
Systems Engineering And Assurance Modeling (SEAM)

Vanderbilt University, Nashville, TN, USA.



19 July 2023

Contents

1	Introduction to Model-Based Thinking	4
1.1	Understanding Model-Based Qualitative Systems Modeling	4
1.2	An Example of Quantitative Physical-Law Based Systems Modeling	5
1.3	The First Model-Based System Representation: The Block Diagram	6
1.4	The Second Model-Based System Representation: The Functional Decomposition Diagram	7
2	Systems Engineering And Assurance Modeling (SEAM)	10
2.1	What is Model-Based Mission Assurance and SEAM?	10
2.2	Creating a SEAM Account	10
2.3	Useful Resources	12
2.4	Hosting a Local Version of SEAM	13
2.5	SEAM Disclaimer	13
3	SEAM - Project Management	15
3.1	Projects	15
3.2	Managing Projects	16
3.3	Navigation	18
4	Libraries and Resources	21
4.1	Apps	21
4.2	Definitions	21
4.3	Libraries	23
4.4	R&M Objective Hierarchy	24
4.5	Requirements Models	25
5	Goal Structuring Notation (GSN) Models	26
5.1	Definition of GSN Elements	26
5.2	NASA R&M Objective Hierarchy	28
5.3	Creating a GSN Model	29
5.4	Example GSN Model	31
5.5	GSN Resources	34
6	SYSTEMS MODELING LANGUAGE (SYSML) MODELS	36
6.1	Definition of Boxes	36
6.2	Creating a SysML Model	38
6.3	Example SysML Model	40
6.4	SysML Model Resources	41
7	Functional Decomposition Models	42
7.1	Definition of Boxes	42
7.2	Creating a Functional Decomposition Model	42
7.3	Example Functional Decomposition Models	43
7.4	Functional Decomposition Model Resources	44

8	Linking Models	45
8.1	GSN + SysML	45
8.2	SysML + Functional Decomposition Model	45
8.3	Functional Decomposition + GSN	45
9	SEAM Outputs	46
9.1	Coverage Check	46
9.2	Creating a Coverage Check	48
9.3	Fault Trees	49
9.4	References	51
10	SEAM STANDARD PARTS LIBRARY	52
10.1	Instructions for using the Standard Part Type Library	52

1 Introduction to Model-Based Thinking

1.1 Understanding Model-Based Qualitative Systems Modeling

Most engineers working in the space arena are typically trained in electrical, mechanical, or aerospace engineering. Hence, most of them understand detailed quantitative drawings and quantitative models, such as solving a circuit with Kirchhoff's Laws to find a particular voltage in a circuit or drawing a free body diagram to solve for forces in a static mechanical system. This kind of modeling is valuable for understanding the behavior of a circuit or system in both a physical and mathematical manner. Quantitative simulation tools such as LTSpice or MATLAB can also help evaluate these kinds of models when they get too complicated to solve by hand.

However, when the scale of a system becomes so complex that quantitative tools are cumbersome to use, the tools are less useful for design and/or analysis. Examples include an integrated circuit with so many transistors that it would take weeks for an LTSpice simulation to run. Quantitative tools are also less useful when there are many kinds of physical domains (e.g., mechanical, electronic, optical in the same system) such as for spacecraft or self-driving cars. These tools are also less useful if there a lot of software is managing the system's behavior (for example, millions of lines of code in an airplane). A second practical issue is that during the beginning of the design cycle, many of the physical parameters needed for quantitative modeling are unknown because the design is in flux. In these situations, qualitative system modeling is more useful.

System models tend to be qualitative and logical, rather than physical and numerical, because the computational cost of modeling components numerically increases steeply with the number of components being modeled in the same simulation. In this chapter we use an op amp integrator to demonstrate the differences between quantitative and qualitative models. It is not necessary to follow the math, but it can help provide a more wholistic understanding of the processes. A qualitative representation of a system is the approach taken in the System Engineering Assurance Modeling platform (SEAM), which is discussed in this manual. This chapter is meant to help people trained in quantitative modeling make a transition to qualitative system modeling. We will first build a quantitative model of the integrator circuit, then build a qualitative model of the same circuit using standard system modeling diagrams. The purpose of this chapter is to help transition people with backgrounds in specialized engineering of science to SEAM.

Before getting into the example, we should investigate reasons for using a model-based approach to system models. We have already mentioned two reasons, namely, that modern systems, such as satellites or robots are too complex to simulate the whole system numerically, and that many of the physical parameters needed for quantitative models are unknown early in the design cycle. We can add a third reason, which is that modern engineering is making a transition from "document-based system engineering" (DBSE) to "model-based system engineering" (MBSE). In other words, instead of specifying a system and capturing all the information about the system in documents, design engineers are moving toward capturing all the specifications and information about the system in digital (software) objects, which are stored in a central repository. This helps solve one of the toughest problems of large organizations designing complex systems, which is to make sure every person working on the system is operating from the same specifications and level of information about the system at a given time (often called "versioning"). Prior to MBSE, keeping track of design changes and document versions was a huge problem for large research and development organizations. People in differ-

ent departments, or different physical locations, could be working from different document versions. This would result in mistakes and unnecessary work to correct them. This approach is called “Multiple Sources of Truth (MSOT). Under the MBSE paradigm, new versions are instantly created when a change is made and are available to everyone, and old versions can be stored for reference. This idea of a “single source of truth” (SSOT) available to everyone working on the system is one of the great benefits of an MBSE approach to documenting system designs.

1.2 An Example of Quantitative Physical-Law Based Systems Modeling

A small-scale example to illustrate these points about quantitative and qualitative modeling can be seen in Figure I.1, which shows an ideal inverting amplifier circuit configured so that the output is the integral of the input. We can use equations based on physical laws to relate the variables we see in the circuit. For example, we know that the voltage v_C is related to the current i_2 through the capacitor by an integral expression:

$$v_C(t) = \frac{1}{C} \int_0^t i_2(\tau) d\tau + V_C(0)$$

Likewise, we can invoke the ideal property of the op amp that the feedback in the op amp circuit always adjusts the voltages at the op amp input terminals to be zero, so $v^+ = v^- = 0$. That allows us to write the current i_2 in terms of the input voltage v_1 .

$$i_2(t) = \frac{v_1(t)}{R}$$

We can insert this expression into our original expression and use the fact that $v^- = -v_2$ to write the final expression for the output voltage.

$$v_2(t) = \frac{-1}{RC} \int_0^t v_1(\tau) d\tau + V_C(0)$$

Hence if we can set the initial condition on the capacitor voltage to zero (just put a switch across it), we can obtain an ideal integral relation between the output and the input voltages.

This example is the kind of modeling familiar to most engineers. It begins with a detailed physical description of the system (the circuit schematic, in this case). The interaction of the components is modeled in terms of equations derived from physical laws. Since the model is equation-based, we could also simulate the circuit using a numerical simulator that solves differential equations. This circuit would probably be included in a larger circuit, maybe included in an integrated circuit, which itself would be on a circuit board, which might be part of an assembly of several circuit boards and electromechanical parts, such as motors. Hence this approach is a bottom-up system description.

However, when we consider large engineered systems such as automobiles or spacecraft, at some point the complexity of the system exceeds what can be modeled in this quantitative way. For one thing, even with modern computing power, the system gets too complex to be analyzed or simulated in this quantitative way. In addition, we may not know with precision all the values of all the possible physical variables that are necessary for a detailed quantitative description of a large engineered

system. Lastly, such a complex physical representation is too complicated for one human being to understand and make decisions about the inter-relationships of the subsystems.

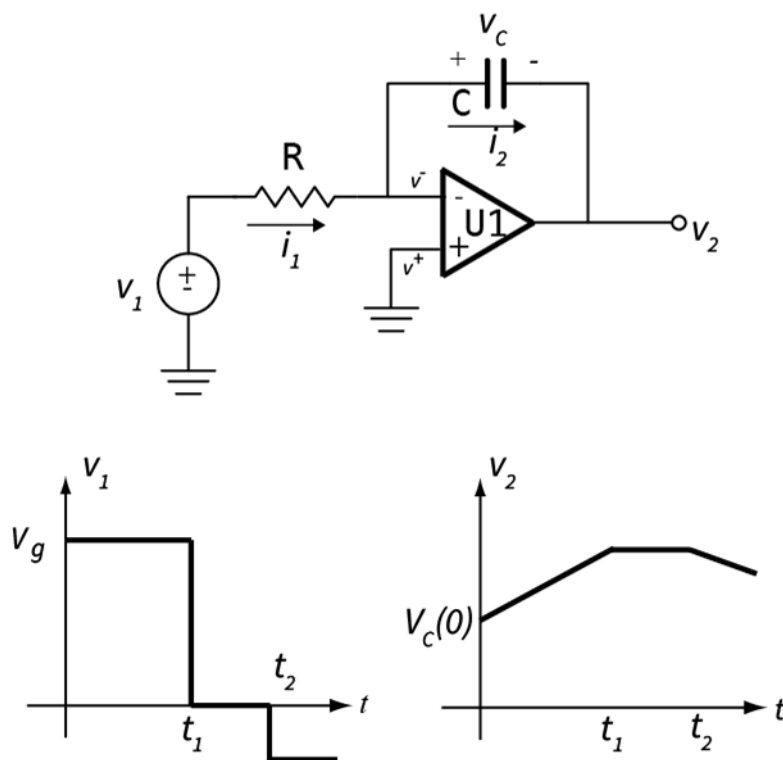


Figure 1.1. An ideal op amp integrator circuit, with the input and output voltage waveforms.

1.3 The First Model-Based System Representation: The Block Diagram

In contrast to physical modeling, model-based thinking is less about mathematical relationships between components and more about the arrangements of components in the system and the functions of components and subsystems, i.e., what they are actually supposed to do. These are sometimes called “logical models,” not because they are based on digital logic, but because they are based on the logical relationships between components and the flow of signals and power through the system. Hence these modeling languages describe the system in terms of various diagrams intended to describe the form, function, and relationships of various parts of the system.

The systems-modeling part of SEAM is based on a language called SysML, which is short for Systems Modeling Language¹. SysML consists of a set of descriptive diagrams that capture various aspects of the system. Let’s look at one of the most basic diagrams of SysML, the block diagram. In the block diagram we are not trying to describe detailed connections and behavior of a sub-system the way the schematic of the integrator circuit does. Rather, we just want to capture the parts of the subsystem and connections between them. Unlike a quantitative circuit schematic, which has only one correct representation, a qualitative model can have multiple forms for the same sub-system, depending on what features the modeler wants to emphasize, and what level of abstraction is needed to represent the aspects of the system the modeler is interested in.

Let’s look at a couple of examples of block diagrams for the integrator. The first is shown in Figure

¹Sanford Friedenthal, Alan Moore, Rick Steiner, “OMG SysML™ Tutorial,” www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf, INCOSE, 2009.

I.2. This block diagram roughly follows the configuration of the circuit schematic. However, this diagram represents the circuit at a higher level of abstraction, which means that it is simpler than the schematic and some of the detailed information is lost. We have lost the definitions of the currents and voltages, as well as the detailed connections between connections. The diagram is no longer “solvable” using equations. All that remains is some information about which components are in the system and some basic idea of how signals flow between them.

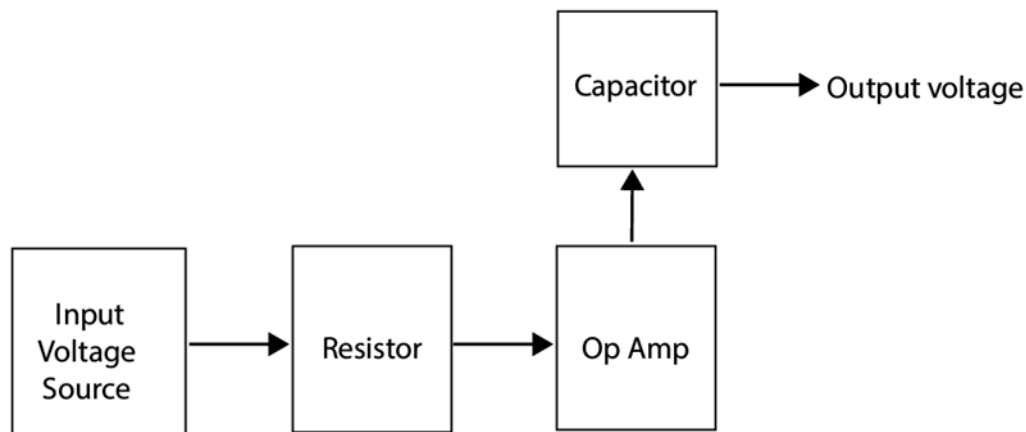


Figure I.2. Block diagram for the op amp integrator.

Now suppose that as a modeler I am interested in the signal processing aspects of the circuit. I could represent the action of the circuit by the block diagram shown in Figure I.3. There is something that creates an input signal, which is not specified, it could be a voltage source, a sensor, or another circuit. There is a block that represents the integrating action of the capacitor, which is not specified. Lastly there is a block I have labeled as “scale factor” which represents the fact that the integral is multiplied by $(RC)^{-1}$, which produces the output signal, which could be voltage, or current, it is not specified in the diagram. Both the block diagrams in Figure I.2 and Figure I.3 are valid block diagrams of the integrator circuit schematic shown in Figure I.1, they are both higher-level abstractions of the integrator that have less information but highlight particular aspects of the integrator. In Figure I.2 the signal flow between actual components is emphasized, in Figure I.3 the signal processing aspects of the circuit are emphasized.

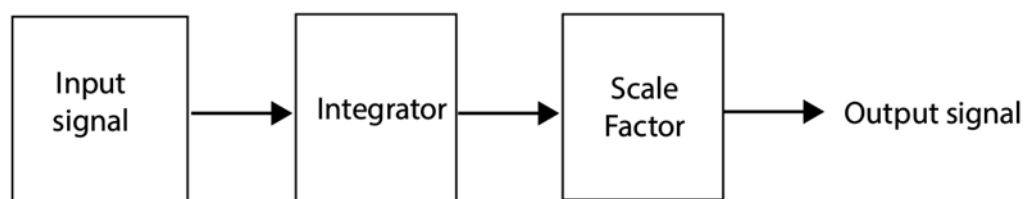


Figure I.3. Signal-processing block diagram for the op amp integrator.

1.4 The Second Model-Based System Representation: The Functional Decomposition Diagram

Now let’s look at another way to represent the system behavior, namely, a diagram that describes the relationships between a function of a sub-system and its constituent components. That is, in this diagram, we assign some “responsibility” for part of the functionality of the subsystem to each of the

parts. This diagram is a key part of being able to discern the impact of a low-level failure on a top-level function.

The top-level function in our example is “Integrate the input voltage.” In Figure I.4 we have redrawn our integrator circuit to show actual components that are associated with the power source, V_{DD} , which is now shown to be a battery labeled V_{B1} , and the input signal, V_1 , which is now shown to be a light sensor based on the phototransistor, Q_1 . We also assigned number values to our previous components R and C to show that in our circuit board they are specific instances of particular resistor and capacitors. In other words, R is the general symbol for a resistor, but R_1 represents a particular physical resistor of a certain value and part number from a particular manufacturer; it is the real resistor in the physical world. For example, maybe R_1 is 2 k Ω , but in the next stage of the circuit there is another 2 k Ω resistor associated with an analog-to-digital converter. Both resistors fit into the general class of 2 k Ω resistors, but physically they are different resistors, so we give them different names; let’s call the second resistor R_4 . So, the physical resistor R_1 is associated with the function of “Integrate the signal from the sensor,” but R_4 is associated with the function “Convert the analog amplifier voltage V_2 to a digital code.” Now we are in a position to draw a functional decomposition diagram (F_{DD}), which assigns responsibility for functions to particular instances of components.

Now that we see the whole picture, we know that what the circuit is really doing is averaging the light flux illuminating the phototransistor, so we can call the system function: “Integrate the Light Flux,” or “Average the Light Flux.” Figure I.5 shows an FDD for the integrator circuit of Figure I.4 for this system function. The top-level system function is illustrated with an upper-case $F(x)$ symbol. The system function is divided into sub-functions (labeled with a lower-case $f(x)$) which are all necessary for the system function to be operational and within specification. The three sub-functions in this case are: (1) provide power, (2) sense the light, and (3) integrate (or average) the sensor signal. Each sub-function is then associated with the components “responsible” for performing the sub-function. All components associated with a sub-function must work correctly for the sub-function to be correct, unless there is redundancy built into the system (for example, two capacitors in parallel in the integrator). Now if there is a degradation or a failure in a component, we can see from the F_{DD} how that failure is going to propagate up to the top-level and impact the system function. For example, let’s say that the “gain,” that is, the proportionality constant between the light input and the current in the collector of the transistor, is changed because of ionizing radiation. We can see this degradation will affect both the sub-function “sense the light” and the system function “Integrate the light flux,” because the averaged light output will differ from the actual physical value because the calibration

is off.

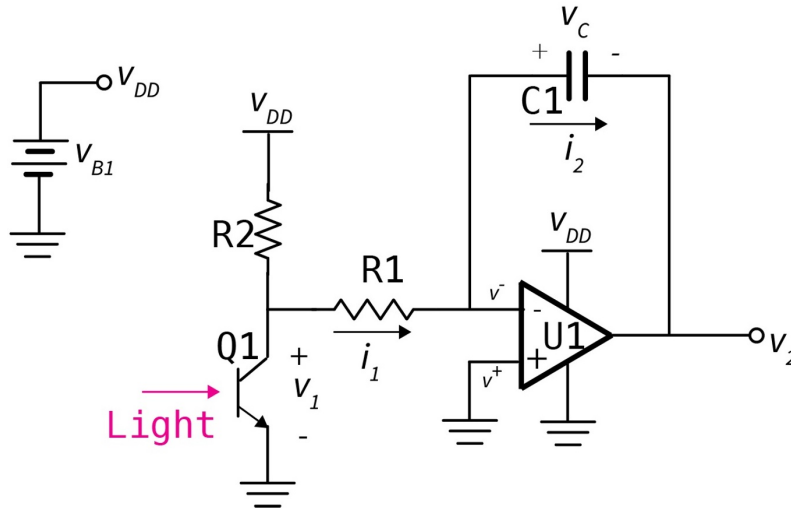


Figure I.4. Integrator showing the components associated with power and signals..

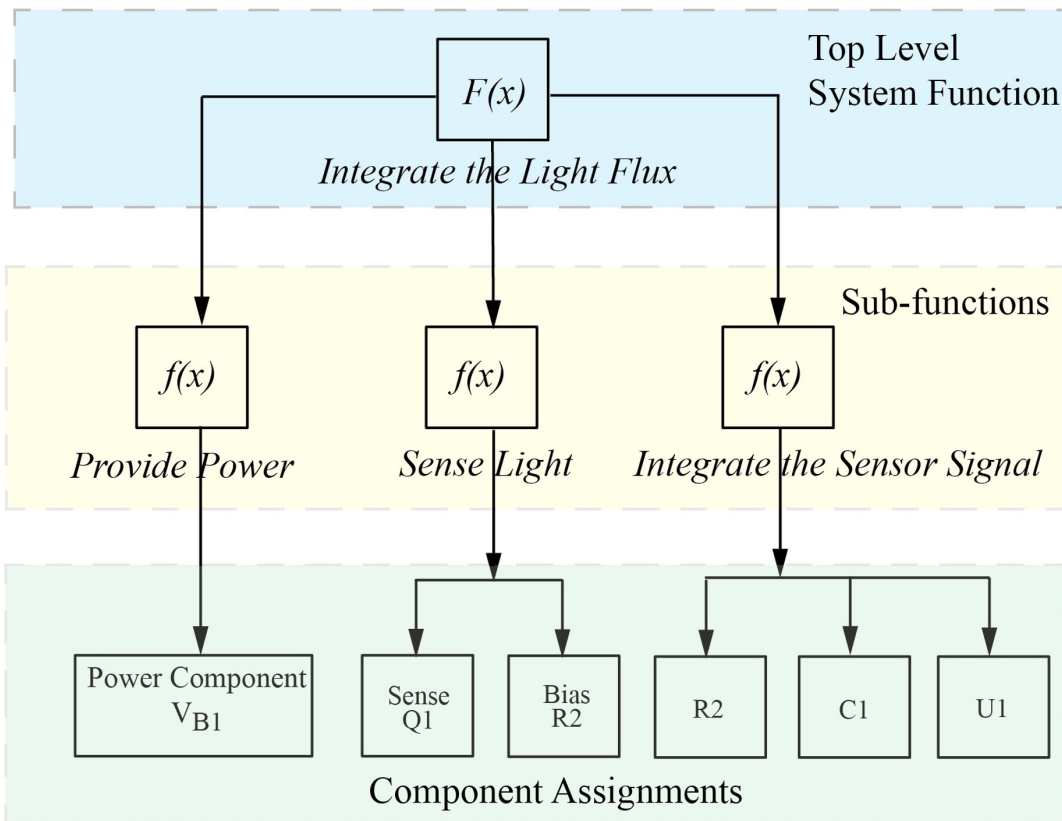


Figure I.5. Functional Decomposition Diagram for the function “Integrate the Light Flux.”

2 Systems Engineering And Assurance Modeling (SEAM)

2.1 What is Model-Based Mission Assurance and SEAM?

Model-based mission assurance (MBMA) is an alternative to document-based mission assurance for establishing the risk and reliability of systems. It leverages block diagrams and fault trees that allow for easy visualization of requirements and fault propagation.

SEAM (Systems Engineering and Assurance Modeling) is a web-based collaborative modeling platform for modeling radiation assurance cases integrated with the models of the system. It is based on NASA's Reliability and Maintainability Standard. In SEAM, project teams can create multiple types of linked models for their systems that can be easily accessed by the entire project team. Some of the types of models that can be created in SEAM are Goal Structuring Notation (GSN) models, System block models (SysML), fault propagation models, and Bayesian nets. These models can be used independently or linked, and all members of a project can have access to them.

2.2 Creating a SEAM Account

The screenshot shows the SEAM website's home page. At the top, there is a navigation bar with links for 'Try Now', 'Documents', 'Tutorials', and 'Contact'. On the right side of the navigation bar are 'Register' and 'Log In' buttons. The main header area features the SEAM logo and a central text block stating: 'SEAM supports system architecture design models that are composed from interconnected block diagram models of individual subsystems.' Below this text is a 'Try it now!' button. The page is divided into six columns, each with a title and a brief description of a SEAM feature:

- GSN Assurance Models:** SEAM supports the Goal Structuring Notations (GSN) standard to build assurance case models. SEAM uses hierarchical models, as well as cross-referencing to manage complexity in GSN models. Additionally, SEAM allows linking assurance cases to system models to provide context to the assurance case argument.
- System Models:** SEAM supports a subset of block diagram models in the SysML modeling standard. These include functional (hierarchical requirement) models and architecture design with block diagram models.
- Fault Models:** SEAM extends the internal block diagram models to allow specification of discrete fault propagation to capture the faults and their anomalous effects within a block (subsystem) and their propagation across the system through subsystem interfaces.
- Integrated Models:** SEAM allows context specification through cross-referencing of modeling entities across the models. Functional models are cross-referenced in the system fault propagation models to capture the impact (function loss or degradation) of and response (mitigation function) to failure effects. Sub-system models that implement specific functions are cross-referenced in functional models. Subsystem and functional models are cross-referenced in the GSN assurance case models to provide context to the
- NASA R&M Hierarchy:** NASA's Reliability and Maintainability Standard serves as a template to build radiation hardness assurance cases for using COTS systems in space missions. SEAM provides template models of the R&M hierarchy to kick-start the assurance case development.
- Collaborate:** Collaborate with your colleagues by simultaneously working on the same project. SEAM uses the WebGME modeling framework that works just like Google Docs. It updates and shows all changes to each user concurrently. And you never lose work because the models are stored in a database in the cloud.

At the bottom of the 'NASA R&M Hierarchy' section, there is an 'Examples' section stating: 'A set of examples is available including: Tutorial Project'.

Figure II.1. SEAM welcome page.

Figure II.1 above shows the SEAM welcome page, which is available at <https://modelbasedassurance.org/>. The “Log In” and “Register” links can be found in the upper right corner of the welcome page. Clicking on the “Register” link will bring up the page shown in Figure II.2, below. After entering in the requested information, reading and acknowledging the Disclaimer, and hitting “Request Account,” an account will be created. The Disclaimer is reprinted at the end of this section. The user will be notified via the email used to register when the account is ready, which can take up to one (1) week.

If you would like to run your own copy of SEAM locally, please see Section E of this chapter.

