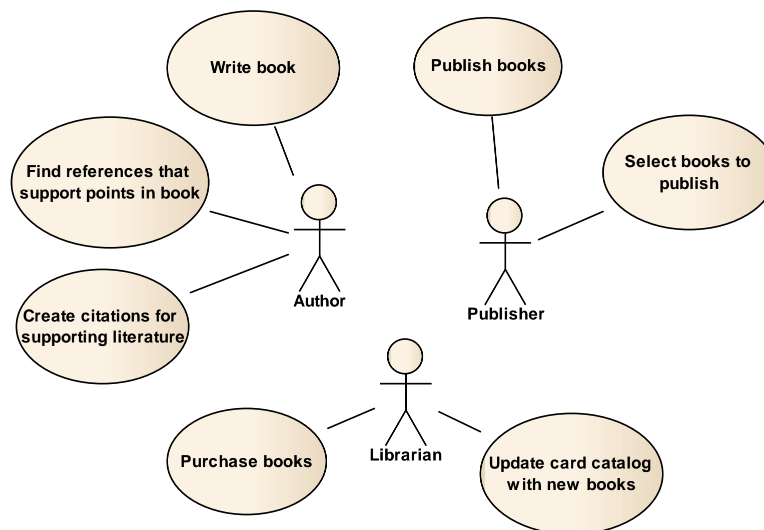
Introduction

The INCOSE MBSE Usability Group voted on the most important systems usability issues to investigate at the January 2011 working group. The working group selected the library use case. This paper begins by understanding the value of a brick and mortar library. Brick and mortar library concepts are then used to understand why the library concept is important and how to take the best ideas of a brick and mortar library into a Model-Based Systems Engineering (MBSE) environment. Finally, concepts essential to implementing a library use case in tools and languages are listed.

Brick and Mortar Library Example

To understand MBSE libraries, it is helpful to think of the original definition of a library. Webster's Student Dictionary of 1938 defines a library as a room or building given over to a collection of books kept for use and not for sale; also an institution for managing such a collection. This definition was true in 1938 and is still true today. The beauty of a library is its simplicity as an institution. This section develops use cases for the library. Use cases described in this paper are simplified. For illustration, consider a student in 1938 that needs to write a term paper for a school assignment. Actors and use cases are depicted below:

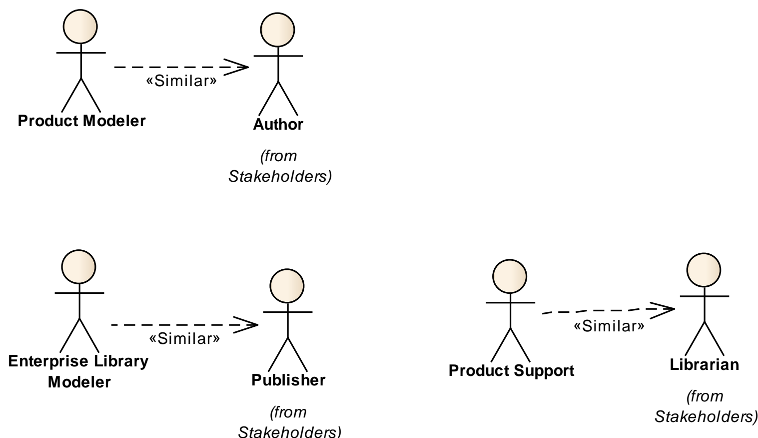


The author needs to write a book, find quality relevant literature, and finally to support the book theme with literature citations. A publisher reviews book manuscripts to find publishable literature. Publishers also fix any quality issues and publish the literature. Lastly, the librarian needs to find and acquire books. The librarian helps the authors find supporting literature. This help may be direct discussion or indirect help via aids such as card catalogs.

The library model is simple to understand and to use. The public finds value in libraries and continues to support and sponsor new libraries. Perhaps this is why the INCOSE MBSE Usability community identified library use cases as one of the top four use cases of interest.

