Features.” SolidWorks Help. 2021. http://help.solidworks.com/2021/english/SolidWorks/cworks/c_Analysis_Library_Features_SimHelp.htm?id=61ab5be10d264a50a01e1b293f81a2d4#Pg0.

———. 2021b. “Basic Concepts of Analysis.” SolidWorks Help. 2021. http://help.solidworks.com/2021/english/SolidWorks/cworks/c_Basic_Concepts_of_Analysis.htm.

Giammarco, Kristin. 2019a. “MP Tutorial Part 1- Abridged (Slides with Narrative Transcript).” Tutorial, May 2019. https://wiki.nps.edu/display/MP/Tutorials.

———. 2019b. “MP Tutorial Part 2- Abridged (Slides with Narrative Transcript).” Tutorial, May 2019. https://wiki.nps.edu/display/MP/Tutorials.

Javelin A TriMech Company. 2017. “SolidWorks SimulationXpress.” Javelin A TriMech Company. December 10, 2017. https://www.javelin-tech.com/3d/solidworks-simulationxpress/.

Lakshmipathy, Ramesh. 2019. “SolidWorks Tech Tip - Simulation Productivity Tools: LIBRARY FEATURES.” *SolidWorks Tech Blog* (blog). September 17, 2019. https://blogs.solidworks.com/tech/2019/09/simulation-productivity-tools-library-features.html.

Lin, Airs. n.d. “PSpice 01: DC Circuit Analysis.” Air Supply Lab Learning Embedded Design. Accessed June 6, 2021. http://www.airsupplylab.com/lab_electrical-measurements-and-circuits/143-ee2049-pspice-01-dc-circuit-analysis.html#2-draw-the-electronic-circuit.

Marrs, Sean. 2017. “Hawk Ridge Systems.” *SolidWorks: Custom Material Library* (blog). February 8, 2017. https://hawkridgesys.com/blog/solidworks-custom-material-library.

Monterey Phoenix. 2021. Example 28 MP Model of MP Architecture, version 4. Monterey Phoenix. Naval Postgraduate School. https://firebird.nps.edu/.

Office of the Deputy Assistant Secretary of Defense for Systems Engineering. 2018. *Department of Defense Digital Engineering Strategy*. Washington, DC: Office of the Deputy Assistance Secretary of Defense for Systems Engineering. https://fas.org/man/eprint/digeng-2018.pdf.

onemilimeter. 2011. "PSPICE Simulation: Ideal Opamp vs UA741." *EEVblog Electronics Community Forum* (blog), July 7, 2011. https://www.eevblog.com/forum/chat/pspice-simulation-ideal-opamp-vs-ua741/.

OrCAD Cadence PCB Solutions. 2014. "OrCAD Capture Fast, Intuitive PCB Schematic Design Solution." OrCAD Cadence PCB Solutions. 2014. https://www.orcad.com/resources/library/orcad-capture-datasheet.

———. 2018. "OrCAD PSpice Designer Plus Optimize Design Performance, Cost-Effectiveness, and Reliability." OrCAD Cadence PCB Solutions. 2018. https://www.pspice.com/sites/default/files/PSpice%20Designer%20Plus%20Datasheet.pdf.

Schnaars, Toby. 2018. "Using the Design Library and Smart Components." Engineers Rule Technology for Design and Engineering. September 18, 2018. https://www.engineersrule.com/library-features-smart-components/.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California