Authentication

Register a new membership

👤	Name
☀️	User ID
🔒	Password 🔑
↔️	Confirm password 🔑
✉️	Email
🏢	Organization Name
📍	Organization Address
🌐	Organization Country

I have read the disclaimer

[Request Account](#)

[I already have an account](#)

Disclaimer

The SEAM tool set and the associated models have been prepared for the Radiation Effects research community for informational purposes that are not export controlled. Your privacy and security are important to us; please **do not** upload any data that is **controlled unclassified information, export controlled, or considered to be intellectual property.**

Please provide a professional email address that we will use only to confirm your registration. Your registration information may be reviewed by our sponsor whose approval is necessary to grant access.

Note: The content of this website and the SEAM tool set are considered "Beta Software".

Vanderbilt University disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall Vanderbilt University be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

Figure II.2. SEAM new membership registration page.

After an account is created, clicking the “Log In” button on the welcome page will bring up the log in screen shown in Figure II.3. After entering the username and password used during registration, the user will be taken to their Projects page. More on the Projects page and project management will be

covered in Section III – Project Management.

Figure II.3. SEAM user login page.

2.3 Useful Resources

This subsection contains links to useful resources for learning about SEAM and MBMA, examples of SEAM+MBMA in use, and resources that are useful for radiation effects analysis. It is not the purpose of this User Manual to describe how to use any of these resources and it is assumed that the user knows how to use any external resources that are necessary for their projects. The most useful resources for new users would be the first six (6) papers listed under the “Papers on SEAM and MBMA” bullet point.

- NASA – <https://www.nasa.gov/>
- ISDE – <http://www.isde.vanderbilt.edu/>
- ISIS – <https://www.isis.vanderbilt.edu/>
- WebGME – <https://webgme.org/>
- NASA R&M Hierarchy – <https://standards.nasa.gov/standard/nasa/nasa-std-87291>
- RGentic – <https://vanguard.isde.vanderbilt.edu/RGentic/>

- CRÈME – <https://creme.isde.vanderbilt.edu/>
- Spenvis – <https://www.spenvis.oma.be/>
- SCRAM – https://scram-pra.org/doc/opsa_support.html
- Papers on SEAM and MBMA:
 - PowerPoint on Reliability & Maintainability Hierarchy – <https://sma.nasa.gov/docs/default-source/News-Documents/r-amp-m-hierarchy.pdf>
 - Goal Structuring Notation Standard – http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf
 - Master’s Thesis on MBAC+ and SEAM – <https://etd.library.vanderbilt.edu//available/etd-06302016-120807/unrestricted/austin.pdf>³
 - Goal Structing Notation in a Radiation Hardening Assurance Case for COTS-Based Spacecraft – https://modelbasedassurance.org/documents/GSN_GOMAC.pdf
 - A CubeSat-Payload Radiation-Reliability Assurance Case using Goal Structuring Notation – https://modelbasedassurance.org/documents/GSN_RAMs.pdf
 - Towards a Framework for Reliability and Safety Analysis of Complex Space Missions – https://modelbasedassurance.org/documents/MBAC_AIAA.pdf
 - Reliability Assurance of CubeSats using Bayesian Nets and Radiation-Induced Fault Propagation Models – https://modelbasedassurance.org/documents/GSN_NEPPETW.pdf

2.4 Hosting a Local Version of SEAM

Clearly the AWS public version of SEAM is meant to be used with non-proprietary information. If you or your organization would like to host a free version of SEAM on your own server behind your own firewall, simply contact the SEAM administrator and we will arrange to send you a Docker Container (www.docker.com), that will easily run SEAM on your server so that you can use it to model your own proprietary or restricted information systems.

2.5 SEAM Disclaimer

The SEAM tool set and the associated models have been prepared for the Radiation Effects research community for informational purposes that are not export controlled. Your privacy and security are important to us; please **[do not]** upload any data that is **[controlled unclassified information, export controlled, or considered to be intellectual property.]**

Please provide a professional email address that we will use only to confirm your registration. Your registration information may be reviewed by our sponsor whose approval is necessary to grant access.

Note: The content of this website and the SEAM tool set are considered "Beta Software".

³<https://etd.library.vanderbilt.edu//available/etd-06302016-120807/unrestricted/austin.pdf>

Vanderbilt University disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall Vanderbilt University be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

3 SEAM - Project Management

3.1 Projects

After you have logged in to your account you will see the Projects page, shown in Figure III.1 below. This page shows all the projects that a user has access to, either as an owner, editor, or as read-only. From here, projects can be opened or created. An existing project can be opened by clicking on the project's name or a new project can be created by clicking on "Create new...", shown in the blue and red boxes in Figure III.1. Opening an existing project will take the user to that project's homepage, an example of which is shown in Figure III.2.

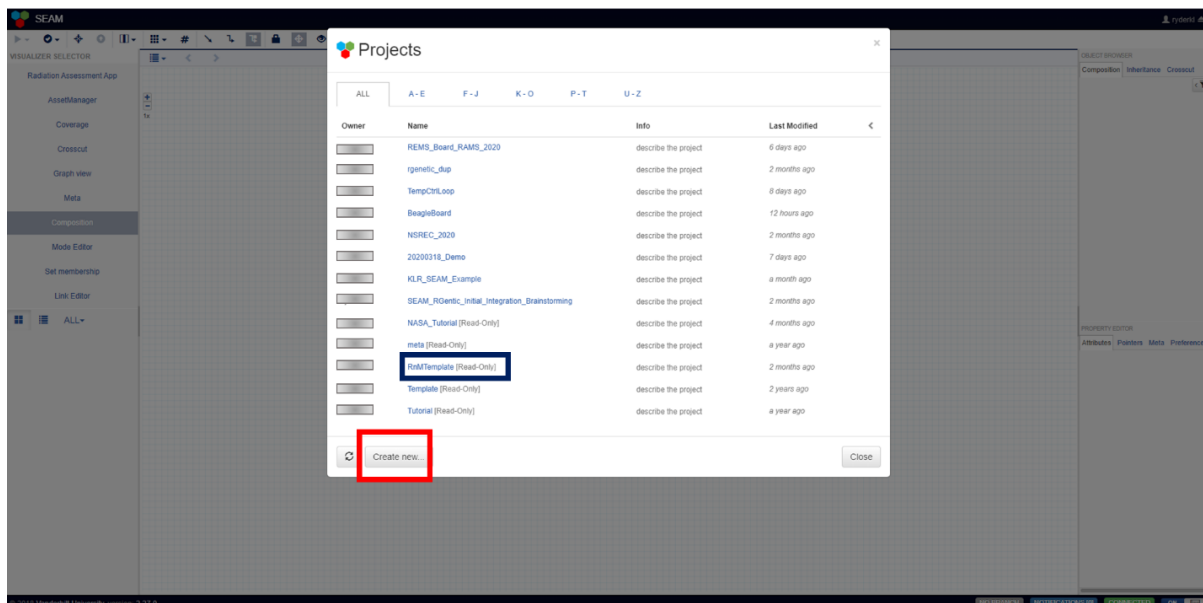


Figure III.1 Projects page as seen after logging in to SEAM. The blue box shows an example existing project that can be opened. The red box shows the "Create new..." button.

Before a new project is created, users must input a name for the project in the text box that appears next to the "Create new" button. There are three options for creating a new project: (1) creating a project from a seed project, (2) creating a duplicate of an existing project, or (3) importing a project from a file. These options appear in Figure III.3. Creating a project from an existing seed project creates a new project from an existing snapshot. It can either be a template file-seed available on the server or a branch from one of the existing projects. The new project will have a single commit⁴ and a branch (master) pointing to the commit⁵. Blank projects can be created in this way. Duplicating a project will make a full copy of an existing project with its entire history. This includes all commits, branches, and data-objects. Finally, importing a project from a file creates a new project from the uploaded snapshot. The new project will have a single commit and a branch (master) pointing to the commit⁶. Once the new project has been created users will be taken to the project home page, which

⁴./definitions.md#Commit

⁵./definitions.md#Commit

⁶./definitions.md#Commit

looks like that shown in Figure III.2.

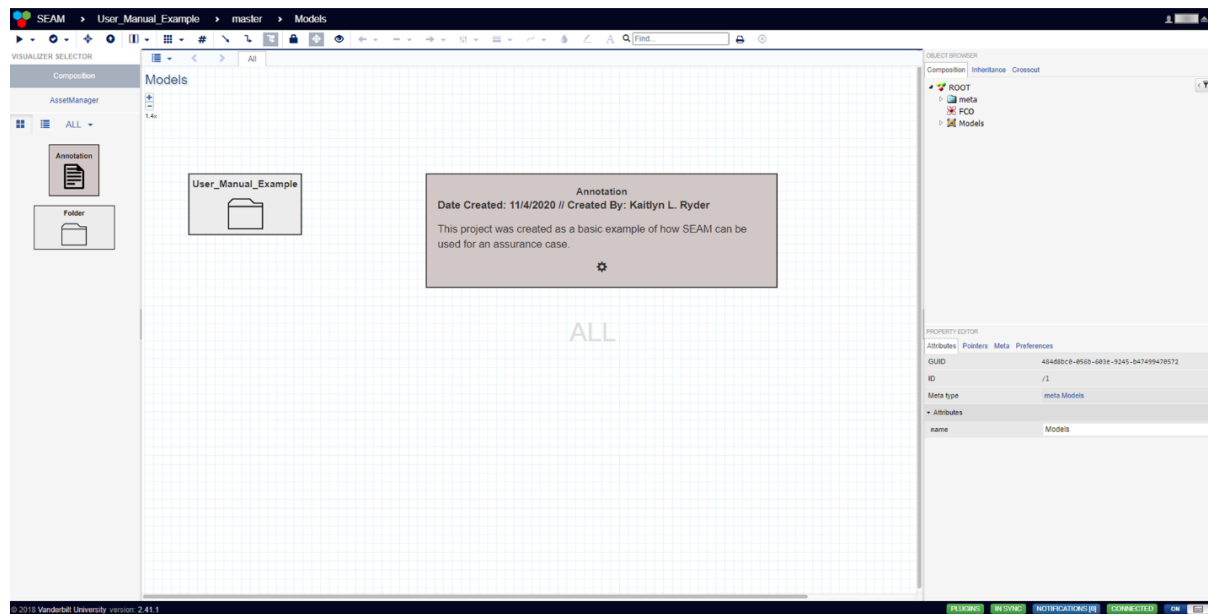


Figure III. 2. Project homepage. Navigation tools are on the right, modules that can be added are on the left. The “User_Manual_Example” folder contains all the models used in this project. Annotations can be added to any page to provide context.

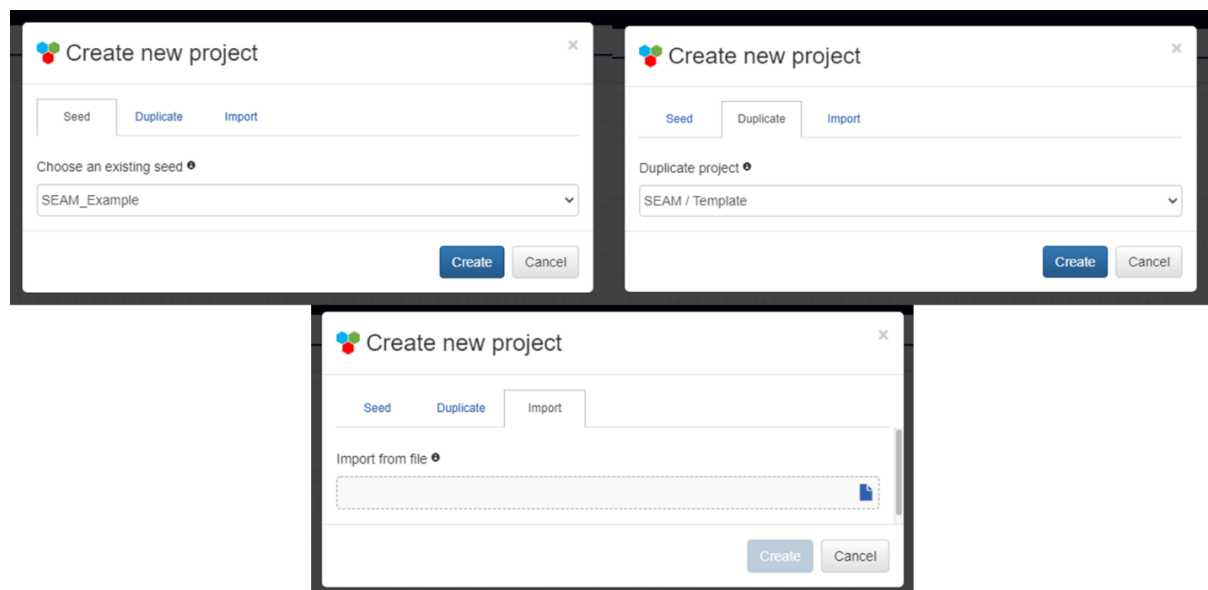


Figure III. 3. Project creation options. (Top left) Create new project using an existing seed, chosen from the dropdown menu. (Top right) Create new project by duplicating an existing project, also chosen from a dropdown menu. (Bottom) Create new project by importing a file.

3.2 Managing Projects

On the top-right of all SEAM pages is the user’s username. Clicking on the username will bring up the GMEProfile page, shown in Figure III.4. From this page, users can manage their profile and projects, see which organizations they belong to, and see a list of SEAM users. On the Projects page, users will see a list of projects they have access to, either as a user or as the owner. An example Projects page

is shown in Figure III.5.

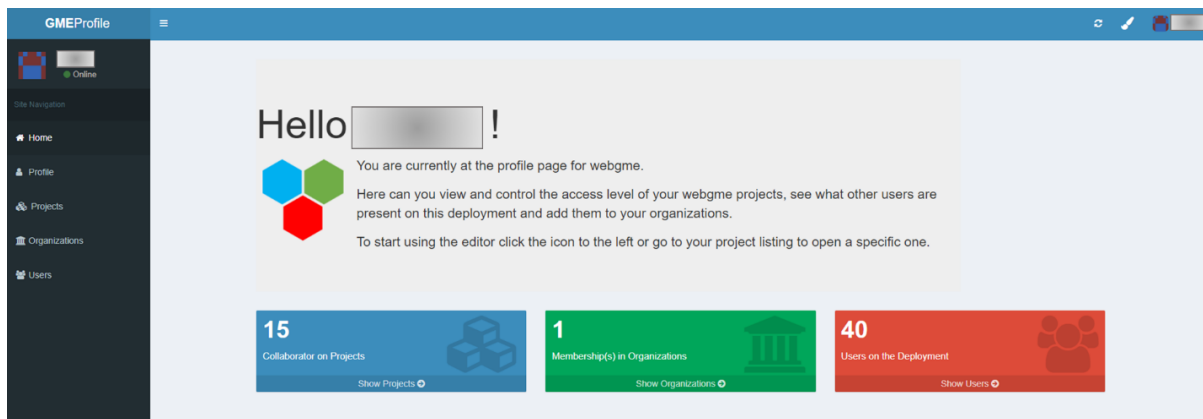


Figure III. 4. GMEProfile page. Dashboard allowing users to manage their profile and projects, see which organizations they belong to, and see a list of other SEAM users.

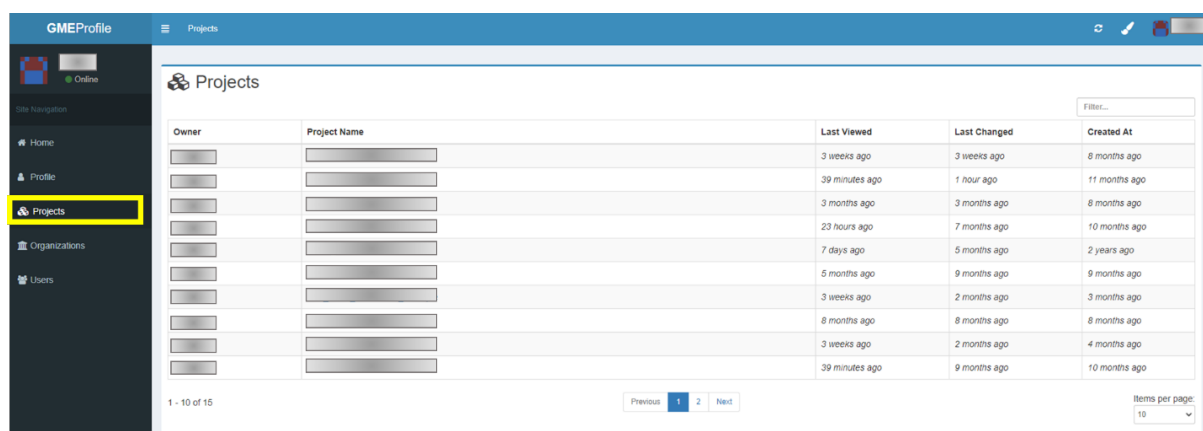


Figure III. 5. Projects page. Shows all the projects a user has access to and who the owner of the project is. Clicking on a project's name will take you to the project's information page.

A project's information page can be accessed by click on its name in the list of projects. Figure III.6 shows the project information page for the KLR_SEAM_Example project that has been used for previous figures. A list of users with access to the project and their permissions is shown on the left. Adding collaborators is on the right (green box) and is done by searching for the user's SEAM username. Permissions are set by clicking on the "R" (for read), "W" (for read and write), and "D" (for read, write, and delete). The box beneath Add Collaborators shows statistics for the project, including when it was last modified, when it was last viewed, when it was created, and a breakdown of commits by user. On the bottom left of the page, shown in the red box, is the "Delete Project" button which permanently

deletes the project.

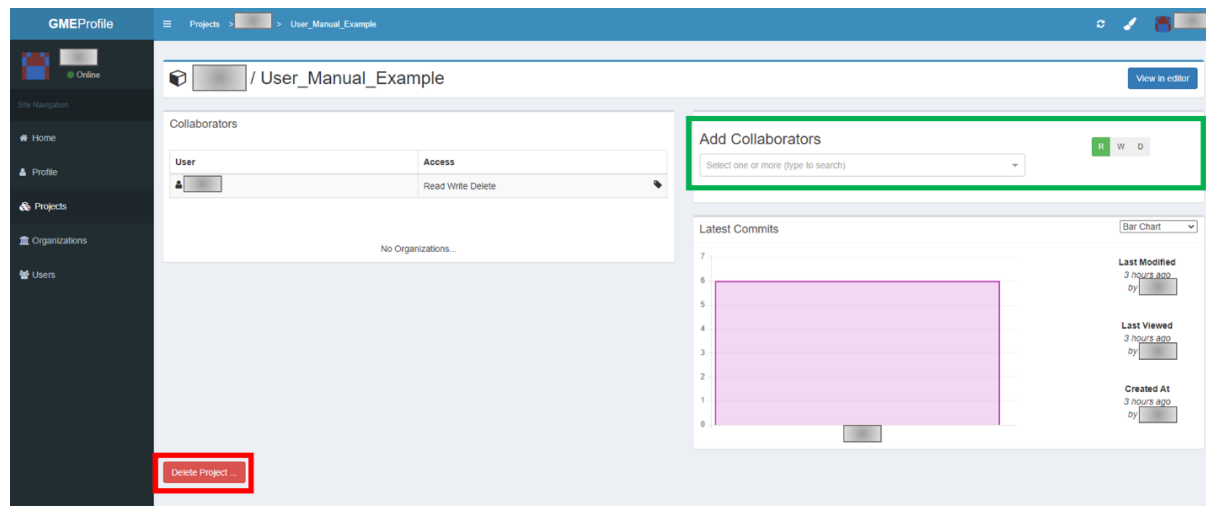


Figure III. 6. Project Information Page. Provides information about a specific project including list of approved users and a breakdown of commits by user. Approved collaborators can be added, user permissions can be changed, and the project can be permanently deleted from this page.

3.3 Navigation

From the project homepage, shown in Figure III.2, there are multiple ways to navigate through a project. Double-clicking on the top-level folder, called `User_Manual_Example` here, will take the user to the main-level folder, shown in Figure III.7. The main-level folder contains all the libraries and references for the project (shown in the gold box in Figure III.7), and all the models used in the project (shown in the green box in Figure III.7). Libraries, references, and models can be added by dragging the appropriate block from the column on the left (shown in the blue box in Figure III.7). Double-clicking on any of the folders will take the user into that folder. Alternatively, users can use the drop-down navigation bar on the right (shown in red in Figure III.7) to navigate into any subfolder. There is an arrow located in the upper left corner (just below and to the right of the yellow box in Figure III.8) that will take users up one level in the project. Finally, by clicking on the SEAM logo in the upper left (shown in the yellow box in Figure III.8) users can switch to a different project, create a new project,

or import an existing project.

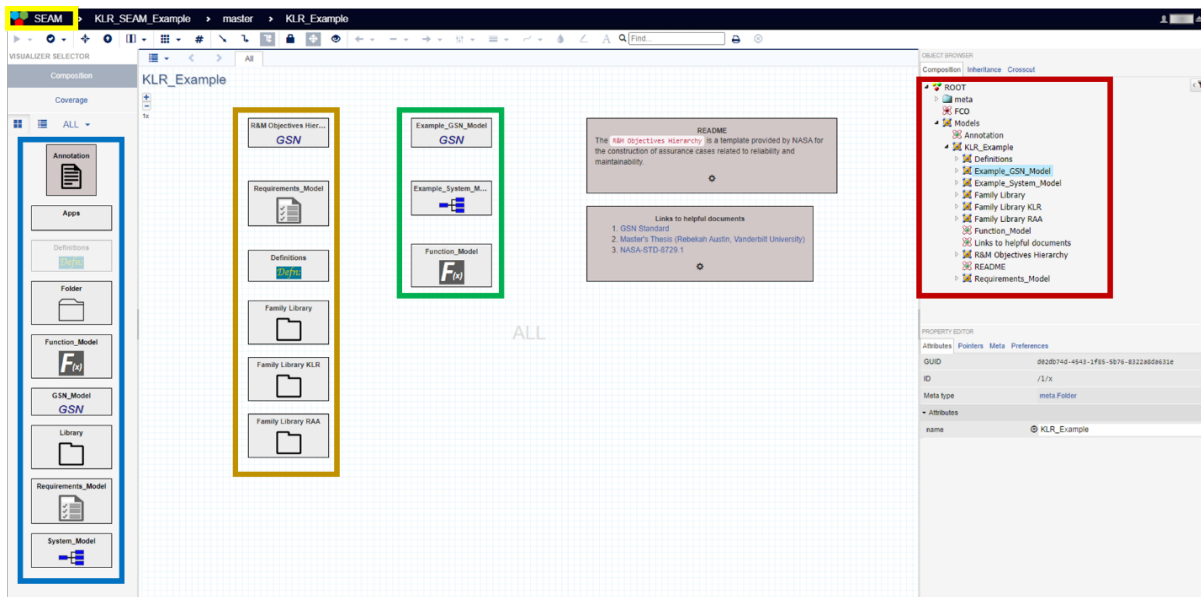


Figure III. 7. Project Main-Level. Libraries, resources, and models can be added by dragging the appropriate box from toolbar on the left (blue box). Libraries, resources, and models can be accessed by double-clicking the appropriate folder (gold and green boxes). Alternatively, sub-folders can be accessed by using the navigation plane on the right (red box). Users can switch to different projects, create new projects, or import current projects by clicking on the SEAM logo in the upper left corner (yellow box)

Another useful navigation feature in SEAM is the ability to split the screen, allowing for two different pages to be active at the same time. Figure III.8 shows an example of split screen, with a functional model being open on the left and a SysML model open on the right. Split screen can be activated by clicking on the split screen symbol in the upper left toolbar (blue box in the figure). Split screen can also be deactivated by clicking on the symbol again and choosing the appropriate option. With split screen open, users can navigate through different areas of the project simultaneously. Clicking on either side of the screen will cause that screen to be “active,” allowing for navigation in the screen. Items can be copied from one area of the project to another by dragging the desired item across the split to the other screen. This is particularly useful when utilizing libraries, which are discussed in more detail in Chapter IV – Libraries and Resources.