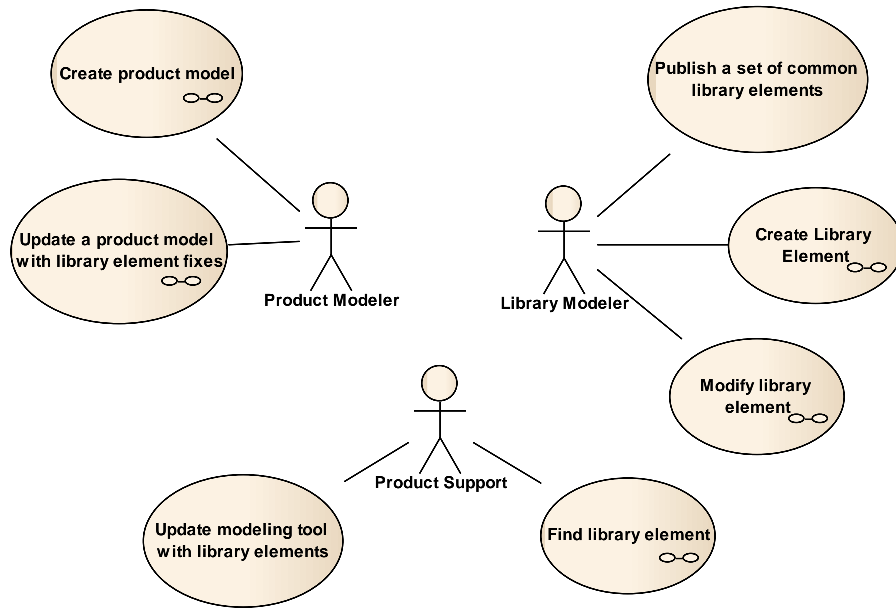
Primary use cases of MBSE libraries

For a model, Webster's definition of a library can be extended into a model library. A MBSE library is as a set of language, process, and tools needed for an enterprise to create, maintain, organize and use a collection of common foundational model library elements. For this new type of library, the product modeler is similar to an author. They both use a language to create a final product using references to support the primary thesis or model purpose. The enterprise library modeler is similar to the publisher. They both are concerned about identifying and making quality product for end customers. The enterprise library modeler does this by converting product requirements and existing model elements into a library element that supports variants for multiple products in the enterprise. Finally, the product support is similar to the librarian. Both are interested in organizing products to make them easy to access. The product support does this by updating modeling tools with the latest and greatest library elements.



The focus of this paper is on the product modeler and on how the product modeler creates a product model. Other use cases will be considered in future updates to this paper.

Actors and use cases for a MBSE library ecosystem are similar to the use cases found in the brick and mortar library. They include:



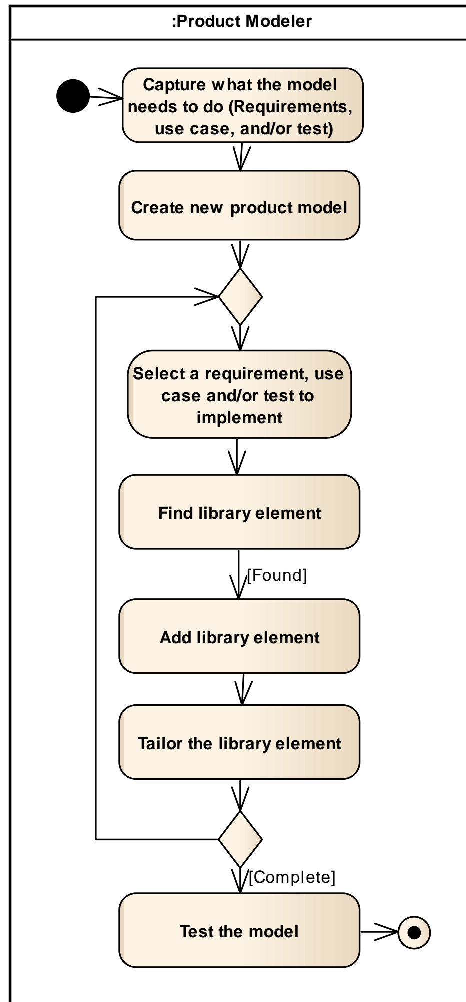
The product modeler use cases

A product modeler builds a behavior or structure model of the product to meet a modeling objective. Example modeling objectives may include items such as generating target software, determining the correct number of switches in a network, or simulating the behaviors of a system prototype for the end customer.

