33rd Annual INCOSE international symposium
hybrid event
Honolulu, HI, USA
July 15 - 20, 2023

Modeling System Configurations Over Time

Matthew Hause
Systems Solutions In (SSI)
3208 Misty Oaks Way, Round Rock, Texas
USA
+1 917 514 7581
mhause@systemxi.com

Lars-Olof Kihlström
CAG Syntell
P.O.Box 10022, SE-10055 Stockholm,
Sweden
+46 706661978
lars.olof.kihlstrom@cag.se

Abstract. One of the most troublesome and challenging parts of systems engineering is that systems keep changing. Some changes are planned as part of a phased system rollout. Others can occur because of changes in the environment, technology, customer need, unexpected emergent properties, competition, discovered system vulnerabilities, etc. Regardless of why, the systems engineer needs to document these configurations as part of a digital thread leading to physical systems that will be manufactured and installed. When using SysML as part of MBSE, there are advantages and disadvantages regarding the modeling variability using inheritance and other structural modeling techniques. Elements in a so-called superclass will be inherited by all members. This is advantageous for additive systems but can cause issues when configurations require removal of systems. This paper will examine system configuration over time showing how to model configurations and changes. It will also look at other techniques such as Product Line Engineering (PLE) with OVM and Feature Based Modeling.

Introduction

The Greek philosopher Plutarch posed what is known as the Paradox of the Ship of Theseus. The question presented the concept of whether a system was composed of its parts, or whether the configuration of the parts was what defined the system. In his story, Theseus goes on a long sea voyage during which all the parts of his ship were replaced with different parts. The question is then raised, is it the same ship or a different ship? It has identical looking parts, but they are new, which will affect when maintenance needs to take place or when they should be replaced. In addition, can he be certain that the new parts are of the same quality of the original parts?

Another example is Abe Lincoln's axe. Lincoln was well known for his ability with an axe, and axes associated with his life are held in various museums. Are they all “Abe Lincoln’s Axe”? This may be an interesting intellectual exercise, but how does it apply to architecture? It is obvious that systems change over time for a variety of reasons. These include:

- System lifecycle of design, manufacture, deployment, maintenance, retirement
- Changes for mission-based configurations
- Changes due to maintenance
- Changes due to updates in technology, new or different competition, etc.

Taking the example of an aircraft, if all its parts are changed is it the same aircraft because it has the same tail number, or is it a different aircraft and should we consider it so when creating an architecture? When dealing with physical systems, the identity of the aircraft is in fact the same no matter

how many changes are made to it. However, these changes will need to be reflected in lifecycle processes, maintenance, etc. The model of a system, however, is a completely different situation. (Hause, Kihlstrom, 2013)

Model-Based Systems Engineering (MBSE)

The Systems Modeling Language (SysML) is the most widely used standardized systems modeling language and notation. It is used to model systems in both the abstract and concrete (logical and physical) views that include behavioral, structural, parametric and requirements views. (OMG, 2017). SysML includes an allocation relationship to represent the allocation of functions to elements, allocation of logical to physical elements and other types of allocation.

SysML Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment. Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment. A SysML block describes a system as a collection of parts and connections between them that enable communication and other forms of interaction. Ports provide access to the internal structure of a block for use when the object is used within the context of a larger structure. The Block Definition Diagram (bdd) is used to describe relationships that exist between blocks. The Internal Block Diagram (ibd) is used to describe block internals. Blocks can be composed of other blocks, using a composition relationship, and blocks can be modeled as types of other blocks. The Unified Architecture Framework (UAF) is built on top of SysML and makes use of these features to describe systems and enterprises elements.

The Unified Architecture Framework (UAF)

For enterprise modeling, an architecture framework is required to understand systems of systems and how they change over time. DoDAF is the Department of Defense Architecture Framework (DoD, 2012) and MODAF is the Ministry of Defence Architecture Framework (MOD, 2020). NATO created NAF version 3 (NATO Architecture Framework) based on MODAF and has recently adopted NAF version 4 (NATO, 2018). NAF version 4 has been adopted by many European countries including the UK. UAF is used to define the overall goals, strategies, capabilities, interactions, standards, operational and systems architecture, systems patterns and so forth (UAF, 2019). Security and human factors (personnel) views were added to the UAF to improve the coverage of these areas of concern. The UAF was previously called the Unified Profile for DoDAF and MODAF (UPDM) and was ratified by the Object Management Group (OMG). Several papers have been written on the UAF and its support of SoS modeling including (Hause, Dandashi 2015) and (Hause 2014). The full details of SysML and UAF are not included here for space reasons. Please see the above references for more information.