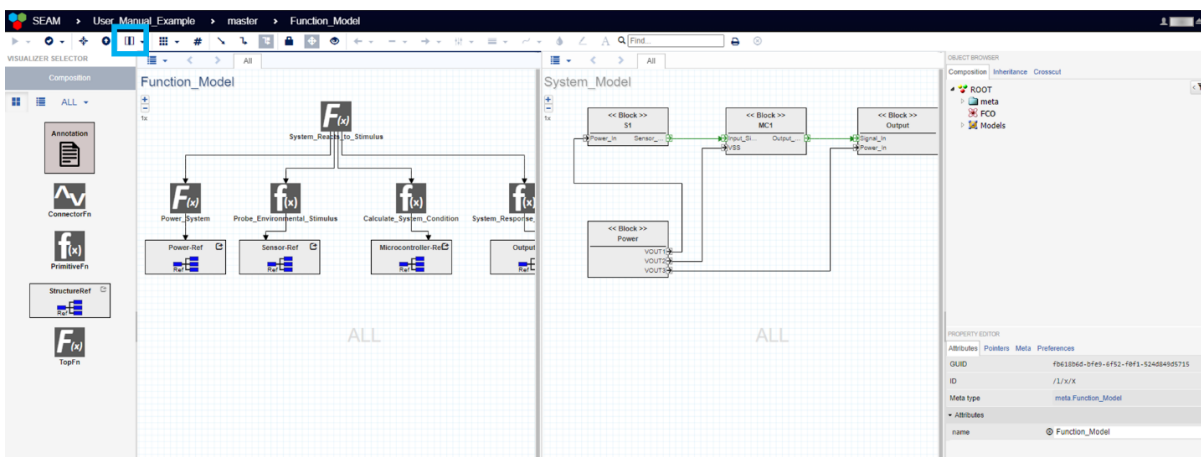


Figure III. 8. Split Screen View. Split screen can be activated/deactivated by clicking on the split screen icon in the upper left toolbar (blue box). Split screens also for simultaneous navigation

of different areas of the project.

4 Libraries and Resources

SEAM provides access to several internal libraries and resources that projects can make use of, shown in Figure IV.1. The available resources include embedded applications, the NASA R&M Objective Hierarchy in GSN format, and example Requirement Models. There are two types of libraries available in SEAM: the Definitions folder containing failure labels for SysML models and tags for GSN models; and Libraries folder that contain model SysML components and subsystems. Each of these libraries and resources will be talked about individually in this chapter.

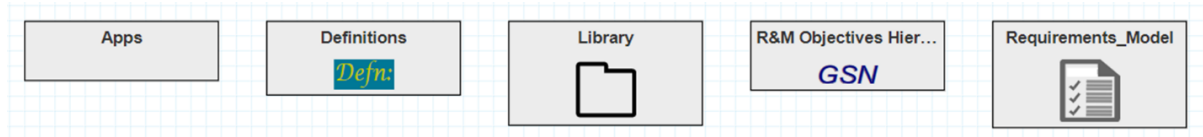


Figure IV.1 All the libraries and resources currently available in SEAM.

4.1 Apps

SEAM can run web-based applications locally, through the Apps folder. Figure IV.2 shows the Apps page and available applications. Applications are opened locally in the SEAM project and run like normal, including any login information. There are currently only two apps available, but more will become available in the future.

Currently available applications:

- CRÈME (<https://creme.isde.vanderbilt.edu/>)
- R-GENTIC (<https://vanguard.isde.vanderbilt.edu/RGentic>)

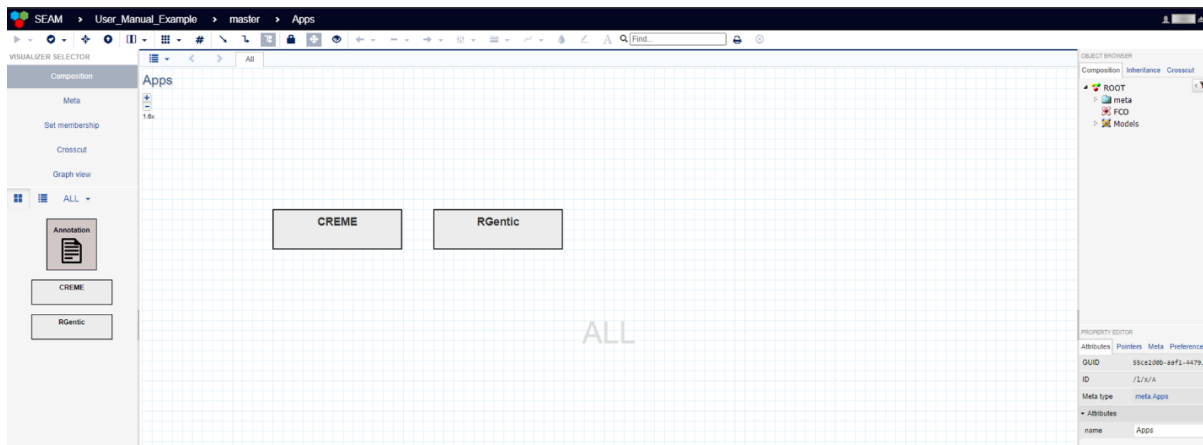


Figure IV.2 Apps page. Currently available applications that can included in SEAM projects are CRÈME and R-GENTIC.

4.2 Definitions

The Definitions folder contains sub-libraries for labels and tags used throughout a project to link different models together and complete coverage checks. The two available sub-libraries are Failure

Label and Tag Library, shown in Figure IV.3. More Definitions sub-libraries may be added in the future as the need arises.

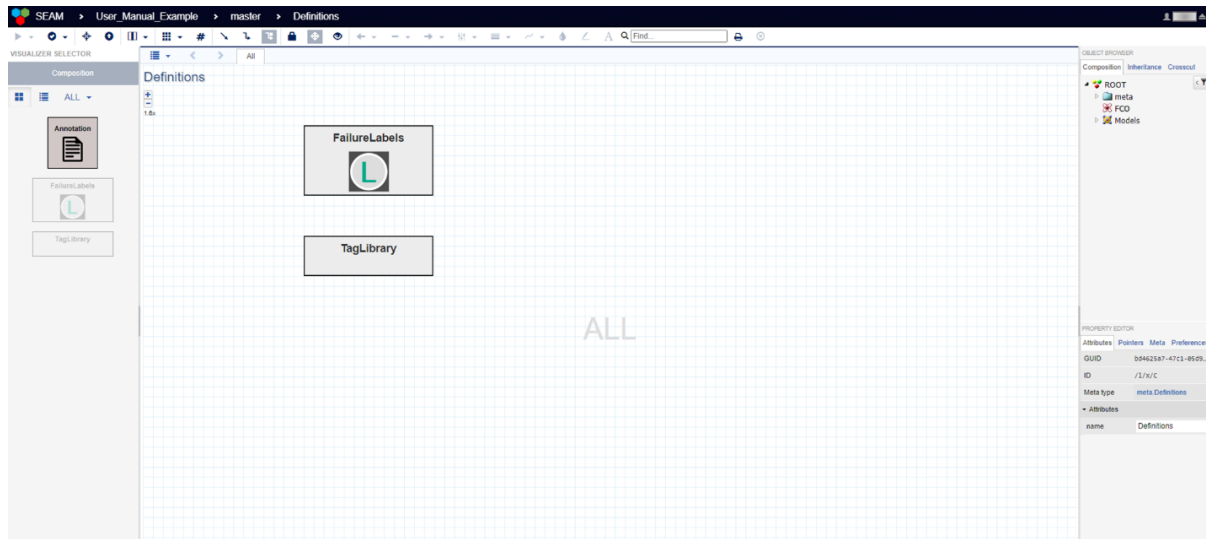


Figure IV.3 Definitions folder page. FailureLabels and TagLibrary subfolders are contained within.

Figure IV.4 shows the FailureLabel sub-library with nine (9) pre-made failure labels. Failure labels are used in SysML models to identify the specific effects (consequences) caused by a failure mode to propagate through the connections, as shown in Figure IV.7. The failure labels shown in the figure are generic labels that are used in SysML model templates and are given as examples. Table IV.1 gives a brief definition for each of the default failure labels. Users can add as many failure labels as needed for their projects by dragging more “FailureLabel” blocks from the toolbar on the left into the main folder page.

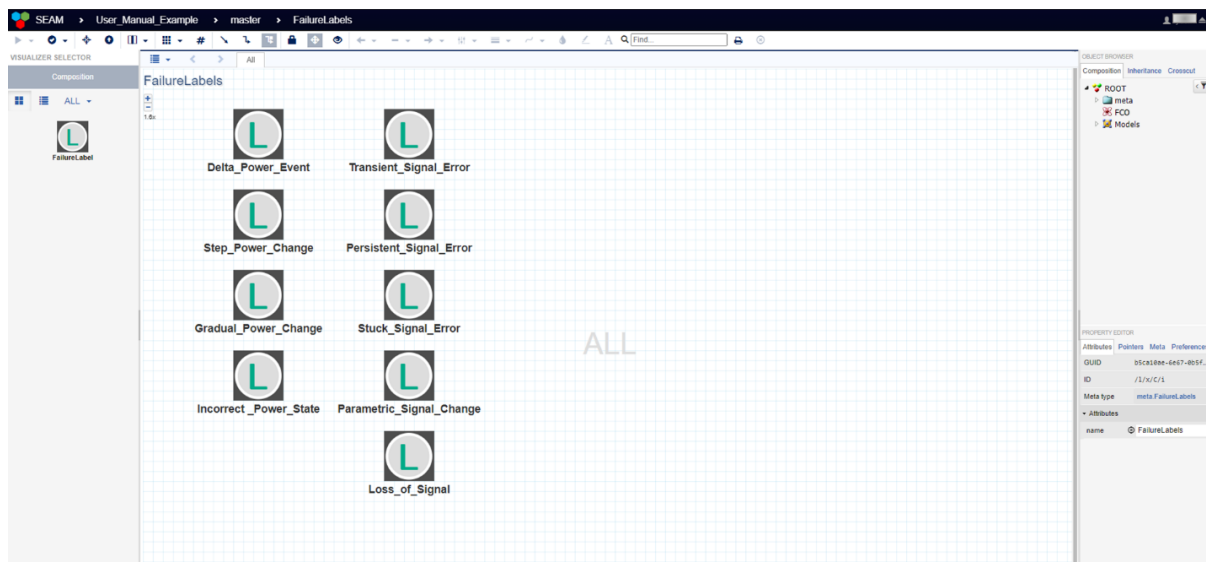


Figure IV.4 FailureLabel folder page. Failure labels to be used throughout the project are created here.

Table IV.1. Definitions of example failure labels.

Failure Label	Definition
Delta_Power_Event	A temporary, self-recovering change in power.

Failure Label	Definition
Step_Power_Change	An abrupt change from a high power state to a low power state or vice versa. The change in power state persists until the system acts on it.
Gradual_Power_Change	A slow change in power state over time.
Incorrect_Power_State	An incorrect power state.
Transient_Signal_Error	A temporary, self-recovering change in signal.
Persistent_Signal_Error	An abrupt change in signal state that persists until the system acts on it.
Stuck_Signal_Error	A persistent change in signal state that cannot be acted upon.
Parametric_Signal_Change	A slow change in single state over time.
Loss_of_Signal	A loss of signal.

The TagLibrary page is shown in Figure IV.5. Tags are used in GSN elements that are cross referenced in SysML models to associate the SysML model to its GSN goal. The tags are presented in coverage checks for the GSN models. The example TagLibrary folder only contains one tag, “RadiationCheck,” but users are able to create as many tags as needed by dragging more “Tag” blocks from the toolbar on the left into the main folder page.

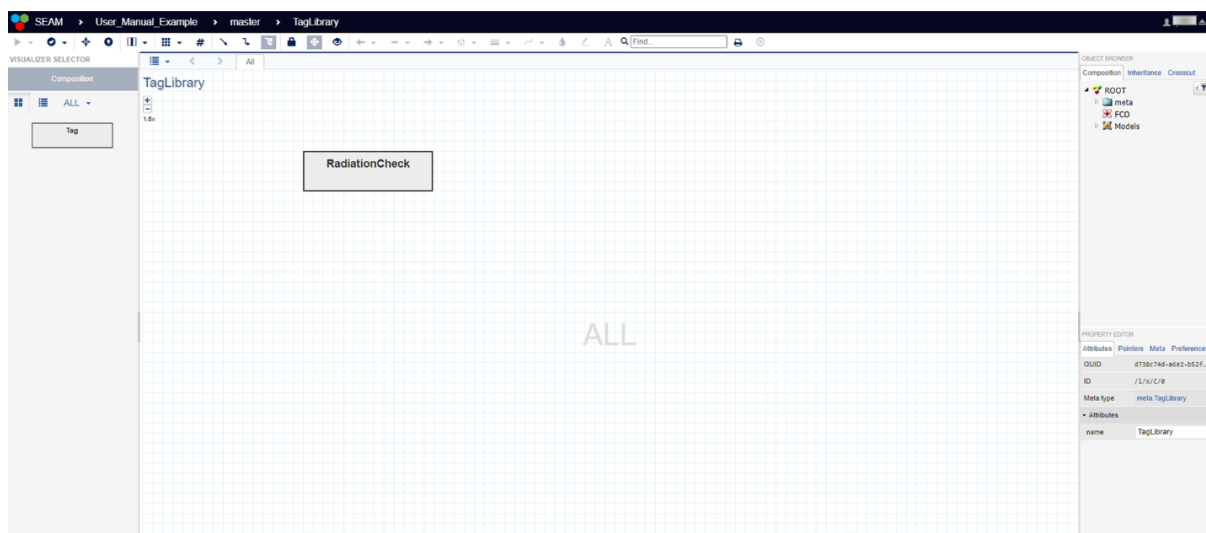


Figure IV.5 TagLibrary folder page. Tags to be used in GSN models for the project are created here.

4.3 Libraries

The Library folder contains template SysML models that can be used throughout any SysML models in the project. Projects can have an unlimited number of Library folders. Figure VI.6 shows an example library that contains four (4) SysML models of commonly used components in the system. Models in libraries can have multiple instances throughout a project, and all instances automatically update

when the library model is updated. This reduces the amount of time spent updating models when changes are made to system components. Figure IV.7 shows the SysML fault model for the Sensor component within the Library. All aspects within this model are carried over to all instances of this model in the project. If any aspect is changed, it is also changed in all instances. More is said about the specifics of SysML models in Chapter 6.

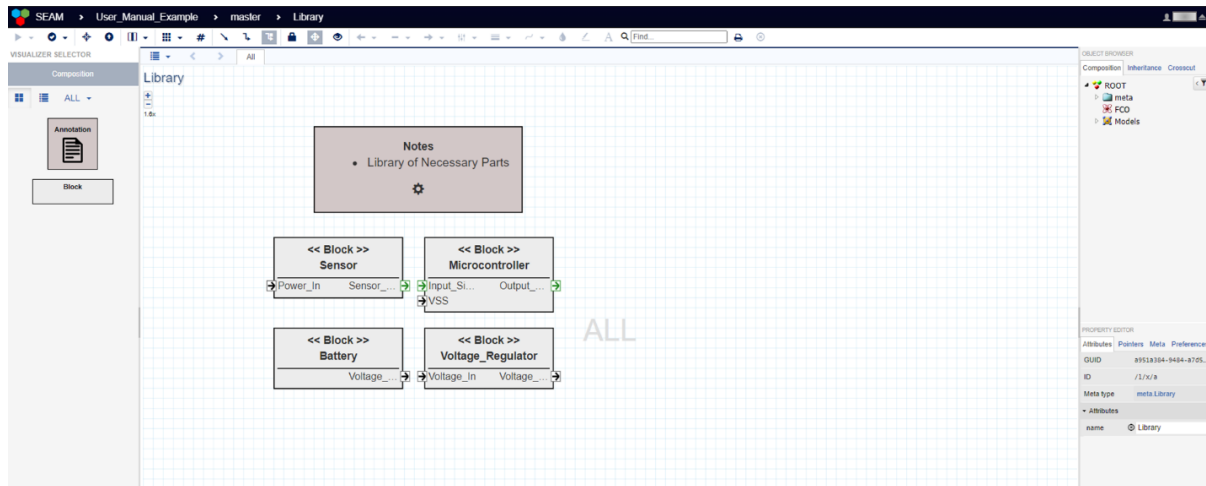


Figure IV.6 Library page. This example Library contains four (4) SysML components.

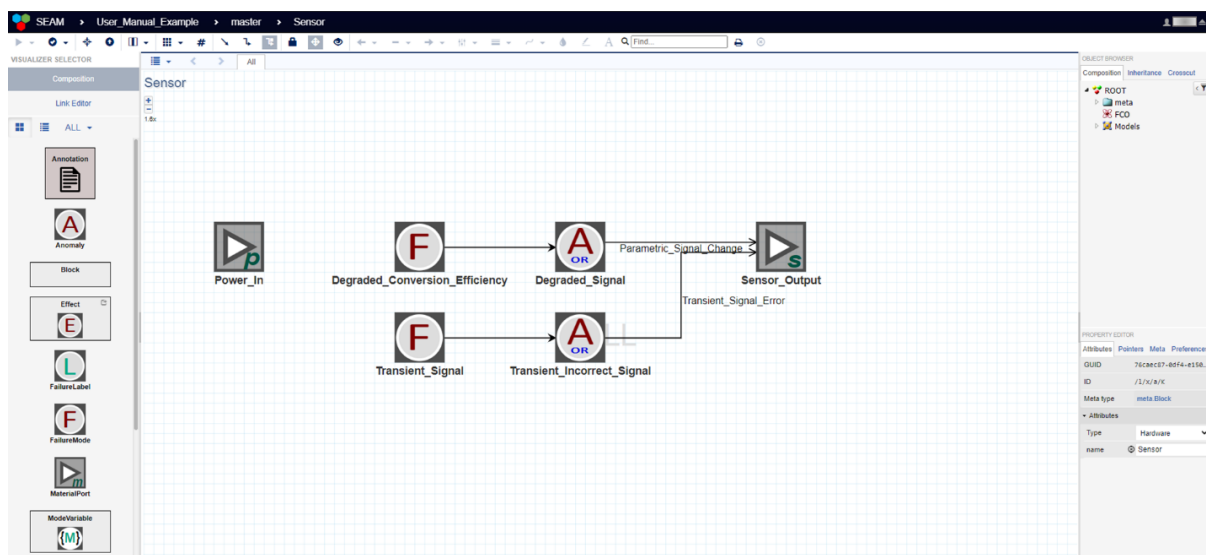


Figure IV.7 Sensor fault model in Library.

4.4 R&M Objective Hierarchy

The NASA Reliability and Maintainability (R&M) Standard “specifies technical objectives and related strategies for NASA programs and projects to be used in planning, executing, and evaluating Reliability and Maintainability,⁷” and is expressed in GSN format. This standard provides a useful starting place for GSN models.

⁷Sanford Friedenthal, Alan Moore, Rick Steiner, “OMG SysML™ Tutorial,” www.omg.sysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf, INCOSE, 2009.

Link to NASA R&M Standard: <https://standards.nasa.gov/standard/nasa/nasa-std-87291>

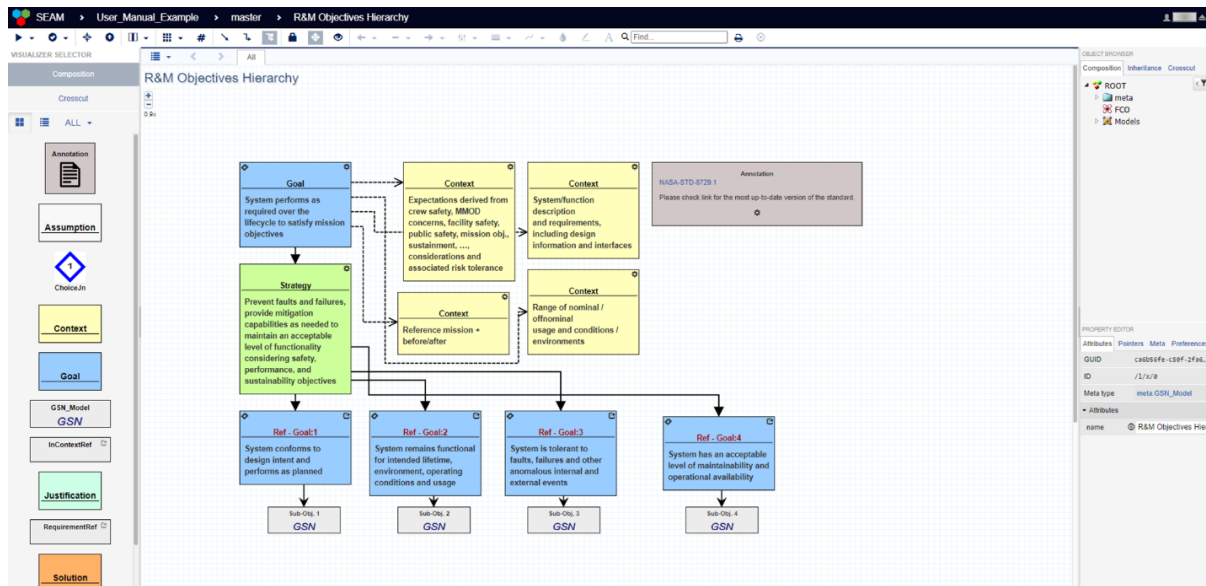


Figure IV.8 NASA R&M Objective Hierarchy. The Hierarchy is in GSN format as a reference for users.

4.5 Requirements Models

Requirements models can be created from scratch following SysML standards or imported from Cameo/ MagicDraw models. Requirements in the requirement models are cross-referenced with goals in GSN models to indicate where certain Goals in GSN model are derived from. Figure IV.9 shows example requirements.

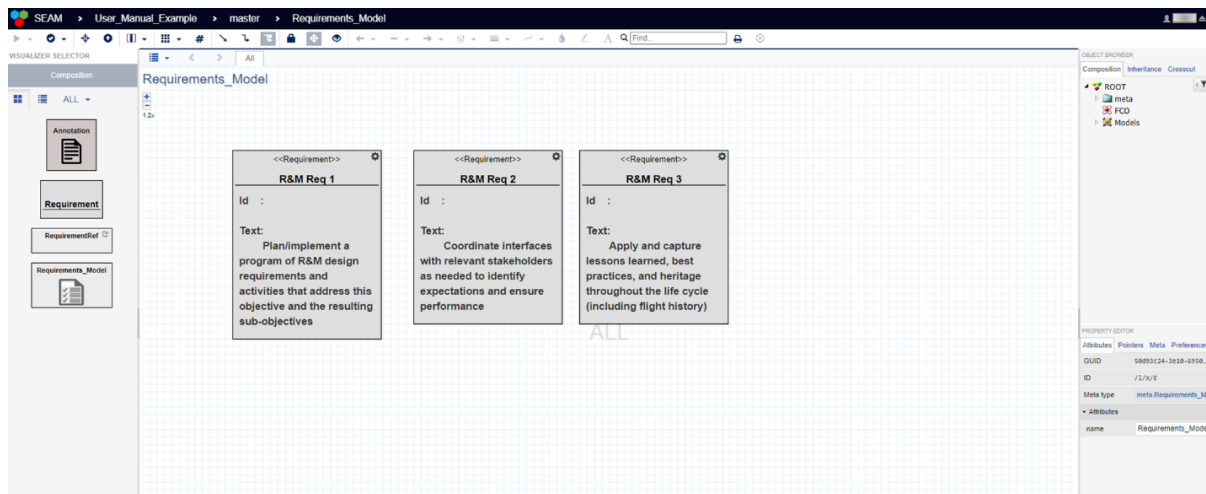


Figure IV.9 Requirements Model page.



5 Goal Structuring Notation (GSN) Models

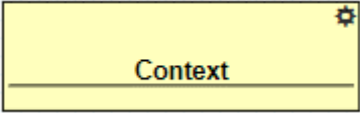



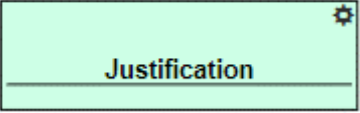
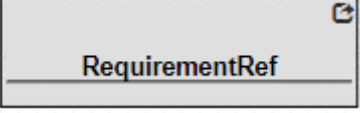
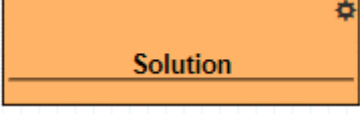
Goal Structuring Notation (GSN) assurance cases are used to document and develop assurance cases and can be developed in parallel with the SysML and Functional Decomposition models within the SEAM tool. Assurance cases are “reasoned, and compelling arguments supported by evidence that a system will operate as intended for a given, defined environment”. Assurance cases document an argument, but do not ensure the truth of the argument; assurance cases based on faulty reasoning or premises can lead to premature system failure. References to the GSN Community Standard and NASA Std 8719.1A are provided within SEAM, and the NASA R&M Objectives Hierarchy (from NASA Std 8719.1A) is provided in GSN format for reference. This chapter describes how to make GSN assurance cases in SEAM and provides a GSN example and GSN-based resources. Linking a GSN assurance case with other SEAM models such as fault propagation model is described in Chapter VIII – Linking Models.