Create product model

Building a product model starts with understanding what the model needs to do and ends with a set of results. As an example, consider developing a simple model that tests the behavior of a low-pass filter. The example is built using Activity Diagrams in the SysML modeling language. Two concepts, tailoring and variants are introduced to describe desired capabilities. A variant is a library element parameter that can change the form or function of the library element as needed. Tailoring is the process needed to customize the library element for a specific use in the product model.

The product modeler follows the actions shown below:



The **pre-condition** for this use case is a list of library elements that is available and ready to use.

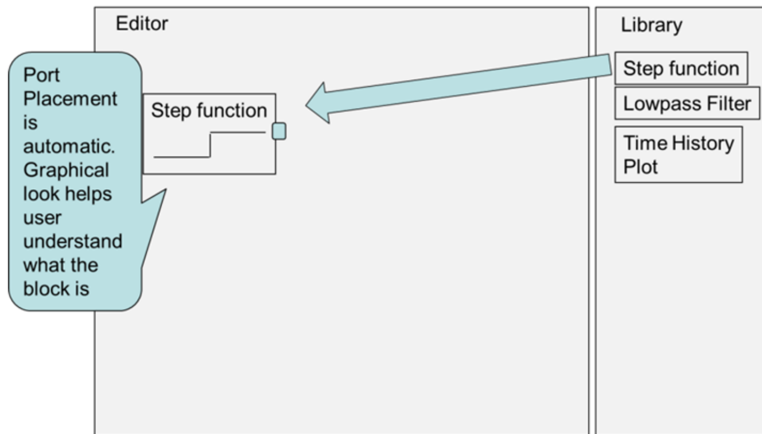
Action details (STEPS):

1. **Capture what the model needs to do (Requirements, use case, and/or test)** – Product models are built for a purpose. The requirements, use cases, and/or tests capture this purpose. For the purpose of this example, the test is to insert a step function into a low pass filter. Verify that changing the low pass filter variant, Tau, produces the correct rise time.
2. **Find library element** - In this example, the Step Function, low pass, and Time History Plot library elements are found in the list of all library elements. The library elements are implemented as SysML activities. The activities contain all of the behaviors needed to execute the model.
    a. Note: Users will often use 20% of the library elements for 80% of the model.
    b. An effective tool minimizes the time it takes to find and understand library elements. Understanding comes from both an example and a description that includes the environmental constraints the library element requirements, and the realized tests.

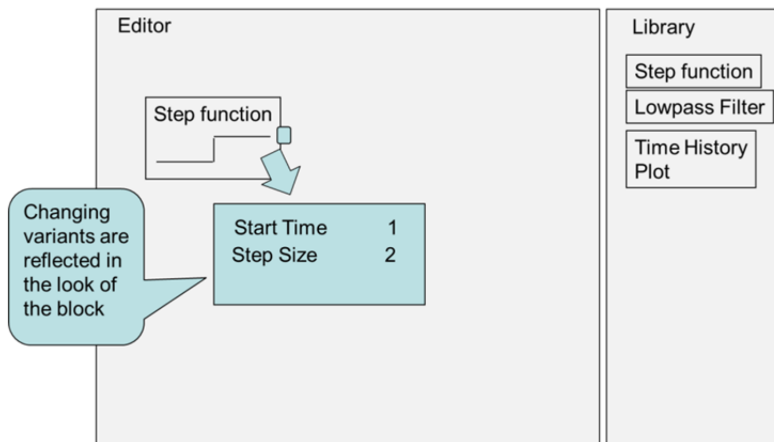
3. **Add library element** - Create an instance of the library element in the editor. In this example, the library elements listed in STEP 1 become SysML Call Actions. These call actions call the library element activity. The size of the call action and the placement and visibility of input and output pins is defined by the sizes and placement of the pins on the library element activity. The icon for the call action is defined by the library element. The icon graphics and text reflect the default tailoring values of parameters defined in the library element.
    a. An effective tool minimizes the number of clicks, minimizes the distance of cursor movement and minimizes the screen real estate needed to add the 20% most common library elements to the editor (STEP 1 and STEP 2).
    b. More than one instance of the step function can be created from the same library element.
4. **Tailor the library element** – Set the variant to a value needed by the product model. Tailoring changes the default value library element variant A variant is a property of the instance that can be changed from the default value. In this example we set the step function instance by setting the step insertion time at 1 second and the step size at a value of 1.
    a. Internal object node values can be tailored. For example, a gain library element multiplies an input by a constant value and sends the result to the output. A object node in the activity holds the value of the constant. The default value of the object node is tailored for each specific instance.
    b. Visible input, output, and input/output pins can be tailored.
    c. The default values of input, output, and input/output values can be tailored.
    d. Text in the icon reflects the tailoring. For example, if the gain value of a gain block is set to 3.5, the text on the icon may say “Gain = 3.5”.
5. **Test the model** – The test involves executing the model, looking at the time-history plot, and verifying that the time constant Tau produces the correct rise time.