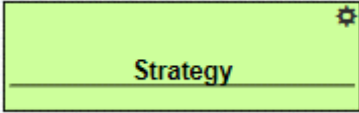
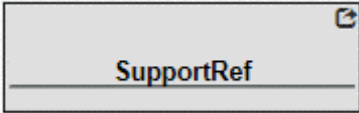
5.1 Definition of GSN Elements

GSN is a method of graphically documenting an assurance case that provides a standard for structuring the argument. The various GSN model elements and their definitions are presented below. GSN models typically start with a **goal** for the system. This **goal** may be conditioned by a **context**, such as mission environment and duration, which frames how **goals** and **strategies** should be interrupted. A **strategy** is developed for meeting the goal. From there, sub-**goals** and sub-**strategies** are used to break the argument down until a **goal** or **strategy** can be supported by a **solution**. **Solutions** are evidence for the argument, and can be test reports, simulation results, or any other piece of information that supports the claim being made. **Justifications** are used to explain why goals and strategies are acceptable in the argument. Other elements are available in the SEAM GSN models that allow for the assurance case to be connected to other SEAM models.

Table V.1 GSN Elements.

NAME	IMAGE	DESCRIPTION
Assumption		Assumption boxes are used to document what assumptions are being made about a strategy, goal, or solution.
Choice Junction		Choice junctions allow for M out of N logic paths to be used to prove a goal.

NAME	IMAGE	DESCRIPTION
Context		Context boxes are used to frame how a claim or set of reasoning should be interpreted.
Goal		Goal Goals are the claims the assurance case are trying to prove.
GSN Model		Contains a GSN model that can be linked to or used as a sub-model.
InContextRef		Reference to contexts used elsewhere in a SEAM GSN project.
Justification		Justifications provide an explanation as to why a certain claim or argument is acceptable.
Requirement Reference		Reference to project requirements that are contained in the project's Requirements Model.
Solution		Solutions provide evidence supporting the truth of an argument.

NAME	IMAGE	DESCRIPTION
Strategy		Strategy Strategies are reasoning steps to help show the validity of the goal.
Support Reference		Support Reference Reference to support material used elsewhere in the SEAM project.

5.2 NASA R&M Objective Hierarchy

The NASA Reliability and Maintainability (R&M) Standard “specifies technical objectives and related strategies for NASA programs and projects to be used in planning, executing, and evaluating Reliability and Maintainability.” A version of the R&M Hierarchy is included in GSN format for reference, and was briefly discussed in Chapter 4⁹. Figure V.1 shows the top-level view of the R&M Hierarchy which specifies the overall top-level goal for a system, “performs as required over the lifecycle to satisfy mission objectives,” a top-level strategy, and four subgoals that work to meet the overall top-level goal. These subgoals provide excellent starting places for GSN arguments for many different types of systems.

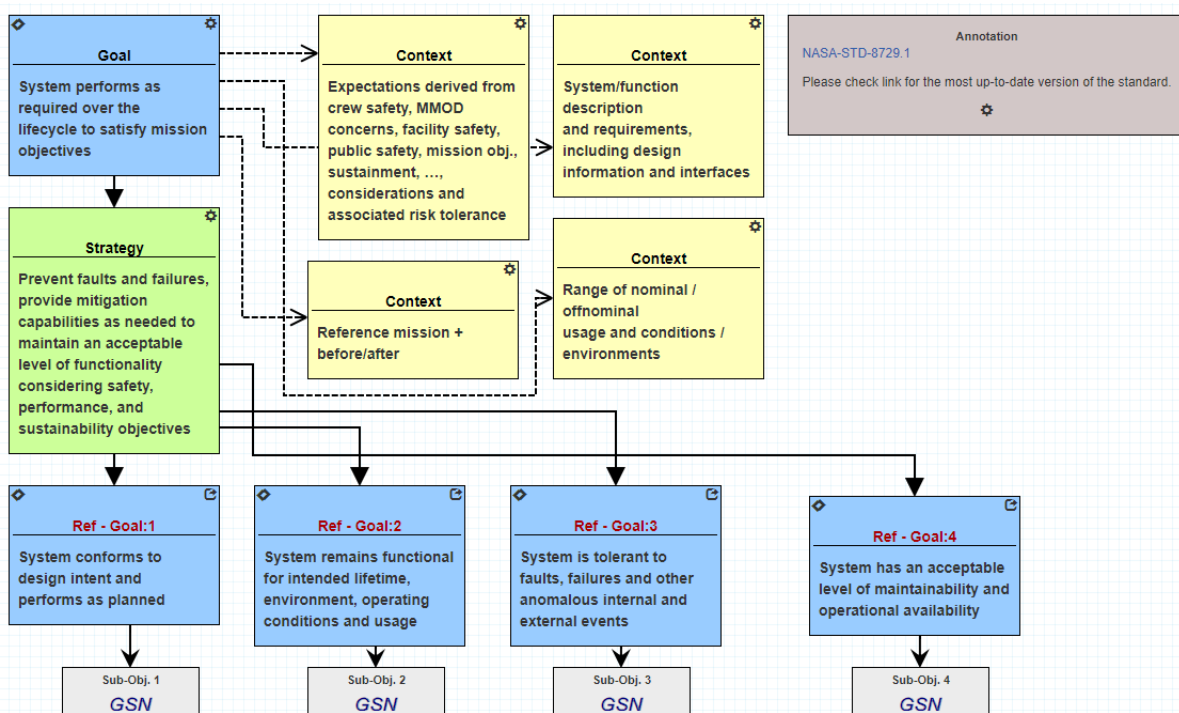


Figure V.1 NASA R&M Objective Hierarchy in SEAM GSN format as a reference for users.

⁹chapter4.md

5.3 Creating a GSN Model

Starting a GSN model from scratch begins with the canvas shown in Figure V.2. The simulation space (given by the orange box) is initially empty. GSN elements can be dragged and dropped from the list on the left (red box). The Object Browser (green box on the right) provides a means of navigating to other models within the project and the Property Editor (blue box on the right) is used to modify properties of the GSN elements.

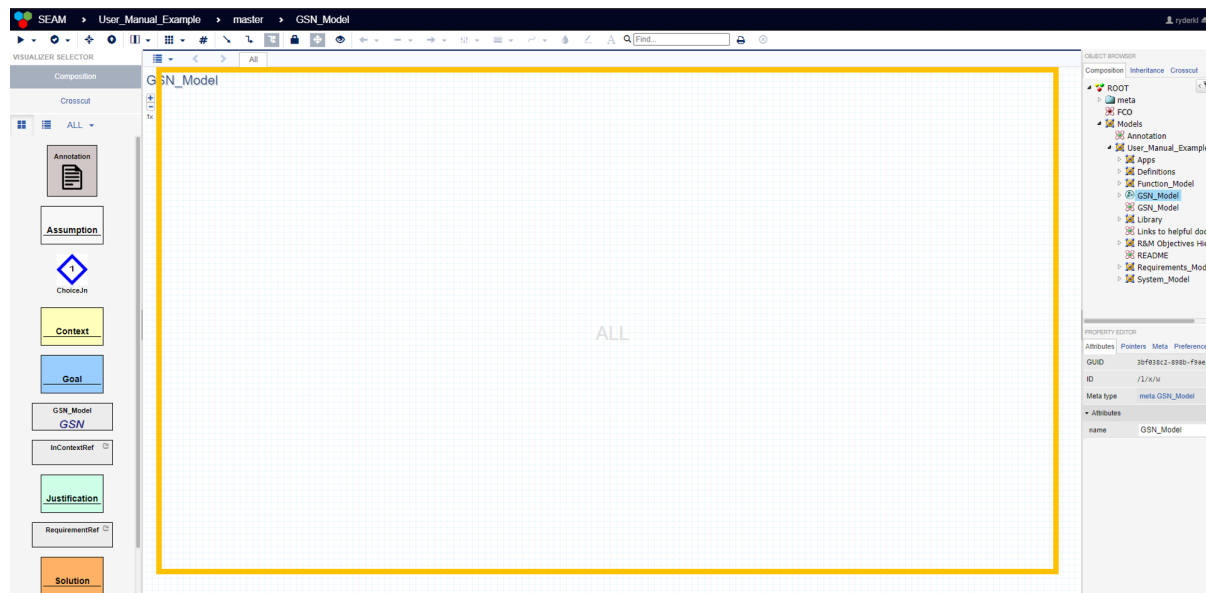


Figure V.2. Empty GSN model. GSN elements (red box) can be dragged and dropped into the simulation space (yellow box) to create GSN models. The property editor (blue box) is used to manage the properties of the GSN elements and the object browser (green box) can be used to navigate to different models within the project.

Once a GSN element has been placed in the simulation space its properties can be changed, and text can be added. Figure V.3 shows a goal element that has been placed into the simulation space (upper left), as well as the element's Property Editor (upper right), and the element's Edit Description box (bottom). Clicking on the cog icon (orange box) brings up the Edit Description box, which allows users to add text descriptions to the GSN elements. This is where the actual goals, solutions, etc. would be typed out by the user. Right clicking on the GSN element once brings up its properties in the Property Editor. For GSN elements, the most important properties are the "In Development" true/false flag, the "Meets Criteria" dropdown box, and the "name" text box. The "name" property is used to give unique names to the GSN elements. Here, the element's name is "Goal," however it could be changed to something more specific, like "Goal 1.A." The "Meets Criteria" dropdown box allows for GSN elements to be tagged as meeting criteria, not meeting criteria, partially meeting criteria, or it is unknown if it meets criteria. This creates a record of which arguments satisfy the claim being made and which do not. Finally, the "In Development" flag marks which GSN elements are completed, and which are still being developed. The diamond mark on the GSN element in Figure V.3 (red box) denotes that this element is still in development. When the flag is set to false, the diamond will disappear, allowing

users to easily see which elements are still in development and which are complete.

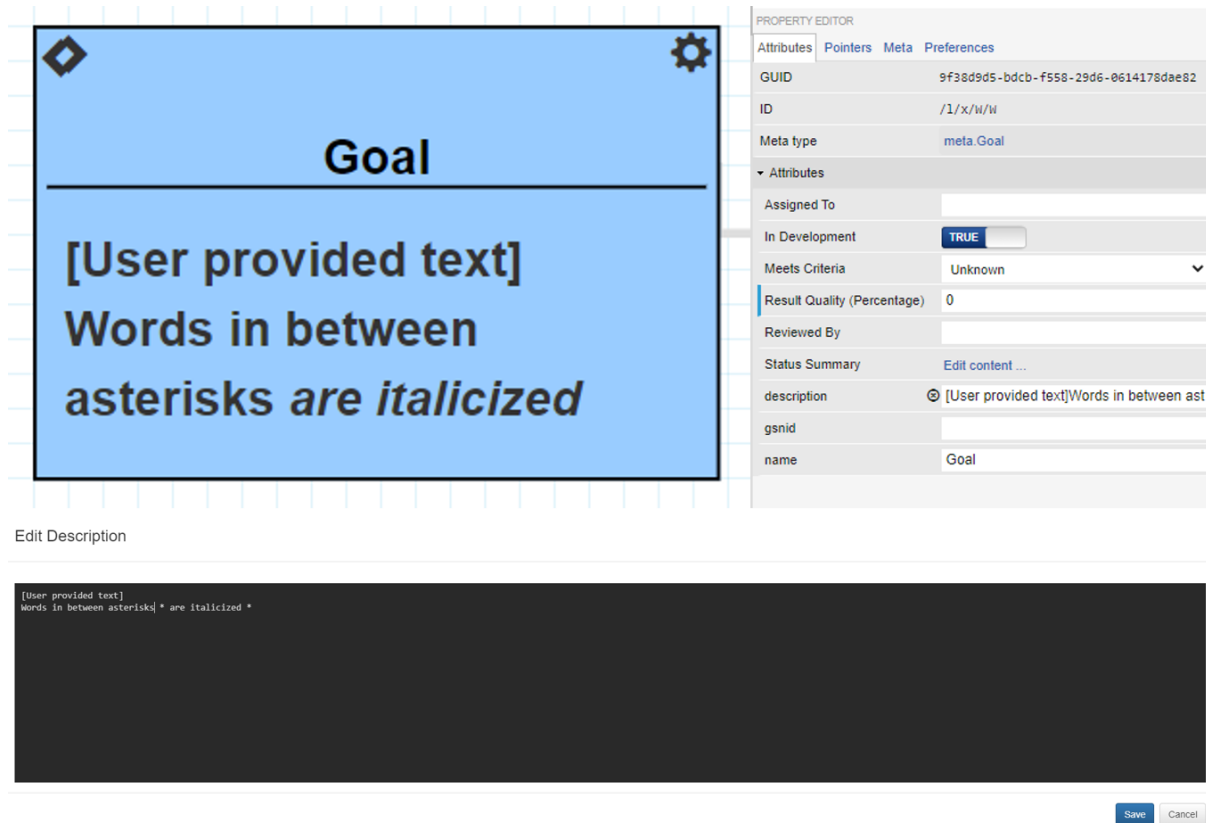


Figure V.3. A Goal GSN element (top left). The Goal’s text can be edited by clicking on the cog (orange box), which brings up the Edit Description prompt (bottom). The property editor (top right) shows the different attributes of the element that can be modified. This GSN element is still in development (red box)

All GSN elements contain the same set of properties and test boxes. By adding different GSN elements and linking them together, a GSN argument can be constructed. Table V.2 provides a list of the possible connections for the basic GSN elements from Table V.1. The solid lines represent connections in the argument, while dashed lines represent supporting documentation for the specific element the arrows are coming from.

Table V.2 GSN Elements Connections.

CONNECTION	IMAGE	DESCRIPTION
Goal-to-Strategy Strategy-to-Goal		Goals and Strategies can be connected to each other in either direction.
Goal-to-Solution Strategy-to-Solution		Goals and Strategies both connect to Solutions. Solutions are unable to form connections.

CONNECTION	IMAGE	DESCRIPTION
Goal-to-Assumption Strategy-to-Assumption	<p>The diagram shows two boxes on the left: a blue 'Goal' box and a green 'Strategy' box. On the right, there are two white 'Assumption' boxes. Dashed arrows point from the 'Goal' box to the top 'Assumption' box, and from the 'Strategy' box to the bottom 'Assumption' box. Each box has a small icon in its top right corner.</p>	Goals and Strategies both connect to Assumptions. Assumptions are unable to form connections.
Goal-to-Context Strategy-to-Context	<p>The diagram shows two boxes on the left: a blue 'Goal' box and a green 'Strategy' box. On the right, there are two yellow 'Context' boxes. Dashed arrows point from the 'Goal' box to the top 'Context' box, and from the 'Strategy' box to the bottom 'Context' box. Each box has a small icon in its top right corner.</p>	Goals and Strategies both connect to Contexts. Contexts are unable to form connections.
Goal-to-Justification Strategy-to-Justification	<p>The diagram shows two boxes on the left: a blue 'Goal' box and a green 'Strategy' box. On the right, there are two light green 'Justification' boxes. Dashed arrows point from the 'Goal' box to the top 'Justification' box, and from the 'Strategy' box to the bottom 'Justification' box. Each box has a small icon in its top right corner.</p>	Goals and Strategies both connect to Justifications. Justifications are unable to form connections.

A complete GSN argument would have no elements marked as still in development, and preferably all argument chains would be labeled a meeting criterion.

5.4 Example GSN Model

An example GSN model has been made for a generic embedded system. The generic embedded system example is used throughout Chapters V – VIII to demonstrate the various capabilities within SEAM. GSN models for more specific systems can be found in the references provided at the end of the chapter. Figure V.4 shows the top-level goal for this system: “System is able to register environmental stimulus for intended lifetime, environment, operating conditions, and usage,” which just means that the example system remains operational for the duration of the mission. This goal was taken directly from the NASA R&M Hierarchy (described in the next subsection), but in a realistic system the top-level goal would be more specific to the mission objectives. Context for the top-level goal, provided next to it, includes the proposed mission environment such as the orbit and mission lifetime. Other types of context may include functional and behavioral models of the system and mission constraints such as size, weight, and power constraints. The top-level strategy for this GSN model is to “prevent faults and failures, provide mitigation capabilities as needed to maintain an acceptable level of functionality considering safety, performance, and sustainability objectives.” This strategy was taken directly

from the NASA R&M Hierarchy as a starting point for developing this model.

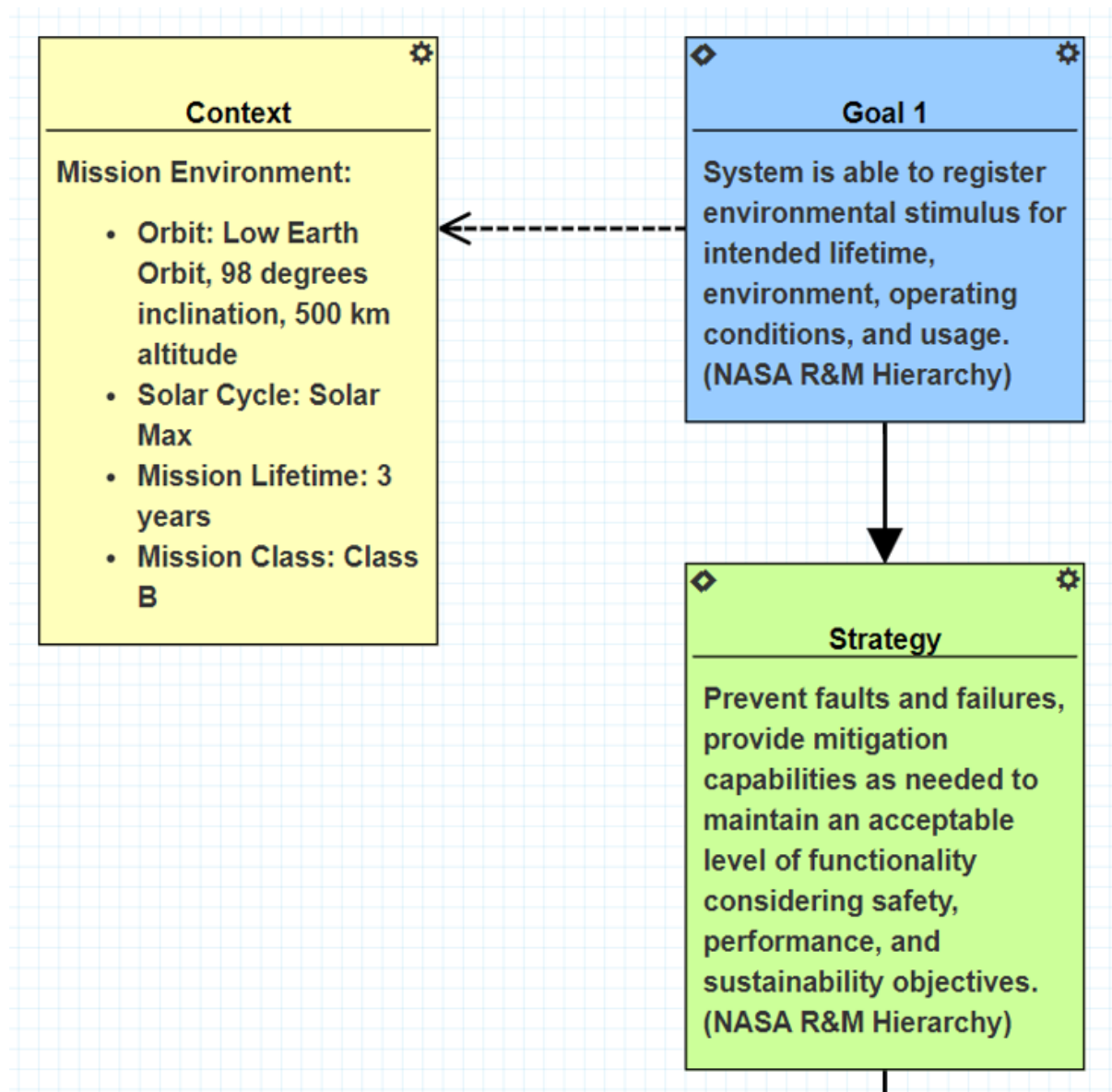


Figure V.4. Top-level goal, top-level strategy, and mission environment context for an example GSN argument.

Beneath the top-level goal and strategy, sub-goals and sub-strategies are used to breakdown the argument until a goal can be supported directly by a piece of evidence. Figure V.5 shows some of the sub-level goals that help support the top-level goal and strategy. To ensure this example system can meet its top-level goal, the system needs to be “tolerant to faults, failures, and other anomalous in-

ternal and external events,” as do each of the system’s subsystems.

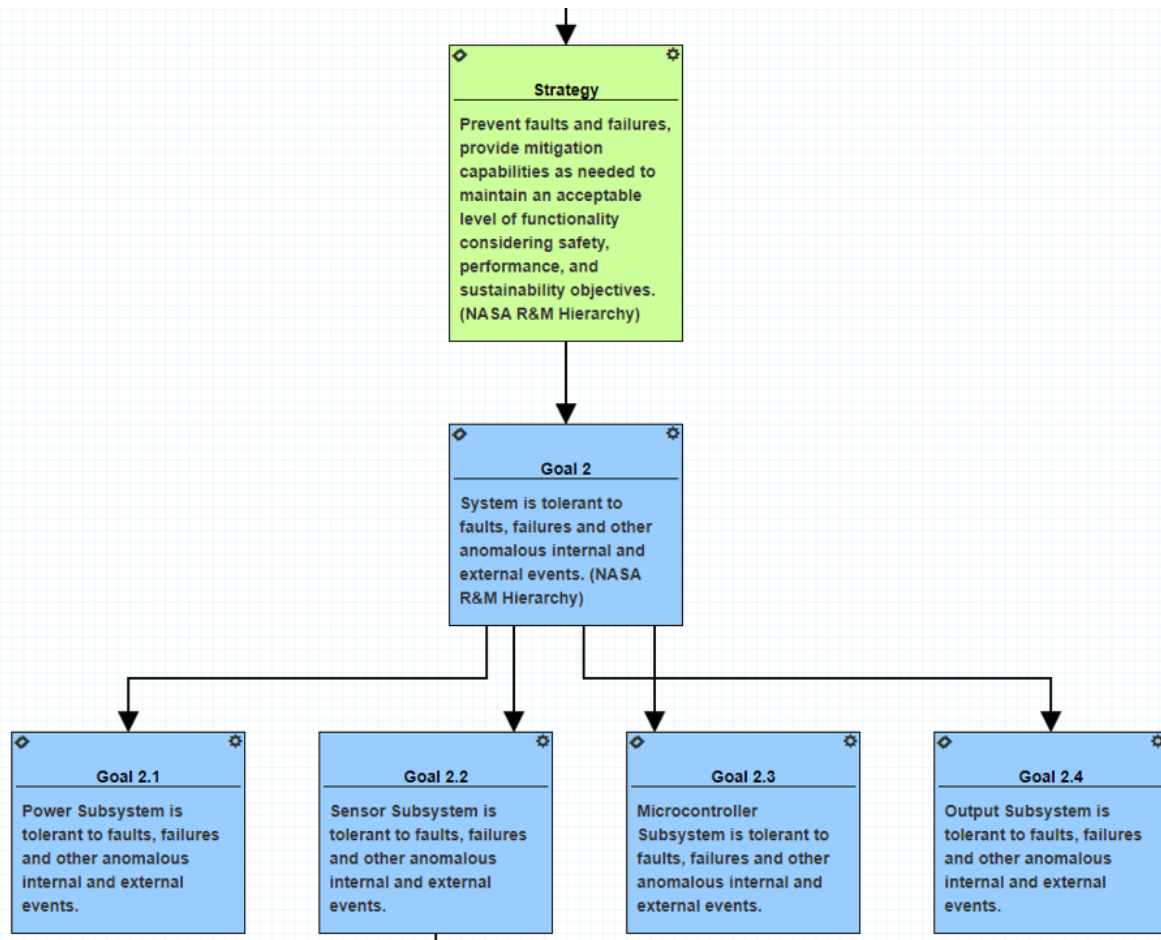


Figure V.5. Sub-goals used to support the top-level strategy.

Sub-goals continue to be broken down into more sub-goals and sub-strategies until a goal can be supported directly with evidence and then marked complete. Figure V.6 shows the rest of the GSN argument for Goal 2.2 “Sensor Subsystem is tolerant to faults, failures, and other anomalous internal and external events.” This sub-goal has one strategy that was taken from the NASA R&M Hierarchy, and two sub-goals have been identified as paths toward demonstrating the Goal 2.2 has been met. These sub-goals are specifically related to the possible radiation effects, total ionizing dose (TID) and single event effects (SEEs), that may occur in the Sensor Subsystem. Both sub goals 2.2.1 and 2.2.2 are considered complete because they have completed solutions that show the goals have been met. Testing was done for both types of radiation effects and the Sensor Subsystem was found to remain within specifications. Test reports could be attached to these solutions as evidence of the goals being met. Goal 2.2.2, which relates to SEEs, also has a justification attached to it., which explains why some

amount of SEEs in the Sensor Subsystem are considered acceptable to the overall project.

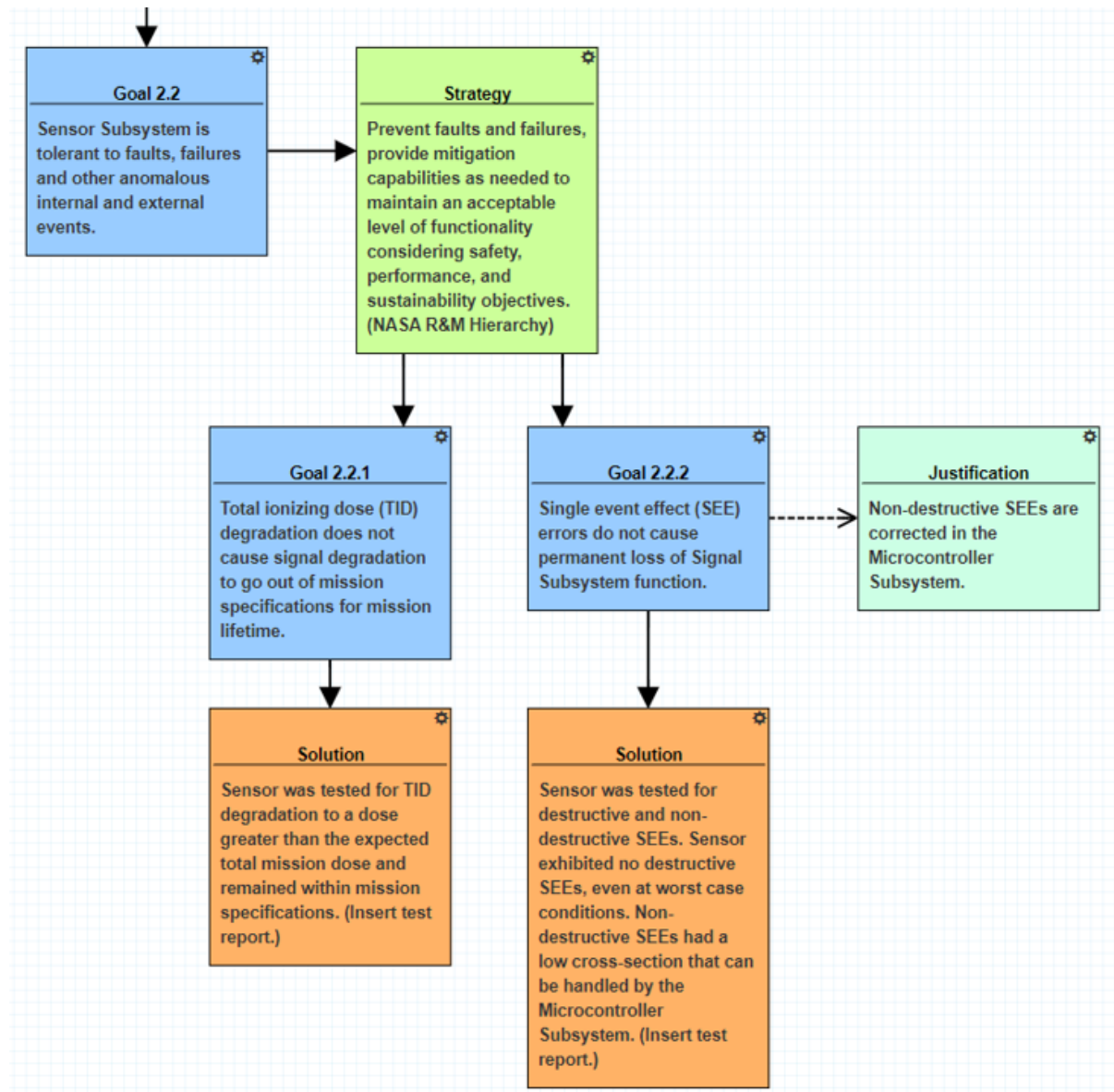


Figure V.6. A completed GSN argument for sub-goal 2.2 with solutions that can be supported by evidence and justifications for why certain behaviors are acceptable.

Goal 2.2 is considered complete and the “In Development” marker does not appear on any of the arguments beneath it. In order for the top-level goal to be complete, all sub-goals beneath it need to have completed arguments and no longer be “In Development.” Once that is true, the GSN argument is considered complete. It should be reiterated that a complete GSN argument does not necessarily mean the argument is true and valid, or that the argument will not change during a project’s lifetime.

5.5 GSN Resources

- NASA R&M Objectives Hierarchy (Standard¹⁰, Slides¹¹)
- Goal Structuring Notation standard¹²

¹⁰<https://standards.nasa.gov/standard/nasa/nasa-std-87291>

¹¹<https://sma.nasa.gov/docs/default-source/News-Documents/r-amp-m-hierarchy.pdf>

¹²<https://scsc.uk/gsn>

- R. A. Austin “A Radiation-Reliability Assurance Case using Goal Structuring Notation for a Cube-Sat Experiment,” M.S. Thesis, Vanderbilt University, July 6, 2016¹³
- A. Witulski et al., “Goal Structuring Notation in a Radiation Hardening Assurance Case for COTS-Based Spacecraft,” GOMAC 2016¹⁴
- R. A. Austin et al., “A CubeSat-Payload Radiation-Reliability Assurance Case using Goal Structuring Notation,” RAMS 2017.¹⁵
- J. W. Evans et al., “Towards a Framework for Reliability and Safety Analysis of Complex Space Missions”¹⁶

¹³<https://ir.vanderbilt.edu/handle/1803/12763>

¹⁴https://modelbasedassurance.org/documents/GSN_GOMAC.pdf

¹⁵https://modelbasedassurance.org/documents/GSN_RAMs.pdf

¹⁶https://modelbasedassurance.org/documents/MBAC_AIAA.pdf

6 SYSTEMS MODELING LANGUAGE (SYSML) MODELS

Systems Modeling Language (SysML) is a graphical language for systems engineers, for representing a complex system in terms of its structure, behavior, and dependencies. The model construction process utilizes and arranges SysML “blocks” in such a way that it captures the system’s architecture and its behavior. Specifically, the blocks represent all possible means by which the system can respond to environmental stimulus. This type of schematic, which consists of blocks in their correct arrangement according to its system architecture, can be presented to non-technical contributors (e.g., customers, funders, etc.) because it is an intuitive top-level view of the system. Also, as can be inferred, SysML is also useful in managing the development of complex systems and enhancing knowledge capture.

6.1 Definition of Boxes

A SysML block diagram describes the system’s structure by focusing on the flow of information (data) and power through the system. The diagram is composed of blocks and interconnections. A block (Figure VI. 1, top image) represents a system component that encapsulates all the details of that component (for example, constraints and properties). The name of the block can be changed by going to the ‘Attributes’ section on the right side of the work screen and changing the block’s name in ‘name’ (Figure VI. 1, top image). It is also possible to change the border color of the SysML block by going to ‘Preference’ on the right side of the screen (after clicking the block) and clicking the empty rectangle next to ‘borderColor’ under ‘Color.’ Similarly, the color of the SysML block itself can be changed by clicking the empty rectangle next to ‘color’ (Figure VI. 1, middle image).

Ports on the blocks represent interfaces that can allow blocks to connect with each other via interconnections. These interconnections represent how the blocks interact with each other.² The colors of a connection between blocks can be adjusted by clicking the connection and going to ‘Preference.’ From there, the user can change the color by clicking the empty rectangle next to ‘color’ under the

‘Color’ tab (Figure VI. 1, bottom image).

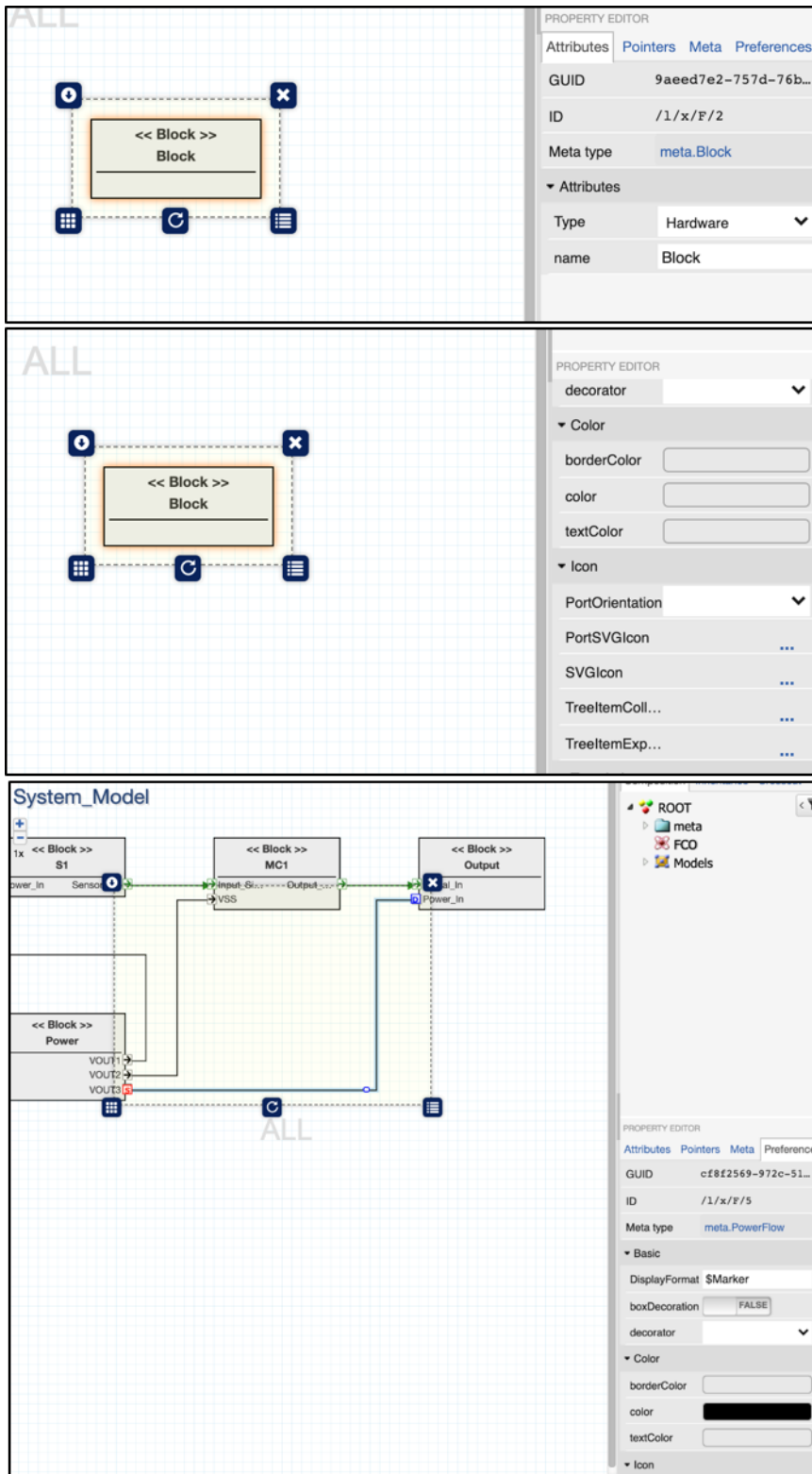


Figure VI.1. All properties of a SysML Block

6.2 Creating a SysML Model

To create a SysML model, the user must first create a 'System_Model' library folder and enter it (Figure VI. 2).

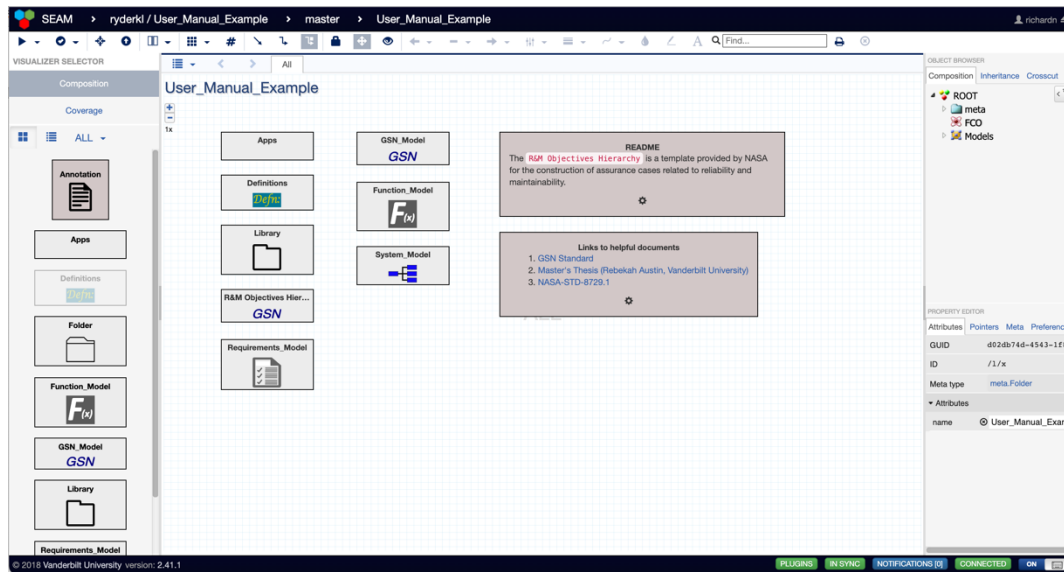


Figure VI.2. Workspace containing the 'System_Model' Folder.

This folder already has a pre-created SysML schematic (Figure VI. 3). It consists of four blocks interconnected in various ways. In order to create a block (Figure VI. 4), the user will need to drag and drop a block from the left-side bar onto the screen. Blocks can be connected via ports.

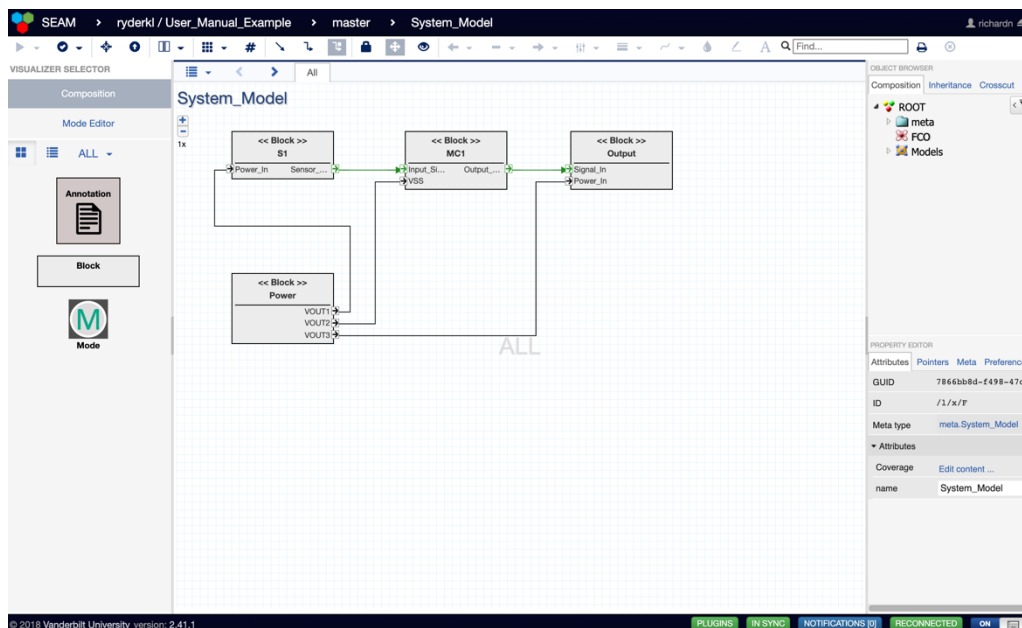


Figure VI.3. System Architecture Schematic within the 'System_Model' Library Folder

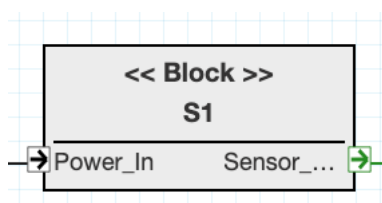


Figure VI.4. Example of a SysML block. In this case, it is a Sensor block.

Double-clicking inside any of these blocks (such as Figure VI. 4) allows the user to view the failure modes within the clicked block (Figure VI. 5). In the case of the Sensor block, we see an input power port ('Power_In') two 'Failure Modes' (Total_Ionizing_Dose and Single_Event_Effect), two 'Anomalies' (Degraded_Signal and Transient_Incorrect_Signal), and one output signal port (Sensor_Output). The two anomalies also have failure labels identifying how the anomalies are expressed at the output port. For example, 'Persistent_Signal_Error' is an error that can be recovered from by resetting the system. 'Stuck_Signal_Error' is an error that causes permanent damage and cannot be changed back. All other failure labels can be found in Figure VI. 6, with definitions for each failure label located on the right box titled 'Annotation' in Figure VI. 6.

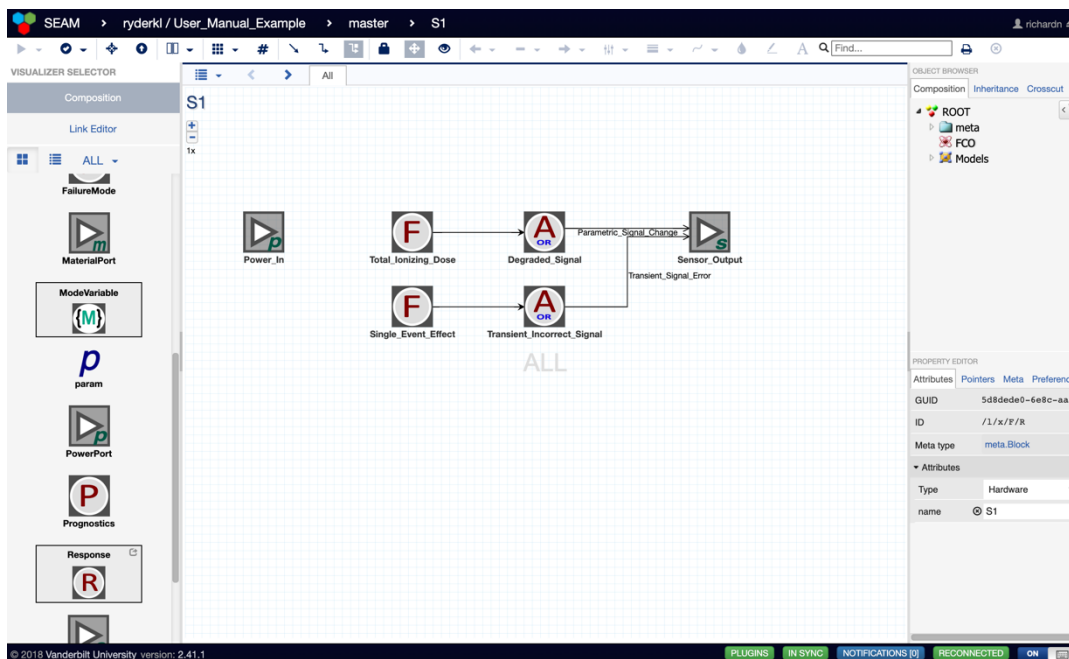
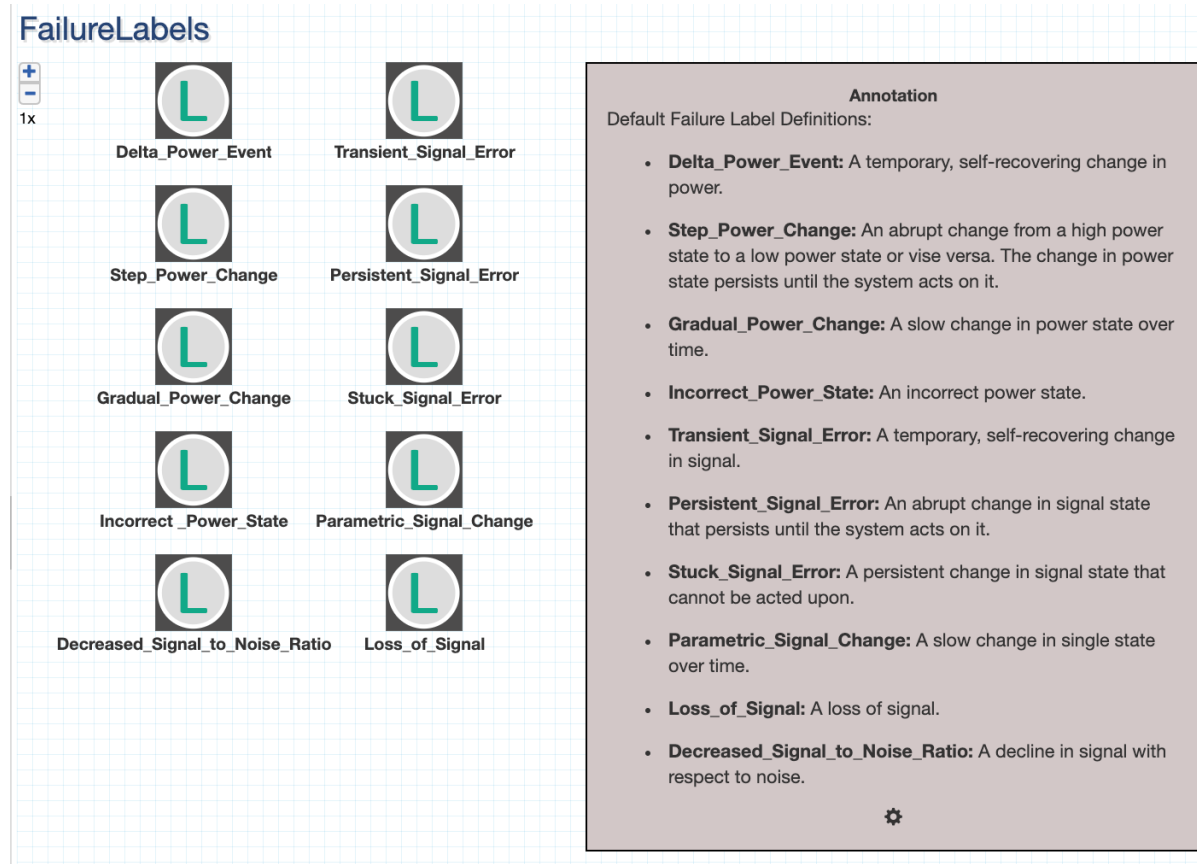


Figure VI.5. Schematic of Failure Modes for the Sensor Block**Figure VI.6. . List of all possible Failure Labels with definitions**

6.3 Example SysML Model

A SysML block diagram can be found in Figure VI. 7. This figure represents a sensor by showing all the individual subsystems that compose this system: the sensor subsystem, the microcontroller subsys-

tem, the output subsystem, and the power subsystem.

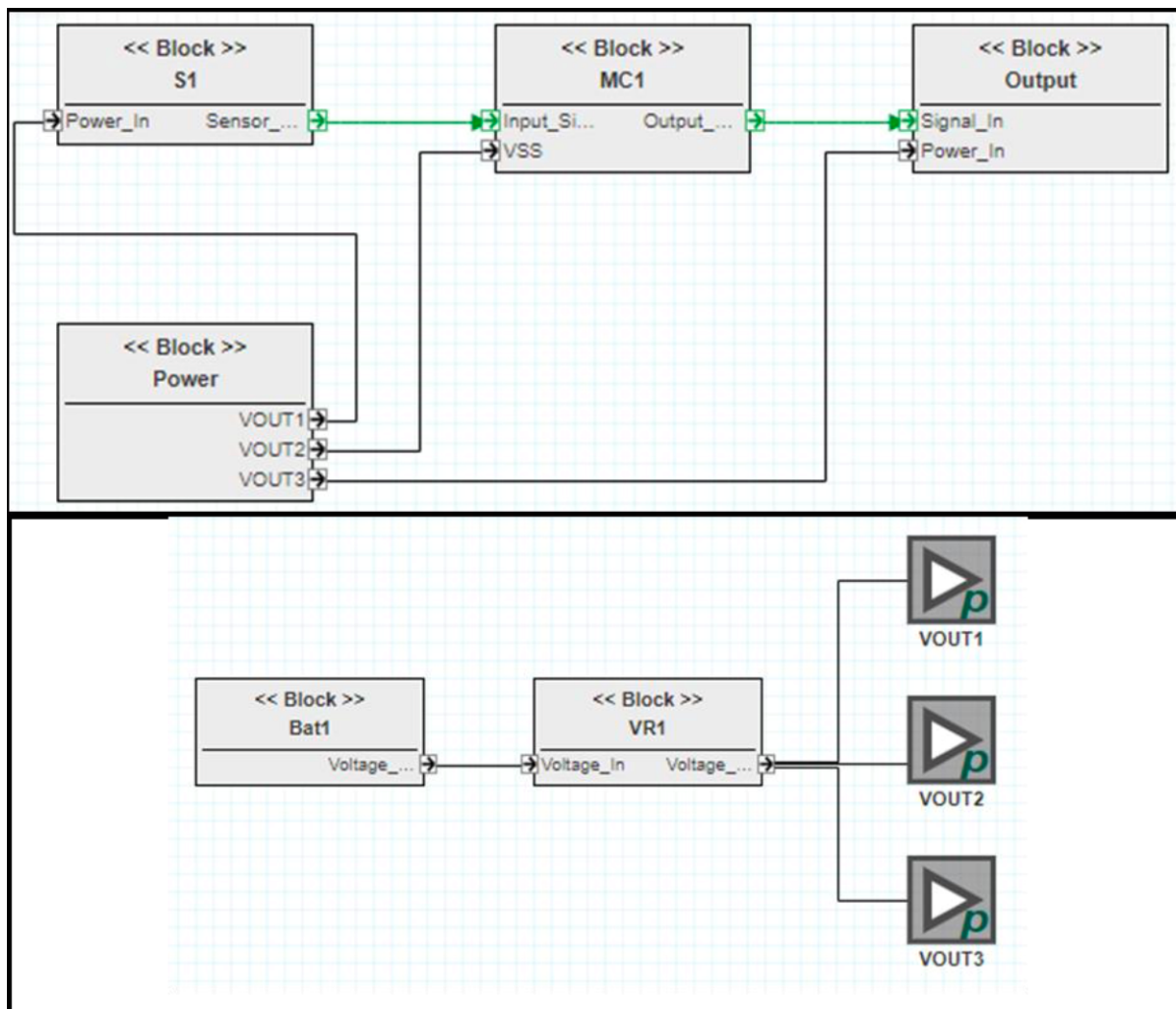


Fig. 2 (Top) SysML block diagram of a generic embedded system that uses a sensor to determine the next state of the system. Each block represents an abstraction of a subsystem: Sensor subsystem (S1), Microcontroller subsystem (MC1), Output subsystem, and Power subsystem. (Bottom) Lower level abstraction of the Power subsystem.