To illustrate this use case, develop a simulation that tests a low-pass filter. Inject a step function into the low-pass filter and view a time-history plot of the result. The use case example is detailed below:

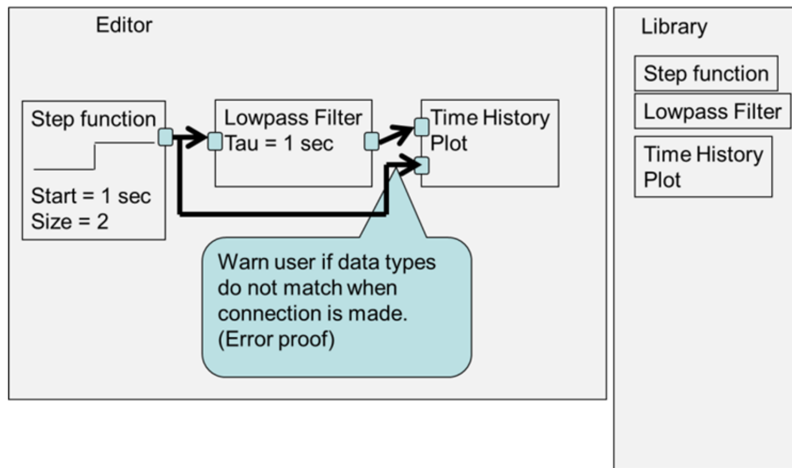
1. **Find library element** - Begin by finding the step function in the library.
2. **Add library element** - Place the step function in the editor. The step function has one output and no inputs. The pin placement is visible and is located in the same place as the parameter on the library element activity. The icon draws a picture of a step function and scales the picture to mirror the default value of 1 for the step input and 1 for the step size. See the diagram below:



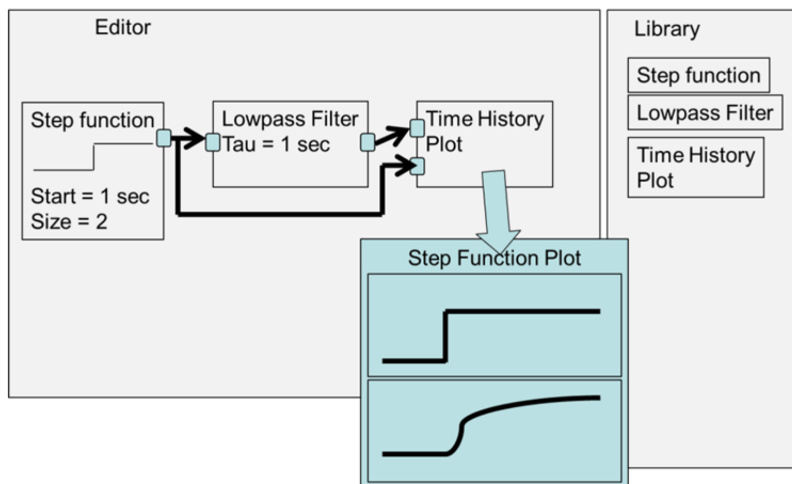
3. **Tailor the library element** – The step function is next tailored. The step is inserted at a time of 1 second and the size of the step is changed to 2 from 1. The Icon adjusts the vertical height of the step to 2. See the diagram below.



4. Repeat steps 1, 2, and 3 above to add the low-pass filter and the time-history plot.
5. Complete implementing the product by connecting the pins located on the actions by using an object flow. See the diagram below. When each connection is made, the data types are checked for compatibility and the object flow is checked for algebraic loops. An error prompts the user to fix any problems as the connections are made. Note that data types include any specified meta data such as the pin/parameter implementation type, the units, the flow port direction. If an error is discovered, the user is notified as the connection is made and suggested solutions are proposed.