Figure VI.7. SysML block diagram of a system using a sensor. Each block represents an abstraction of a subsystem: Sensor subsystem (S1), Microcontroller subsystem (MC1), Output subsystem, and Power subsystem3

6.4 SysML Model Resources

1. Object Management Group, "SysML Diagram Tutorial," SysML.org. <https://sysml.org/res//tutorials/sysml-diagram-tutorial/index.html> (accessed Sep. 08, 2021).
2. K. L. Ryder et al., "SYSTEMS ENGINEERING AND ASSURANCE MODELING (SEAM): A WEB-BASED SOLUTION FOR INTEGRATED MISSION ASSURANCE," Facta Universitatis, Series: Electronics and Energetics, vol. 34, no. 1, Art. no. 1, Feb. 2021.

7 Functional Decomposition Models

The Functional Decomposition Model (FDM) of a system demonstrates what functions the system implements and how they relate to components of the system. The functions themselves are descriptions of capabilities set by requirements/specifications, and represent a form of abstraction for assessing reliability, availability, and safety of a system. The system takes the form of a hierarchical assignment of responsibility for the accomplishment of a function to sub-functions and components.¹

7.1 Definition of Boxes

Functions, represented as $F(x)$, are connected to sub-functions, represented as $f(x)$, which are connected to specific instances of components that support the sub-function¹. In the figure below, only one component is associated with each sub-function. However, in practice, each sub-function can be associated with as many component instances as necessary¹.

In a typical system, the sub-functions (most often) involve one of the following: power, sensing, computation, actuation, or radiation effect mitigation. Functions typically refer to one of the following: a form of user input or output, or a form of remote communication¹.

7.2 Creating a Functional Decomposition Model

In order to create a Functional Decomposition Model, the user must first drag and drop a 'Function_Model' folder from the left side of the screen in Figure VII. 1 onto the main workspace. It is important to note that Figure VII. 1 has been pre-filled and would normally appear empty for a new project.

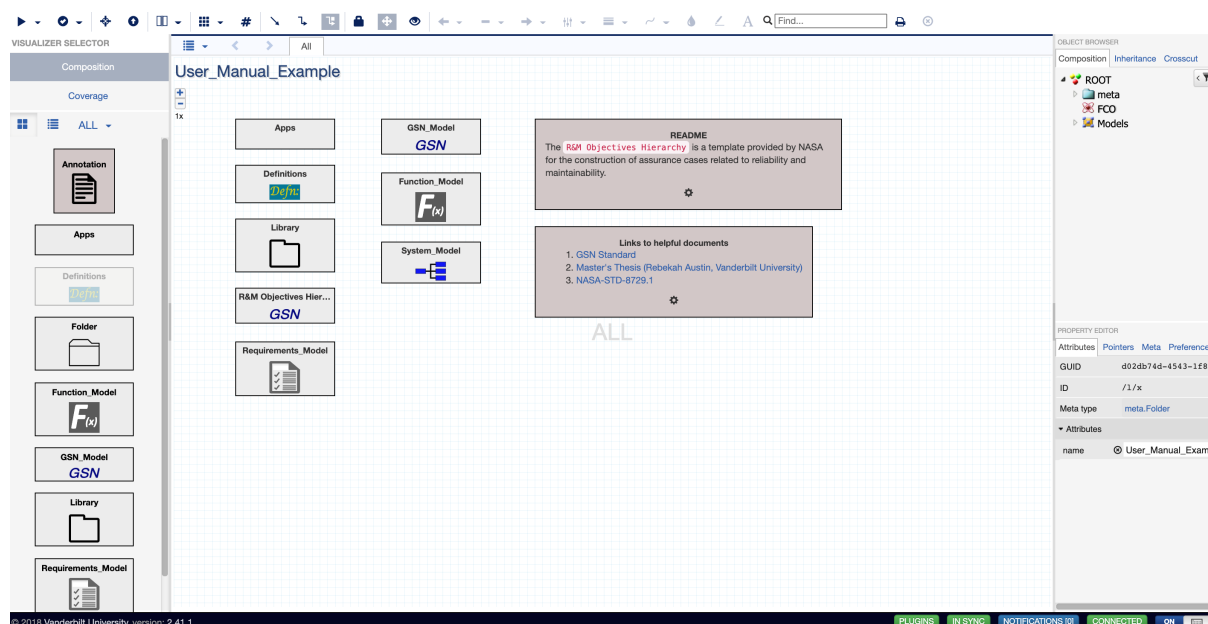


Figure VII.1. Workspace where user drags and drop a 'Function_Model' Library onto the workspace

After double-clicking into the 'Function_Model' folder, the user can construct the Functional Decomposition Model (Figure VII. 2). The user can drag a top function block from the left side of the screen

onto the workspace. The user can also relabel that function by clicking the block, looking to the attributes section on the bottom right side of the screen, and changing the name for that block.

The user can create primitive functions in the same manner, as well as rename the primitive functions in the same manner. Once all user-desired top functions and primitive functions are added onto the workspace, the user can connect top functions to primitive functions by hovering over the top function, clicking one of the sides of the block, and dragging it to one of the inputs of the primitive functions. The ‘Power-Ref,’ ‘Sensor-Ref,’ ‘Microcontroller-Ref,’ and ‘Output-Ref’ are examples of linking SysML models into the Functional Decomposition Model and will be discussed in Chapter 8 (titled “Linking Models”).

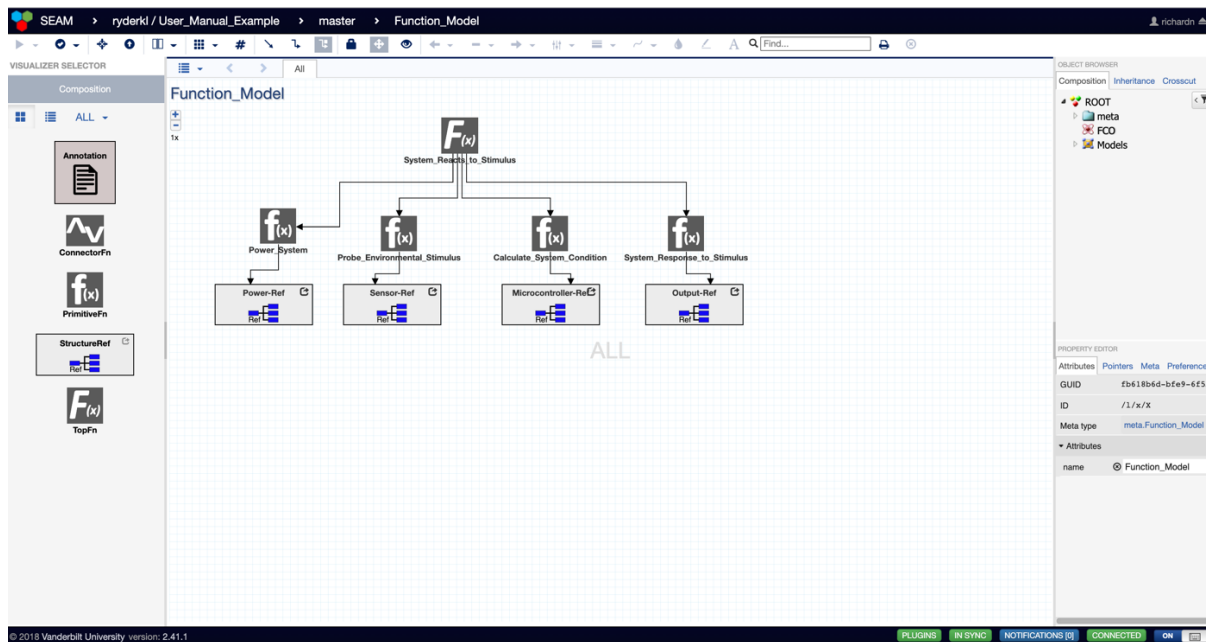


Figure VII.2. Completed Functional Decomposition Model

7.3 Example Functional Decomposition Models

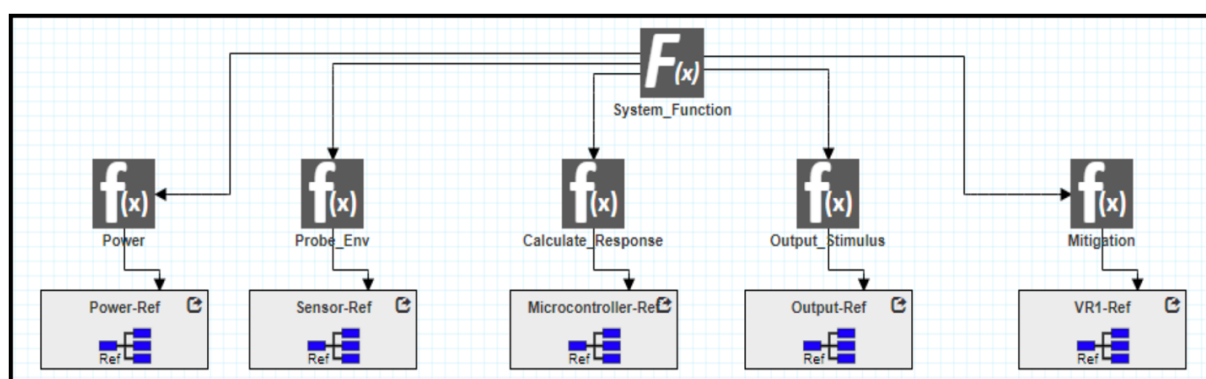


Figure VII.3. Example of a Functional Decomposition Model [1]

Figure VII. 3 represents an example of a functional decomposition model. It begins with a high-level function $F(x)$ and branches off into multiple smaller sub-function (or primitive functions) $f(x)$. These sub-functions vary for Power functions, Sensing function (for probing the environment and system itself), Calculate Response function, Output Stimulus function, and Mitigation function. Through linkage, these subfunctions can be connected to SysML models (which will be discussed in Chapter 8 titled “Linking Models”).

7.4 Functional Decomposition Model Resources

1. K. L. Ryder et al., "SYSTEMS ENGINEERING AND ASSURANCE MODELING (SEAM): A WEB-BASED SOLUTION FOR INTEGRATED MISSION ASSURANCE," Facta Universitatis, Series: Electronics and Energetics, vol. 34, no. 1, Art. no. 1, Feb. 2021.

8 Linking Models

8.1 GSN + SysML

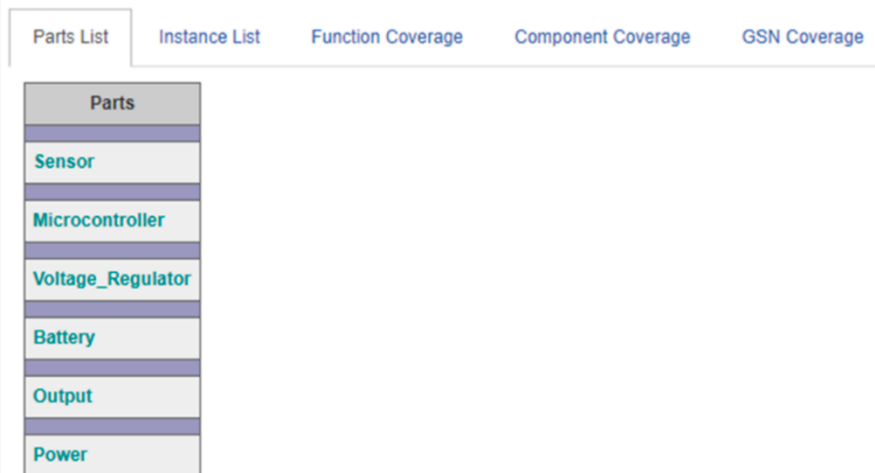
8.2 SysML + Functional Decomposition Model

8.3 Functional Decomposition + GSN

9 SEAM Outputs

9.1 Coverage Check

A coverage check is a process in SEAM that allows the cross-referencing of elements found in the System Architecture Model, the Functional Decomposition Model, and the GSN Assurance Model of a specific system. During the coverage check, the goals, strategies, and solutions in GSN assurance arguments are related to the functional model and the system model. This process is particularly useful for complex systems where the assurance argument needs to account for a myriad of components, as well as for their functional inter-dependences and propagated failures.



Parts
Sensor
Microcontroller
Voltage_Regulator
Battery
Output
Power

Figure IX. 1. Coverage check for the parts list. Each row corresponds to a part model in the parts library



Instances	Parts
MC1	Microcontroller
Bat1	Battery
VR1	Voltage_Regulator
VR2	Voltage_Regulator
VR3	Voltage_Regulator
S1	Sensor

Figure IX. 2. Coverage check for the instance list. Each row corresponds to a component instance in the system model and its associated part in the part library

Each coverage check consists of a parts list, an instance list, a function coverage, a component coverage, and a GSN coverage. The parts list (Figure IX. 1) is a table of part type models associated with the current system model. Each part type model also serves as a hyperlink that allows the user to navigate to the part model and its internal fault model.

The instance list (Figure IX. 2) is a table presenting all component instances in the system's architecture model. For each row, the component instance shows the corresponding part from the part li-

brary.

Parts List	Instance List	Function Coverage	Component Coverage	GSN Coverage
		Function	Implementor Component(s)	
		System_Response_to_Stimulus	Output	
		Probe_Environmental_Stimulus	S1 : Sensor	
		Power_System	Power	
		Calculate_System_Condition	MC1 : Microcontroller	

Figure IX. 3. Coverage check for the function coverage. Each row corresponds to a function in the function decomposition model and the component (part) in the system design that implements the function.

The function coverage list (Figure IX. 3) is a table providing a summary of the functional decomposition model. For each row, the function lists the component(s) implemented in the system model. This table allows functions that lack a corresponding implementation component in the system to be flagged. A function that lacks a corresponding implementation component informs the user that either the functionality is not implemented, or the relationships have not been captured in the functional decomposition model.²

The component coverage list maps component instances in the system model to the appropriate function(s) in the functional decomposition model. It is likely that component instances of the same part support different functions in the system. Entries that are not related to any function are appropriately flagged.²

The GSN coverage (Figure IX. 4) list presents a table detailing the coverage of assurance arguments in the GSN model relative to the components in the system. It also lists the GSN goals and solutions for the assurance argument related to each component instance in the system. It further identifies the specific GSN goals related to each of the underlying component faults, each functional degradation effect of the propagating fault, and each mitigation strategy associated with the fault propagation.²

Overall, Figure IX. 4 shows an example GSN coverage table generated as part of the coverage check. The first column corresponds to the component instance; the next column corresponds to any fault originating from within the component; this is followed by the effect (E) and the mitigation response (R) related to the fault propagation. The links to the GSN arguments are listed in the next column.

Entries with no associated GSN arguments are flagged with a red question mark (“?”) symbol.

Component	Fault	Effect	Response	GSN Node	Tag	Status	Result	Information Source	Action	Assigned To	Comment
MC1 : Microcontroller				Goal 2.3 :		In Development	Unknown	Instance	-	-	-
	Single_Event_Upset			?					-		
	Single_Event_Latchup			?					-		
Output				Goal 2.4 :		In Development	Unknown	Instance	-	-	-
Power				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
Bat1 : Battery				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
VR1 : Voltage_Regulator				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
VR2 : Voltage_Regulator				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
VR3 : Voltage_Regulator				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
S1 : Sensor				Goal 2.1 :		In Development	Unknown	Instance	-	-	-
				Goal 2.2.1 :		Developed	Yes	Instance	-	-	-
				Goal 2.2.2 :		Developed	Yes	Instance	-	-	-
	Total_Ionizing_Dose			Goal 2.2.1 :		Developed	Yes	Instance	-	-	-
	Single_Event_Effect			Goal 2.2.2 :		Developed	Yes	Instance	-	-	-

Figure IX. 4. Coverage check for GSN coverage. The table maps the parts in the system design and their underlying faults to the GSN arguments (goals/solutions).

The next few columns reveal the status of the specific GSN goal (Developed/ In-development), and the result of the argument based on if it meets the specifications or not (yes/ no/ partial). “Information Source” indicates if the argument pertains to the specific component instance or the part it corresponds to. Action column reveals the user decision on the completion status of the argument (completed/needs attention/ ignore). The penultimate column allows for traceability and assignment of individual arguments to persons. The comments column keeps a record of the comments related to the decisions made pertaining to the arguments.

9.2 Creating a Coverage Check

Before creating a coverage check, the user first begins with a GSN model (Figure IX. 5).

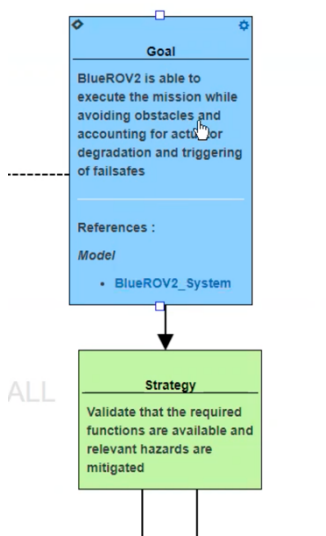


Figure IX. 5. Example of a GSN Model

Connected to the GSN model should be a functional decomposition model (Figure IX. 6).

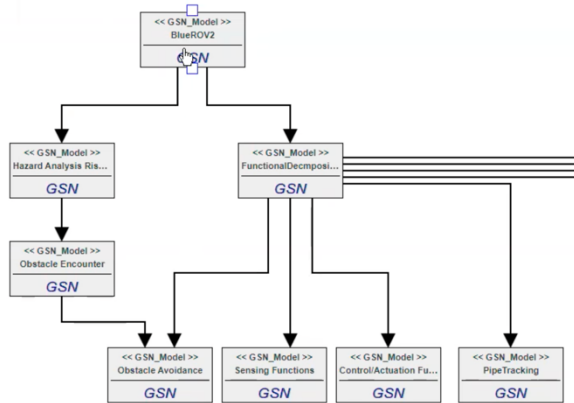


Figure IX.6. Example of a Functional Decomposition Model

Clicking the top function will lead into a list of potential attributes for the functional decomposition model that can have their targets adjusted (Figure IX. 7).

Selection

Context	Target
System Models	BlueROV2_System
Evidences	None
Hazards	None
Functions	None
Requirements	None
Barriers	None
DRMs	None

Figure IX.7. Context Selection Drop Down List

Clicking the ‘System Models’ context row brings the user to a new drop-down list where the ‘Target’ of the ‘Context’ can be altered (Figure IX. 8).

Selection

Target Component	BTree Node
BlueROV2_System\Simulated World\Sensors\Forward Looking Sonar (LaserScan)	
BlueROV2_System\Simulated World\Gazebo Simulation\ThrusterAssembly\Thruster3	
BlueROV2_System\Simulated World\Gazebo Simulation\ThrusterAssembly\Thruster4	
BlueROV2_System\Simulated World\Gazebo Simulation\ThrusterAssembly\Thruster5	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\AA S4 L	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\AA S4 R	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Degradation Tester	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Gazebo world	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Iver FLS	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\INGC failure modes	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Pipeline Generator & Loader	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Pixhawk HW emulation	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Spawn Obstacles	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Thruster failure	
BlueROV2_System\Simulated World\Housekeeping SIM WORLD\Upload BlueROV2	
BlueROV2_System\Simulated World\Sensors	
BlueROV2_System\Simulated World\Sensors\ChemicalConcentration	
BlueROV2_System\Simulated World\Sensors\ChemicalConcentration\SensorCAD	
BlueROV2_System\Simulated World\Sensors\DopplerVelocity	
BlueROV2_System\Simulated World\Sensors\Forward Looking Sonar (LaserScan)	

Figure IX.8. Target Selection Drop Down List

9.3 Fault Trees

Fault trees are graphical models that represent how low-level events, like component faults, combine and propagate to high-level events (like system-wide failures). The combination of the low-level

events is expressed using an AND/OR tree-like structure. The extraction of fault trees, as well as other reliability artifacts, from SysML has been demonstrated as viable means of rapidly building reliability models.

In this tree (Figure IX. 9), primary events are the leaves, intermediate nodes are either logical combiners (disjunctive or conjunctive) or intermediate (e.g., sub-system level) events, and the top (root) node of the tree is the system level event. The arrows connecting the nodes (from leaves towards the root) indicate causation or enablement. The AND/OR operators operate on probabilities of events assigned by the modeler. Fault-trees allow not only the review and logical analysis of how local faults combine and lead to system-level events, but also the calculation of the probability of those events as a function of the probabilities of the low-level events. Thus, they are very useful tools for evaluating the reliability of the system, and in design. For the latter, when the designer changes the system (e.g., by introducing redundancy), the fault tree can help the quantitative evaluation of how the reliability improves (or degrades).

Figure IX. 9 Fault Tree generated by SEAM based on the functional decomposition model, system model and the underlying fault propagation model. Boxes represent top-event or intermediate event in the fault tree. Boxes with a circle below represent the basic event. Loss of Function (LF) events are in blue, Loss of component (LC) events are in brown and Failure mode events are in red. Triangle represents the loss of mitigation function.

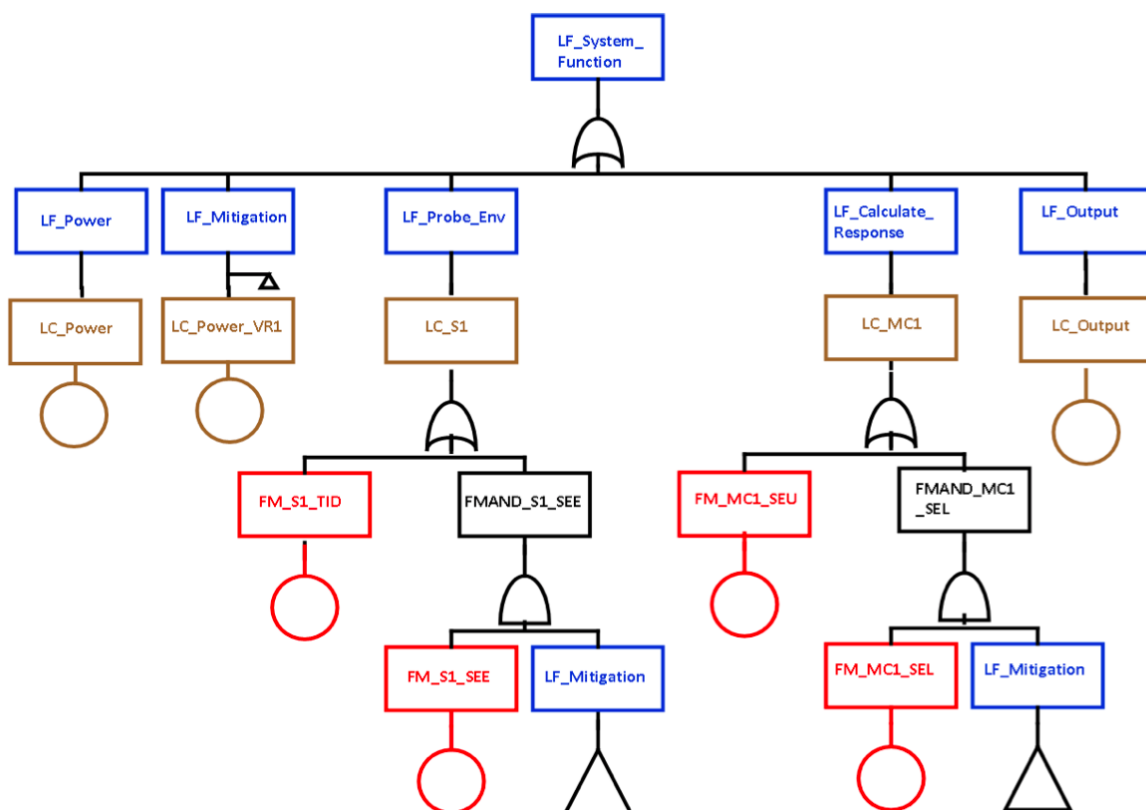


Figure IX.9. Fault Tree generated by SEAM based on the functional decomposition model, system model and the underlying fault propagation model. Boxes represent top-event or intermediate event in the fault tree. Boxes with a circle below represent the basic event. Loss of Function (LF) events are in blue, Loss of component (LC) events are in brown and Failure mode events are in red. Triangle represents the loss of mitigation function.

9.4 References

1. K. L. Ryder et al., "SYSTEMS ENGINEERING AND ASSURANCE MODELING (SEAM): A WEB-BASED SOLUTION FOR INTEGRATED MISSION ASSURANCE," *Facta Universitatis, Series: Electronics and Energetics*, vol. 34, no. 1, Art. no. 1, Feb. 2021.

10 SEAM STANDARD PARTS LIBRARY

The Standard Part Type Library is a SysML library of part types where all potential fault propagations for a given part have been created according to the guidance of radiation effect experts. These part types are also organized according to part family. This library can be utilized by a user to create system models of their own circuits based on their mission environment's radiation parameters. The following families (66 total part types) and radiation concerns are considered in the Standard Part Type Library.

- **Families Present (66 total part types):**
 - Clocks/Timing (4 part types)
 - Digital (5 part types)
 - Discrete (4 part types)
 - Discrete Power (7 part types)
 - Discrete RF (8 part types)
 - Embedded (4 part types)
 - Interfaces (6 part types)
 - Linear (5 part types)
 - Logic (2 part types)
 - Memory (4 part types)
 - Mixed Signal (5 part types)
 - Opto-Electronics (4 part types)
 - Power Hybrid (4 part types)
 - Sensors (4 part types)
- **Radiation Concerns Present:**
 - Single Event Latch-up
 - Single Event Burnout
 - Single Event Transient
 - Single Event Function Interrupt
 - Single Event Gate-Rupture
 - Single Event Upset
 - Multiple Bit Upset
 - Total Ionizing Dose
 - Displacement Damage Dose

Figure X.1 Families and Radiation concerns in Standard Parts Template

10.1 Instructions for using the Standard Part Type Library

In order to incorporate the Standard Part Type Library into SEAM, the user starts with creating a new project in the Model Based System Assurance website. In Figure X. 1, we have created a new project

labeled 'test.'

The screenshot shows the SEAM Projects interface. At the top, there is a header with the 'Projects' logo and a user profile for 'admin'. Below the header, there is a navigation bar with tabs for 'ALL', 'A - E', 'F - J', 'K - O', 'P - T', and 'U - Z'. The main content area displays a table of projects with columns for 'Owner', 'Name', 'Info', and 'Last Modified'. The table lists various projects, including 'europa_copy', 'example1', 'Library', 'meta_w_library', 'patTest', 'rgentic_dup_copy_1_5_2021', 'SEAM_Template', 'startrekft', 'TCLoop_Example', 'Template_Parts_Library', 'Use_Template_Models_1', 'meta', 'meta1_0_13a', 'RnMTemplate', 'Tutorial', and 'RnMTemplate' (owned by 'archive_11_28_18').

Below the table, there is a 'Create New...' button and a 'Close' button. The 'Create New...' button is highlighted, and a dialog box is open. The dialog box has a dropdown menu for 'admin' and a text input field containing 'test'. Below the input field, there are 'Create' and 'Cancel' buttons.

Owner	Name	Info	Last Modified
admin	europa_copy	describe the project	20 days ago
admin	example1	describe the project	2 months ago
admin	Library	describe the project	2 months ago
admin	meta_w_library	describe the project	2 months ago
admin	patTest	describe the project	3 years ago
admin	rgentic_dup_copy_1_5_2021	describe the project	6 months ago
admin	SEAM_Template	describe the project	2 minutes ago
admin	startrekft	describe the project	a year ago
admin	TCLoop_Example	describe the project	44 minutes ago
admin	Template_Parts_Library	describe the project	2 months ago
admin	Use_Template_Models_1	describe the project	5 days ago
archive	meta	describe the project	4 years ago
archive	meta1_0_13a	describe the project	3 years ago
archive	RnMTemplate	describe the project	3 years ago
archive	Tutorial	describe the project	4 years ago
archive_11_28_18	RnMTemplate	describe the project	3 years ago

Figure X.1 New project with parts template library

After creating the project, the user adds the Standard Part Type Library directly into the new project

by duplicating the project labeled 'admin/SEAM_Template,' shown in Figure X. 2.

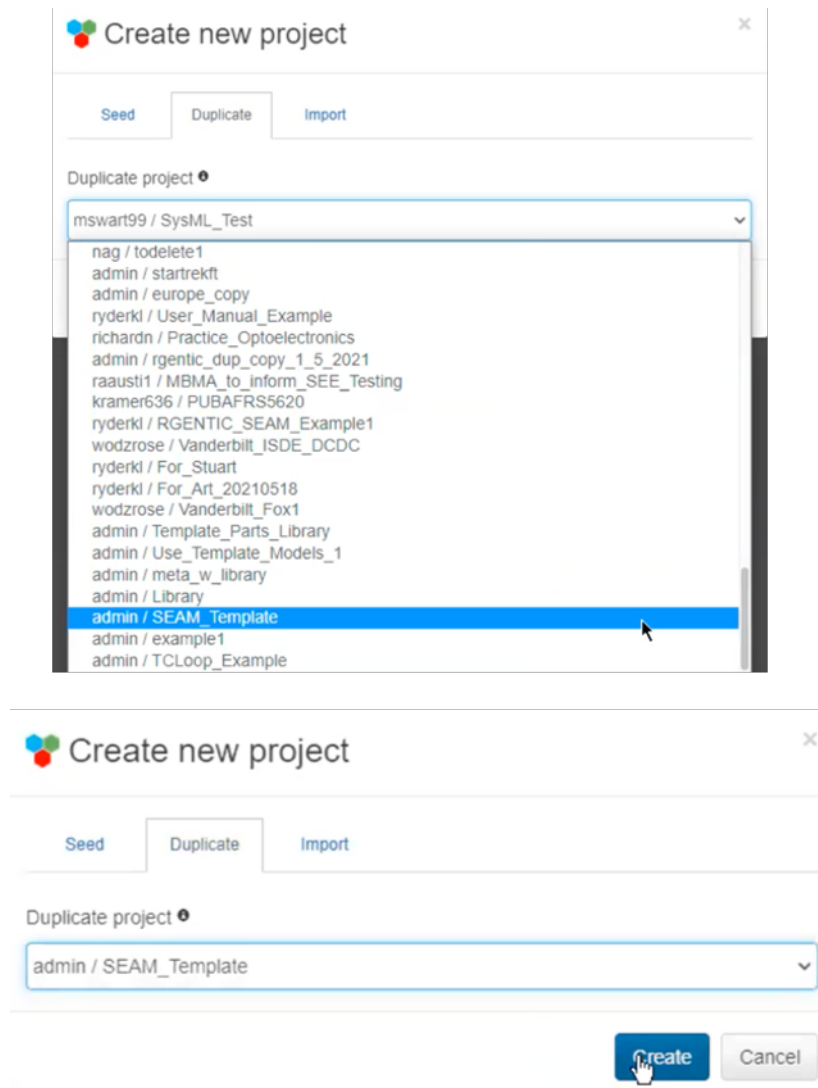


Figure X.2 New project with SEAM template

At this point, it is possible that the Standard Part Type Library needs to be updated based on when the last time 'admin/SEAM_Template' was updated. If this is the case, a light blue rectangle will appear on the top right of the screen. Clicking this box will ensure the Standard Part Type Library is updated

based on the latest reviews by radiation effects experts (Figure X. 3).

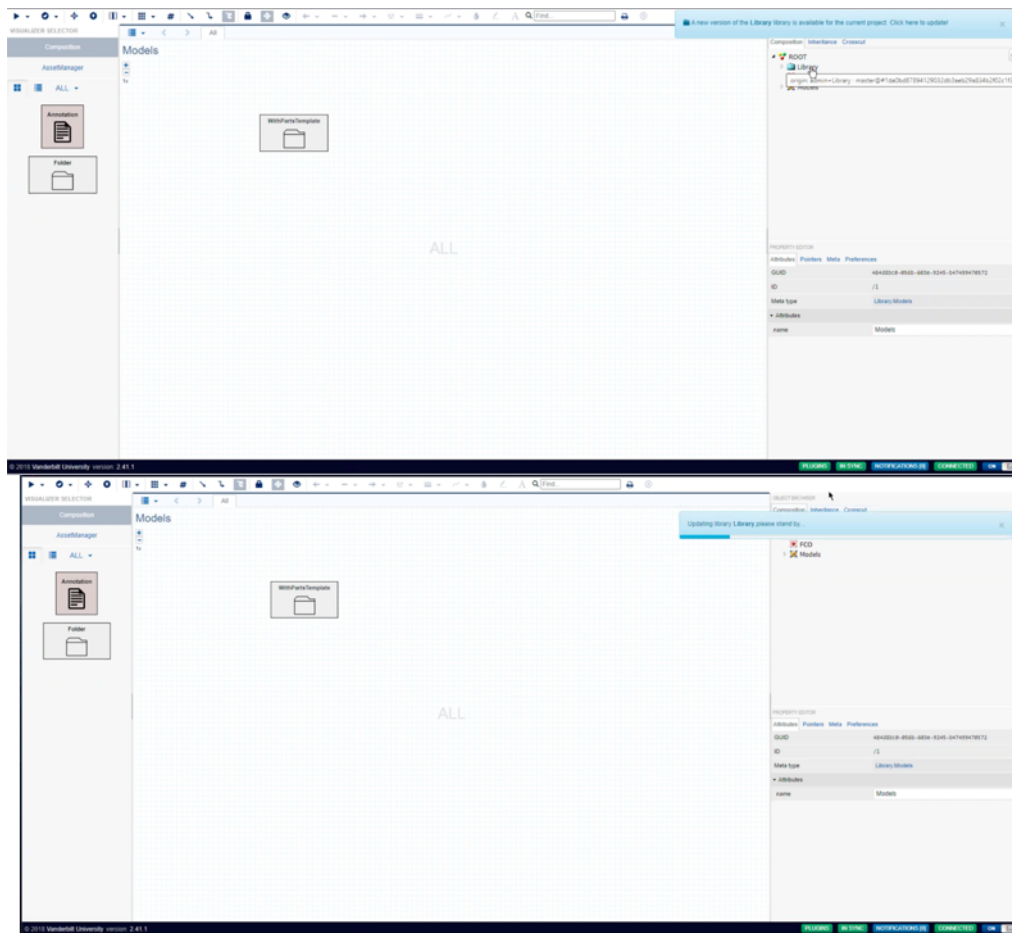


Figure X. 3. Updating the Golden Part Library (only observable if library not in the most recent version)

On the right side of the workspace, the user can click 'Library' to access the Standard Part Type Library. Clicking 'Library.Definitions' leads to models for both failure labels in 'Library.FailureLabels' and for

X. 5).

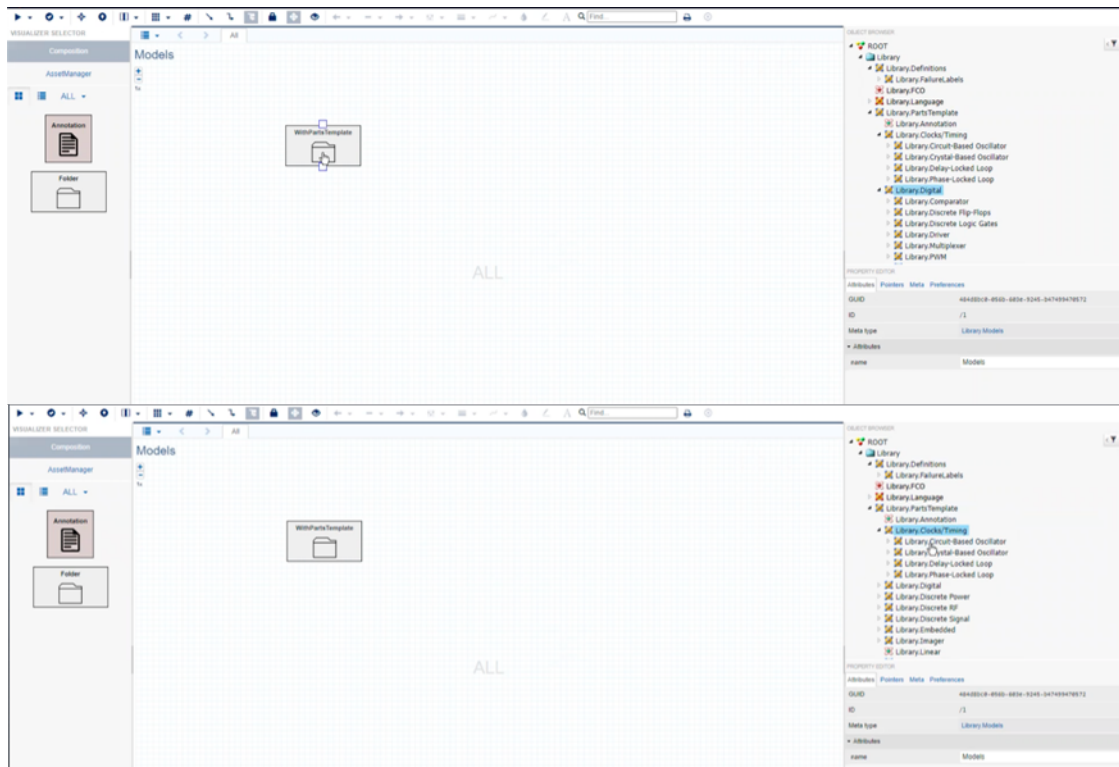


Figure X. 5. Opening ‘Digital Devices’ Family and ‘Clocks/Timing’ Family

Entering ‘WithPartsTemplate’ will allow the user to begin creating a Library, Function Model, and System Model for their desired circuit. Once inside this folder, the user can drag and drop a Library from the right and call it ‘Parts_Library.’ This will serve as a repository for all the user’s desired part instances from the Golden Part Type Family. At this point, the user can also drag and drop the ‘System_Model’ folder, which will serve as the place for all schematic construction using the part in-

stances in the 'Parts_Library' folder. Both are shown in Figure X. 6.

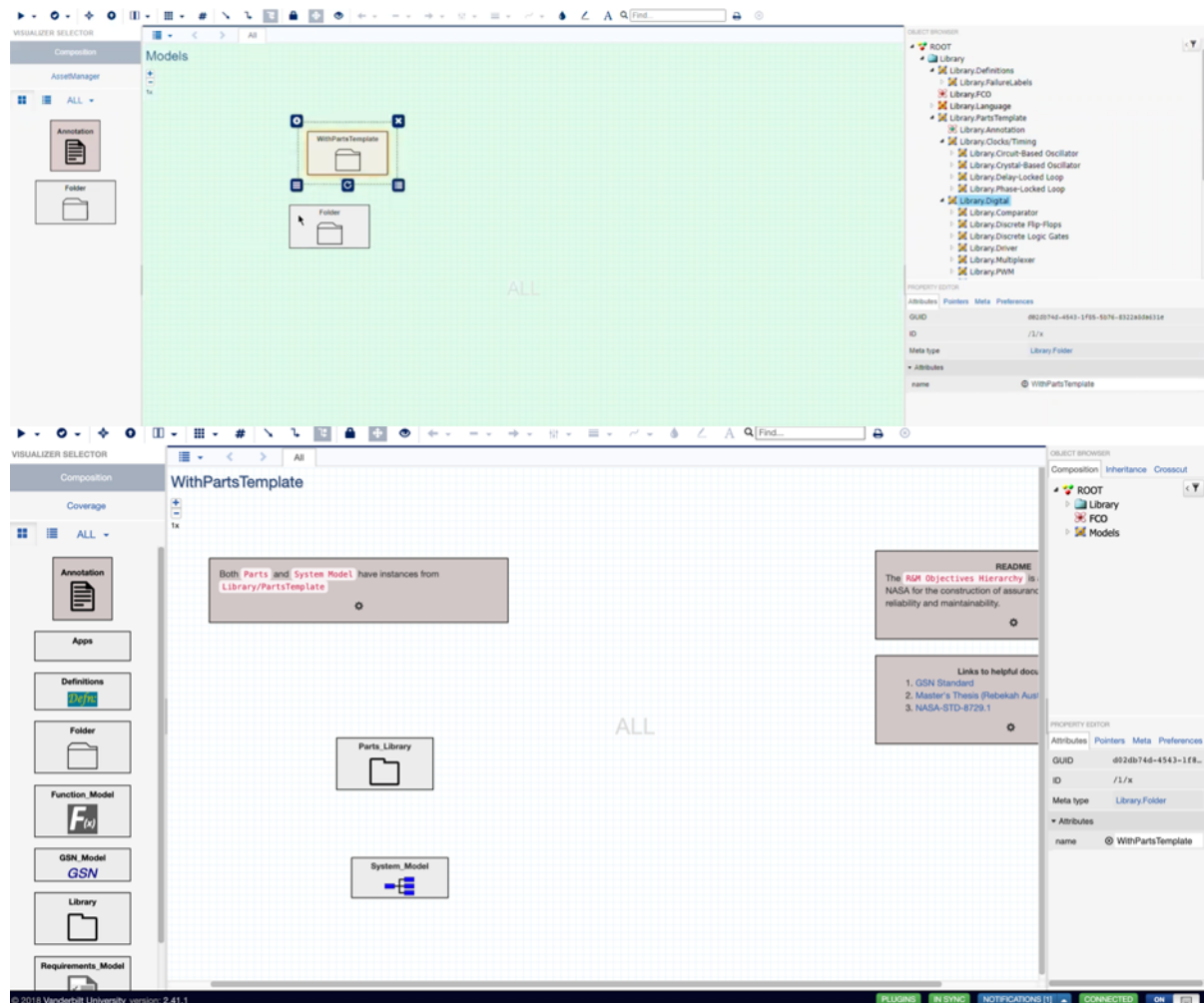


Figure X. 6. Creating an appropriate area to contain the user's preference of part templates

Once inside the 'Parts_Library' folder, the user can drag instances of part templates into this location. By holding the 'control' button, the user can drag and drop an instance from the Golden Part Library onto the workspace. In this example's case, we dragged and dropped an FPGA block and a Microcontroller block. If a pop-up box appears and asks whether to 'Create instance here' or 'Copy here,' choose 'Create instance here.' Overall, creating instances of part types allows the user to modify the part type template within their project. Specifically, it will maintain a link to the user's library model found in the 'Parts_Library' folder. Another benefit is that updates to the Standard Part Type Library by radiation experts will continue to be carried over into the user's specific part library in the 'Parts_Library' folder. One final note is that inside an instance's part template (which can be accessed by double clicking the instance part template), we can adjust radiation effects but not delete them.

These steps can be visually seen in Figure X. 7.

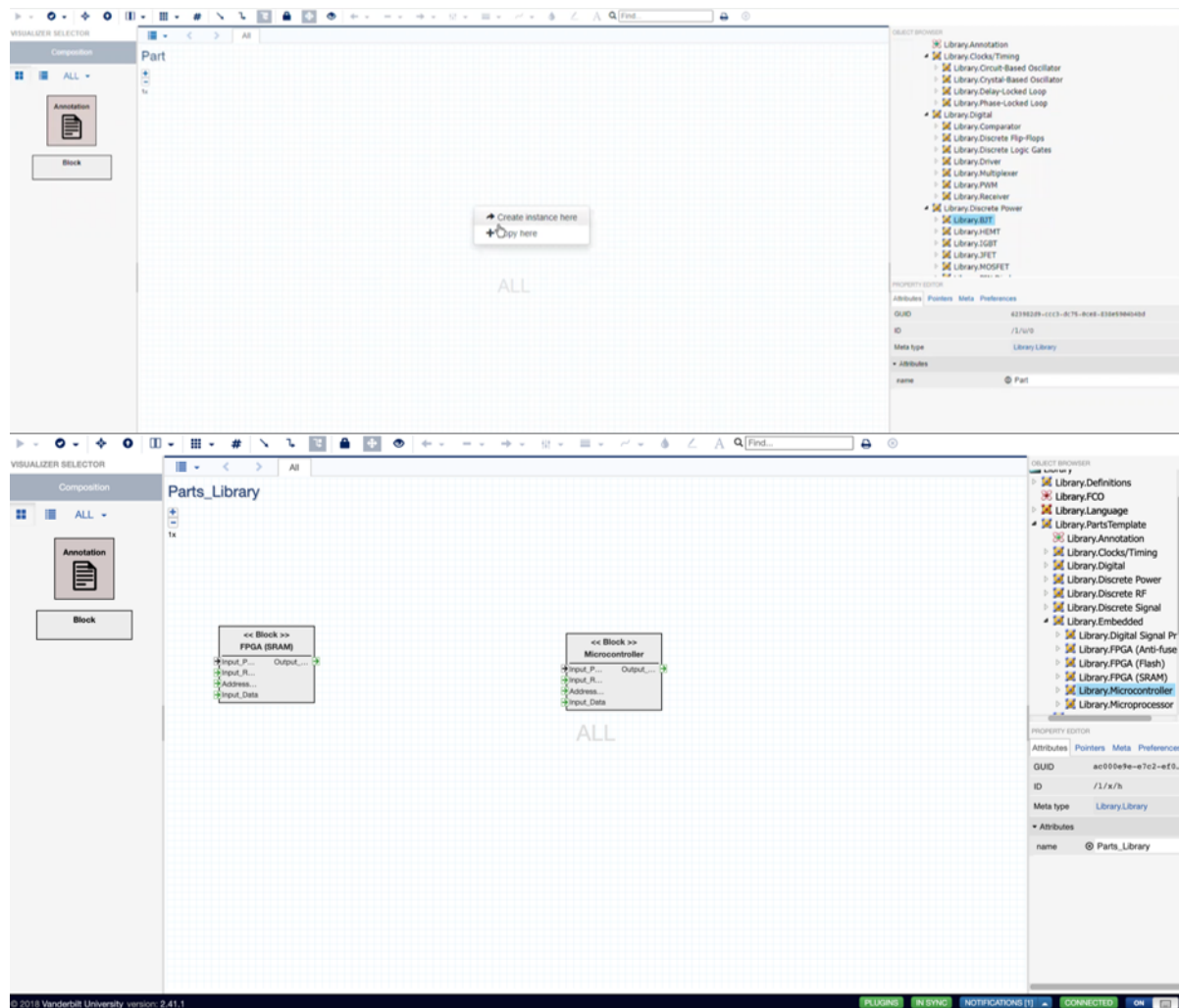


Figure X. 7. Creating instances from Golden Part Library

Now that we have created a parts library within the 'Parts_Library' folder with relevant instances, the user can create a system model. The user can exit the 'Parts_Library' folder by clicking the 'upward-directed arrow' symbol on the upper left of the screen. The user can then split the screen into two vertical screens by clicking the top left icon that looks like a book. This feature is useful for looking at a system model while also looking at our instance part library, all actions can be observed in Figure

X. 8.

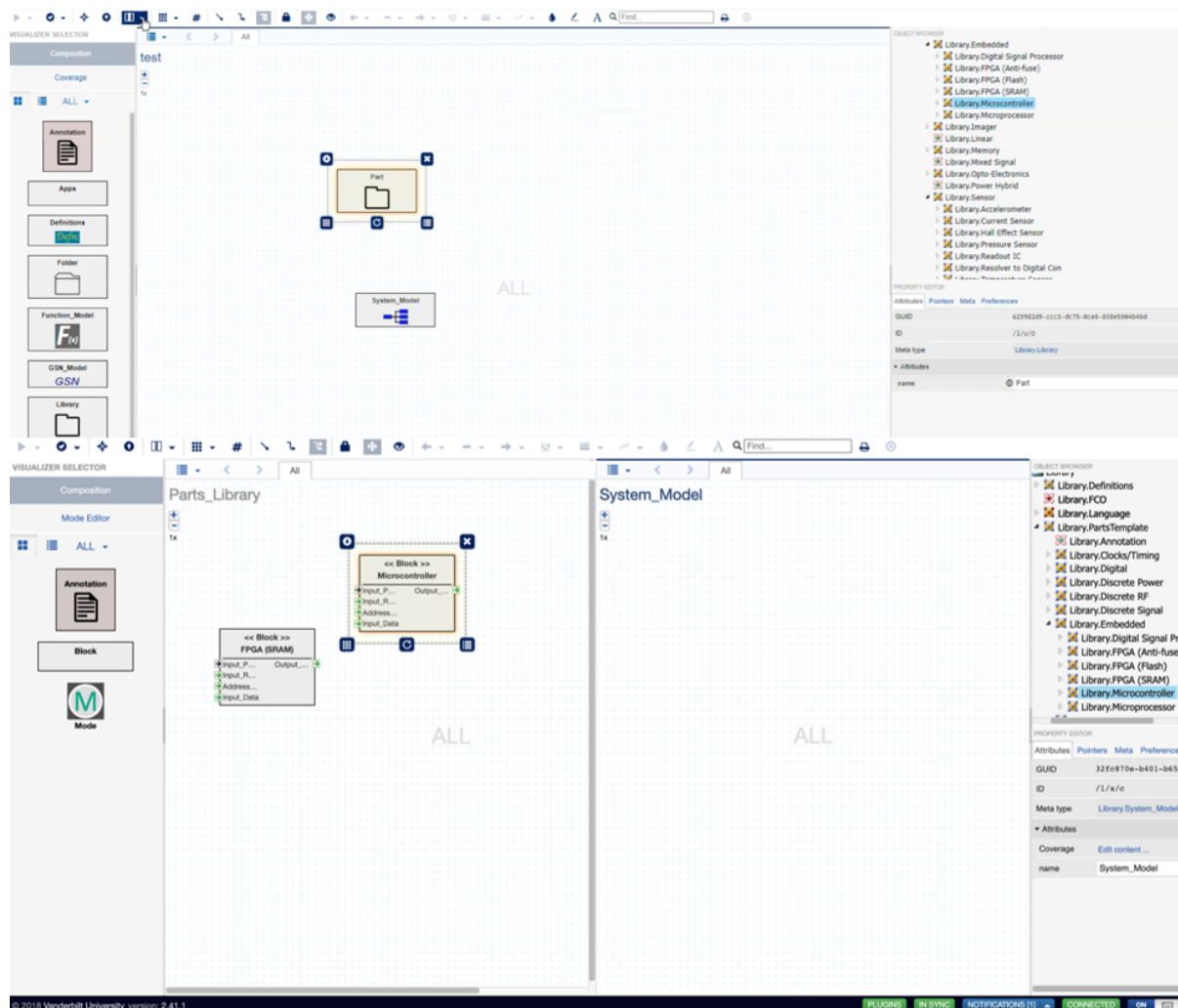


Figure X. 8. Splitting the screen into two vertical screens

With both the part library and the system model folder open, the user can begin creating a system failure propagation model for their circuit. To accomplish this, the user will need to hold the alt key and drag a part instance from the part folder on the left into their system model on the right. Multiple instances of the same part can be created in this manner, and the names of each instance can be changed by going into the attributes of a specific instance. This attributes section can be found on the bottom right of the screen when clicking an instance (after it is in the 'System_Model' folder). The

user can then alter the name (Figure X. 9).