6. **Test the model** – The actions are then executed. The time-history plot of the results is viewed as shown in the following diagram.



Update a product model with library element fixes

Library elements will change over time. Changes may be driven by events such as feature enhancements, dependencies, or bug fixes. The product modeler needs a way to update the product model with changes. It is possible that changes to the library element will require fixes to the product model that use the library element. A product modeler may also choose to not accept changes to a library element. This may happen if the change is an enhancement to the library element that is not needed by the product.

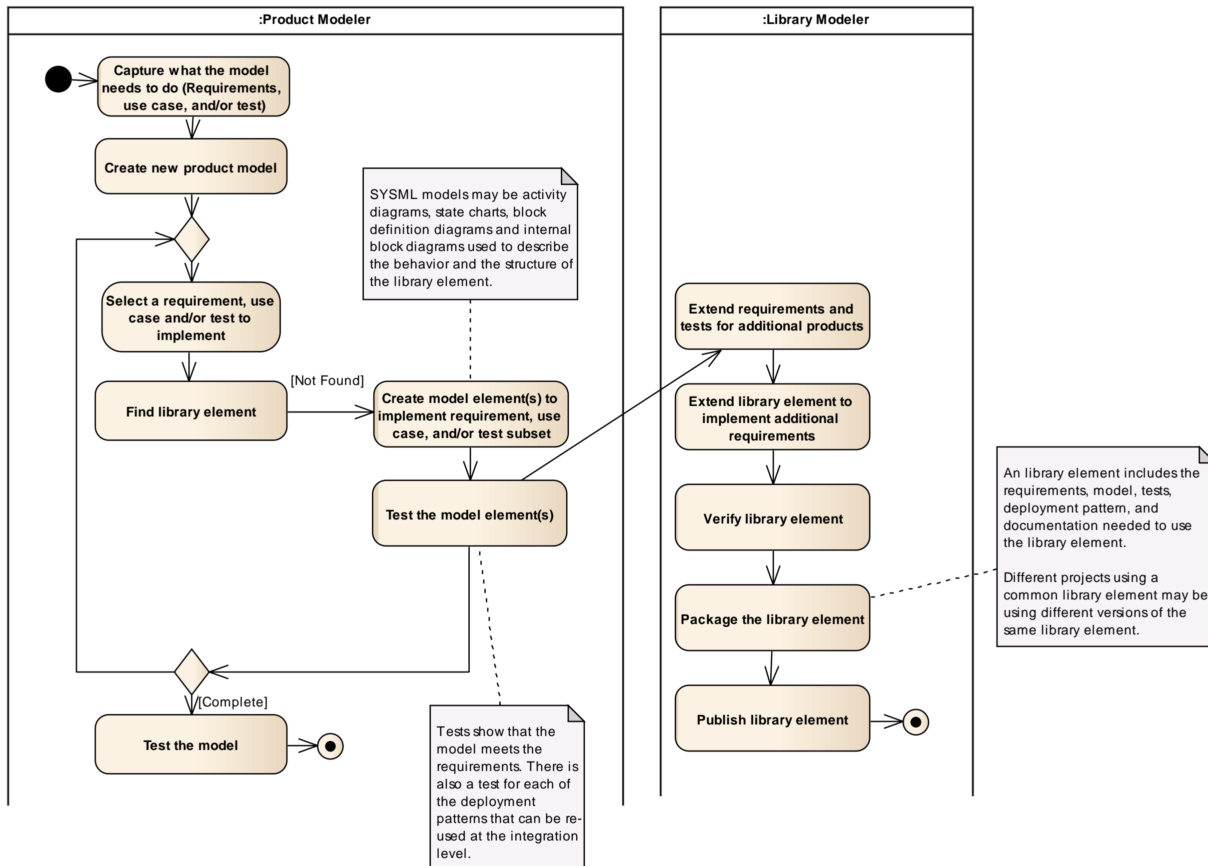
Publish a set of common library elements

Common library elements only have value if they can be found and maintained over time. Publishing common library elements makes changes visible and notifies the product team (product support and product modelers). Product teams have the option to update to the new library elements. To make this

decision, the product teams need to understand the differences between the version of the library element they are using and the new published version.