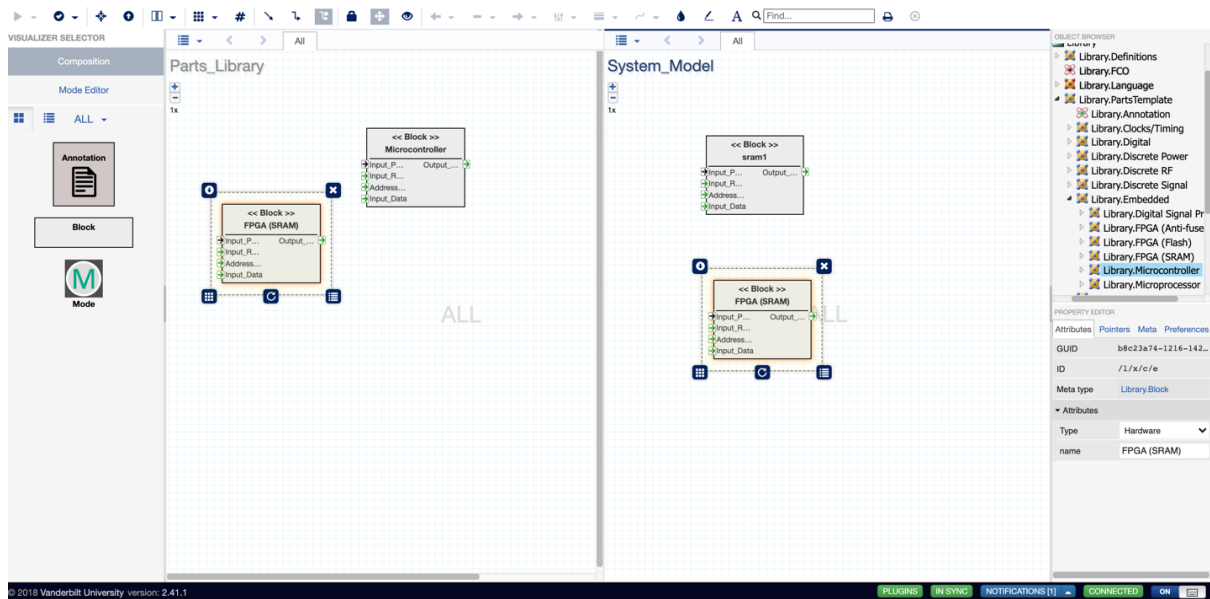


Figure X. 9. Creating instances in the ‘System_Model’ folder and changing the name of each instance in the attributes section

Blocks can then be connected to each other in a way analogous to wiring electronic components together. However, the user must first decide upon which ports to connect to each other. It should be done according to the circuit’s system architecture. Connections cannot be made between blocks in the ‘Parts_Library.’ The ‘Parts_Library’ will not allow those connections to occur (Figure X. 10).

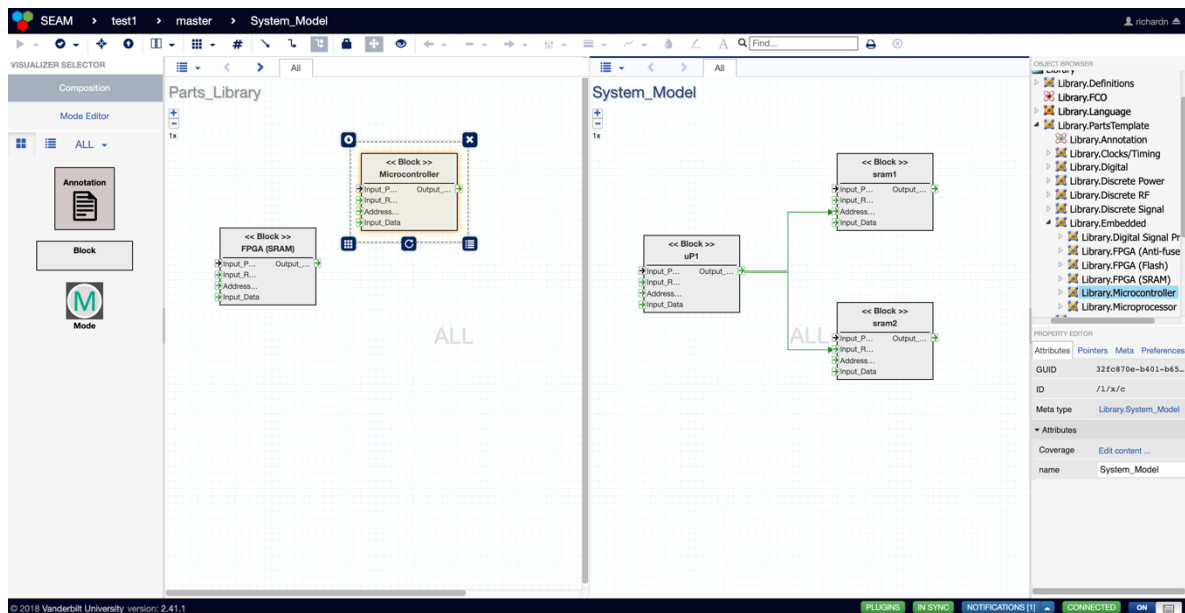


Figure X. 10. Wiring the part instances to each other via the ports

As mentioned previously, the SysML faults can also be accessed within each part instance by double clicking the block (Figure X. 11). This provides the user a means of understanding how the faults

propagate through that specific part instance (from a given input to its output).

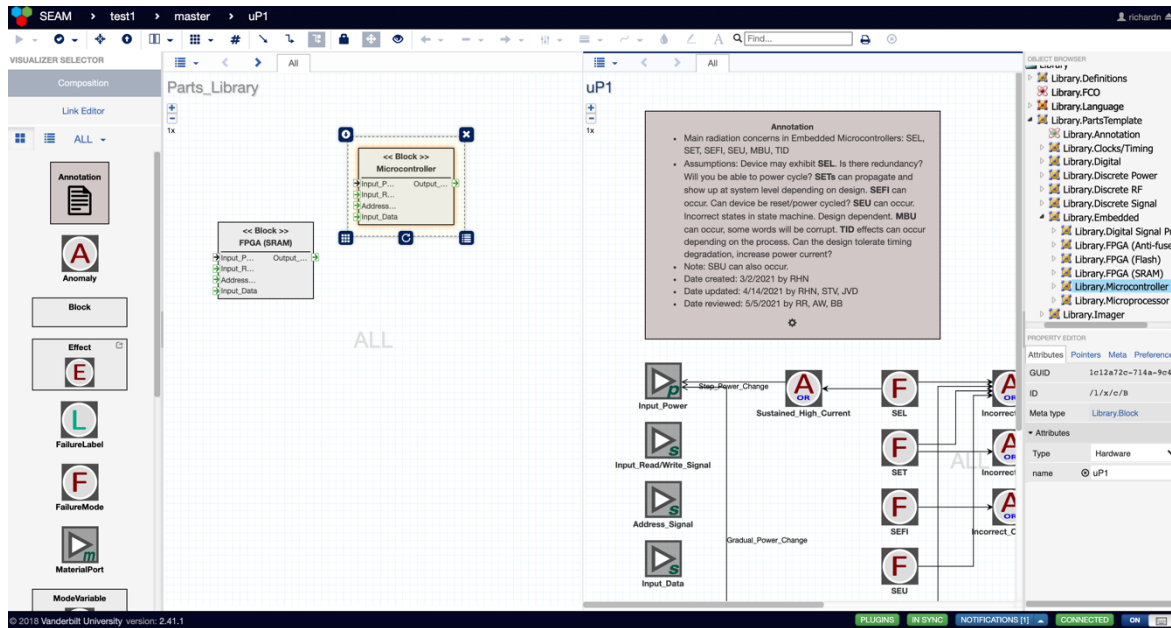


Figure X. 11. Observing how fault propagation occurs through a part instance’s input to its output

The line placement of error labels can be adjusted by clicking the icon that shows three lines with filled-in boxes on them (on the top row). Clicking this icon allows the user to readjust the error labels into three different positions: left-side, center, or right-side (Figure X. 12)

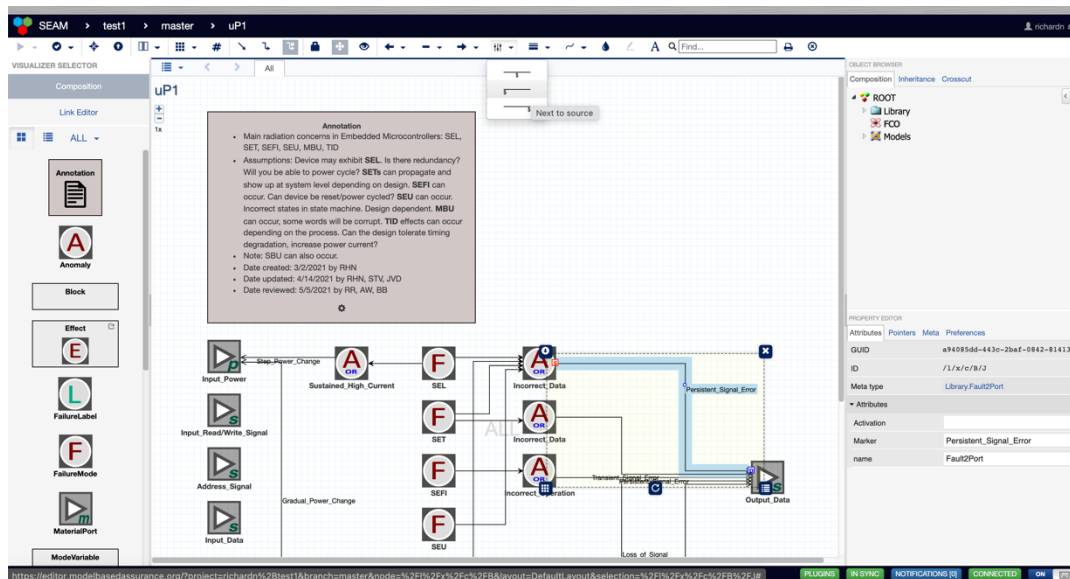


Figure X. 12. Adjusting source label placement

The routing of lines can be altered in three distinct ways. Clicking the ‘straight arrow’ icon results in all lines becoming straight lines. Clicking the ‘straight arrow with 90° edges’ icon results in all arrows bending at 90° edges to best reach its destination. Clicking the ‘straight arrow with 90° edges and filled box in upper right’ icon allows the user to determine the location of those 90° edges. Specifically, the user will need to double click at a particular part of the line and drag that point to the

user-degired location (Figure X. 13)

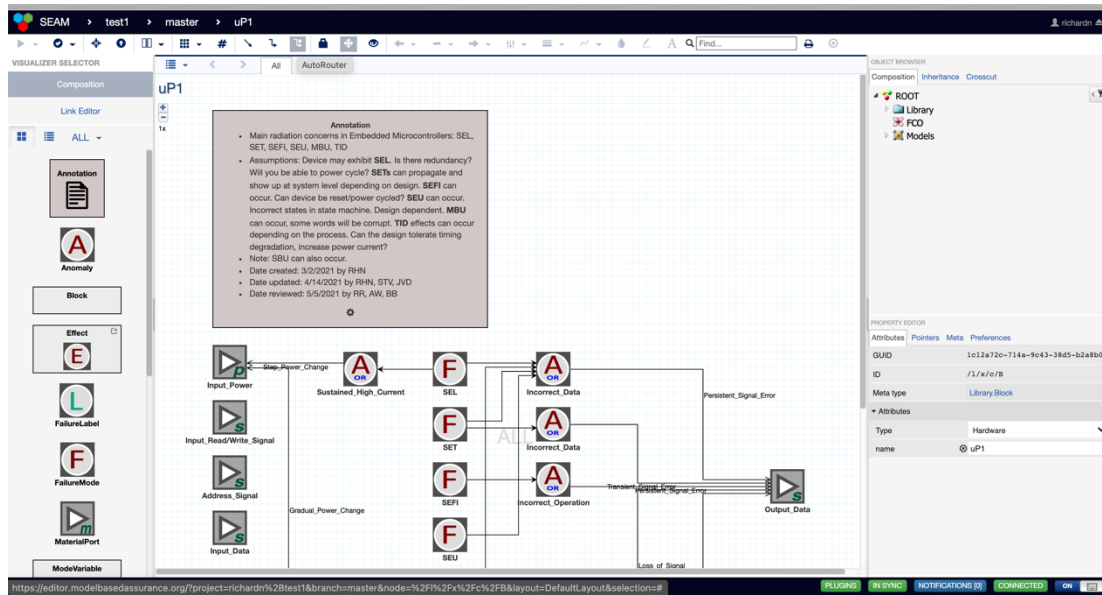


Figure X. 13. Adjusting router of line

To create a function model, the user can leave the split screen mode, and drag and drop a 'Failure_Model' library from the left-hand side onto the screen (Figure X. 14).

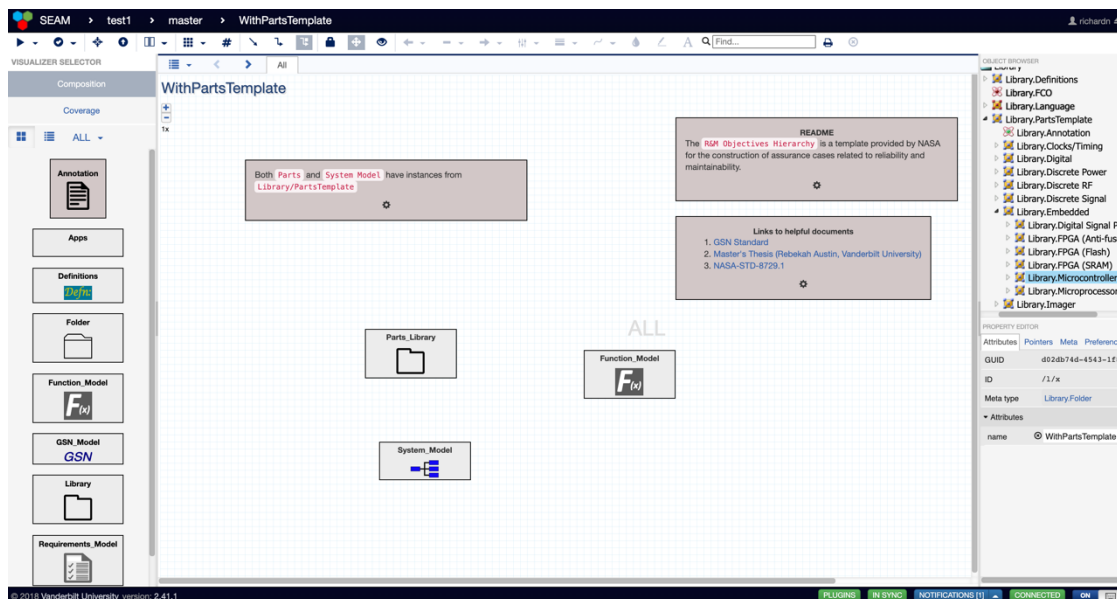


Figure X. 14. Creating a Function Model based on the Golden Part Family Template

The user can double click inside the 'Failure_Model' folder and drag and drop Top Functions and Primitive Functions onto the screen (Figure X. 15). These functions show how functions can potentially fail

due to radiation effects. These failure modes can be connected to each other via the ports.

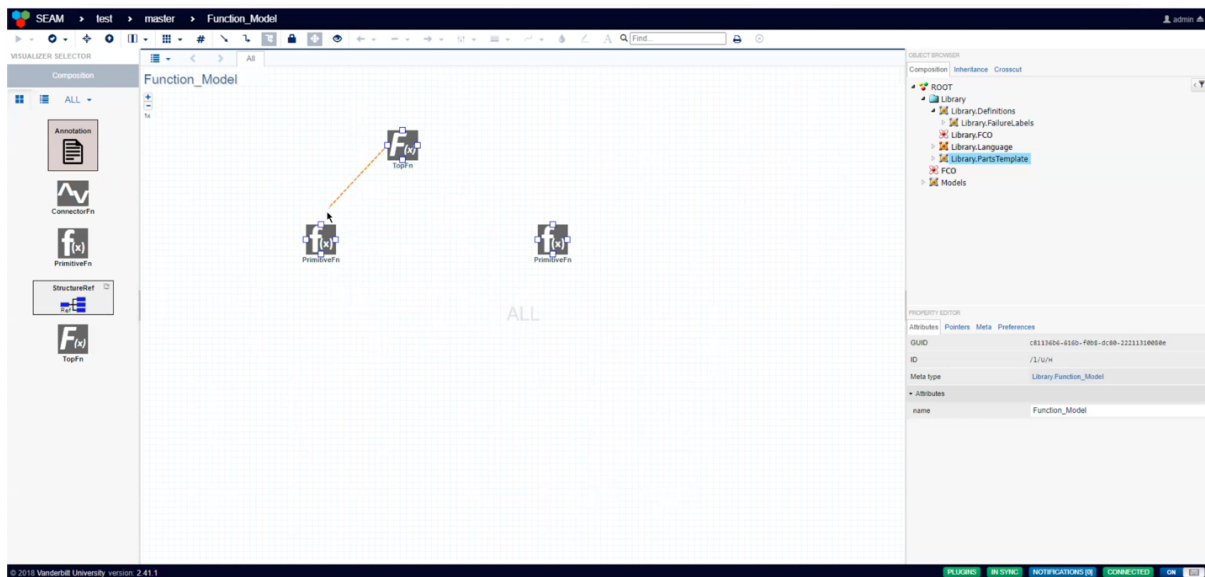


Figure X.15. Creating and connecting functions

If the user desires to reference a system model in the function model, the user can return to the split screen mode and have the system model folder open on the right side of the screen. Then, the user can hold the shift key and drag blocks from the system model onto the function model screen (Figure

X. 16).

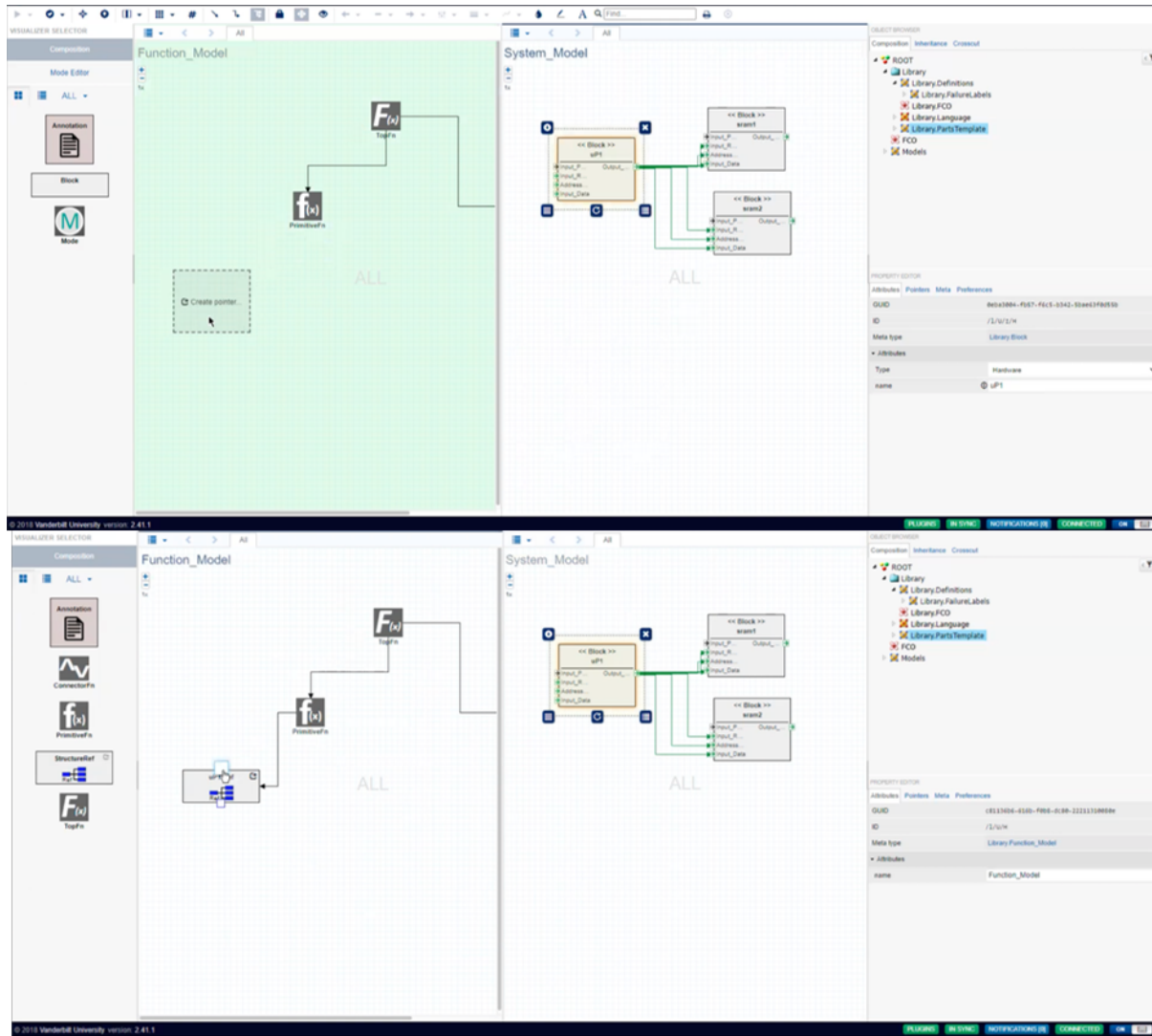


Figure X. 16. Creating system model references in the function model