Create library element

Creating library elements starts with creating a product model. Some of the model elements used to create the product model is usable across multiple products in a product line, across all models in an enterprise or across all models in a modeling discipline. The use case details are shown below:



The **pre-condition** for this use case is a product modeler is unable to find a library element that implements a set of requirements. (Note that I included additional product modeler action steps for clarity)

Action details (STEPS):

1. **Create model element(s) to implement requirement, use case, and/or test subset** –The product modeler uses a set of product requirements to create an element. The product modeler is focused on how to make the model element work for the product, not necessarily for how to make it work for the broader modeling community.
2. **Test the model element(s)** – The product modeler verifies that the library element implements the product requirement and use cases.

3. **Extend requirements and tests for additional products** – The library modeler starts with the product modeler use cases and requirements. The library modeler understands the needs of additional product modelers and extends the use cases and requirements as desired by this community. The new changes may involve adding additional features, enabling variants, or extending the type of environments the model works in. When consensus is reached by the community on the library element use cases and requirements, the library modeler creates tests for library model verification.
4. **Extend library element to implement additional requirements** – The library modeler implements the changes desired by the community.
5. **Verify library element** – The library modeler verifies that the library element implements the requirements.
6. **Package the library element** – The library modeler takes the model element and models or creates sample deployment patterns. A deployment pattern captures the dependencies of the library element on other library elements, adds variants to enable tailoring, and documents examples of how the library element can be used. This involves preparing the requirements, deployment patterns, documentation, test and test results in a way that is shareable across a large group of people.
7. **Publish library element** - After the library element is verified, the library modeler then packages the library element. Finally, the library element is published. A published library element is a library element that is advertised to different products, findable by the products when they need it, and usable in the product models. See the diagram below:

To illustrate this use case, consider the low pass filter example. The use case example is detailed below:

1. **Create model element(s) to implement requirement, use case, and/or test subset** – The Library Modeler captures additional product requirements. For example, products want to vary the frequency the low pass filter is called at (environment) and want to limit the maximum and minimum integrator size within the low pass filter (feature). The library element is updated to enable this variant, and is re-tested. The library modeler records the environment constraint that the filter must be called at a constant periodic rate (design dependency) and that the filter depends upon an additional library element called MaxMinLimit. The artifacts, such as the requirements, use cases, example usage, design, tests, and test results are packaged and then published.
2. **Test the model element(s)**
3. **Extend requirements and tests for additional products** – The Library Modeler captures additional product requirements. For example, products want to vary the frequency the low pass filter is called at (environment) and want to limit the maximum and minimum integrator size within the low pass filter (feature).
4. **Extend library element to implement additional requirements** – The library element is updated to enable this variant
5. **Verify library element** – The library element is re-tested for the new requirements.

6. **Package the library element** – The library modeler records the environment constraint that the filter must be called at a constant periodic rate (design dependency) and that the filter depends upon an additional library element called MaxMinLimit. The artifacts, such as the requirements, use cases, example usage, design, tests, and test results are packaged.
7. **Publish library element** -

Modify library element

Library elements will evolve over time. The library modeler needs a way to update the library elements without direct impact to the product teams.

Update modeling tool with library elements

When new or changed library elements are available, each product team needs to update the modeling environment. If desired, the product team also needs a way to update the existing models to use the new and improved library elements.

Find library element

Product modelers need a way to find library elements. This includes finding changes to published library elements and new library elements. This also includes finding model elements from different product team models.

Conclusion

Creating product models from library elements is the most important use case selected by the MBSE Usability Group. This paper details the use case steps needed to build and manage a model from library elements. The roles of a product modeler, product supporter, and library modeler are described. The product modelers use the modeling environment to create product models and to update a product model with library element fixes. The library modeler uses the modeling environment to publish a set of common library elements, to create library elements, and to modify library elements. The Product Supporter uses the modeling environment to update the modeling tool with library elements and to find library elements.

The use cases revealed a number of necessary features. For example, there needs to be a way to tailor library elements, to capture environmental dependencies, to capture dependencies on other model elements, and to package numerous artifacts together (requirements, tests, test results, etc.). These features need to be supported by the tools, the tool environments, or by the underlying languages the tools are based upon.