Inheritance and Composition

Inheritance is the concept in which one class takes on or assumes the attributes, (properties & relationships) and methods of another class. The inherited class is the Parent class, the class that inherits the properties is the Child class. Along with the inherited properties and methods, the child class can have its own properties and methods. This allows us to define common characteristics as well as variations. Figure 1 shows two block definition diagrams of System A and System B. Since they have the same attributes, parts, ports, and relationships, then System A and System B are the same system, that is, they should be modeled with the same block.

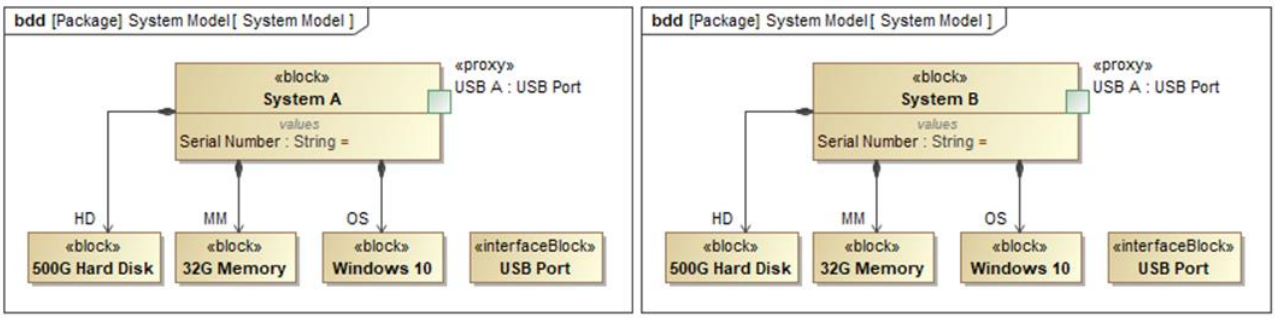

Figure 1. Block Definition Diagrams of System A & B

The left diagram in Figure 2 shows System C. Since system C has an addition port, it is Not in fact the same System as System A.

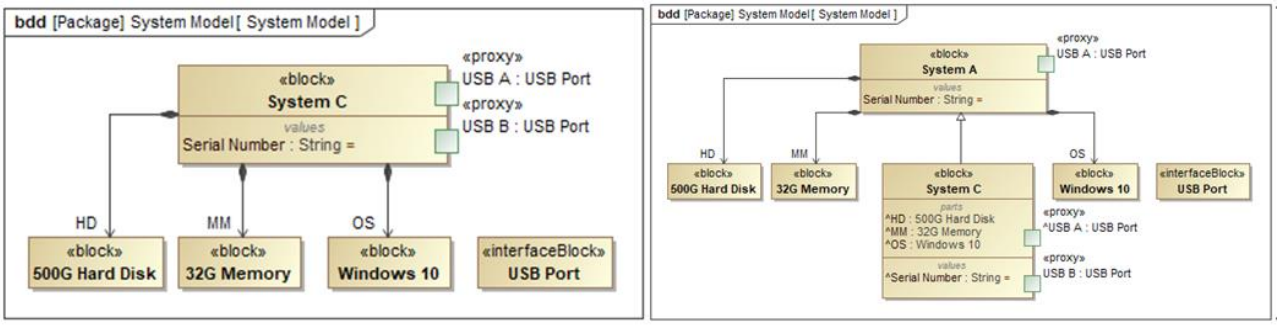

Figure 2. Block Definition Diagrams of System C with and without Inheritance

The right diagram in Figure 2 shows System C inheriting from System A. As is shown in the compartments, it also has the same parts and attributes, and inherits the ports. This use of inheritance means that System C does not need to be defined from scratch but can be built upon System A. Consequently, when building up an inheritance hierarchy it is important to look both up and down the hierarchy to ensure that the parts list corresponds to all the elements in the inheritance hierarchy.

It is important to remember that inherited properties both constrain and enable the child class. The default multiplicity or number of parts is 0..1. Since each class inherits from the parent, it means that they must at least have the capacity to have these parts. Again, when modeling conceptual systems or software this is not a problem. However, for physical systems can be problematic as it implies that interfaces and connections exist in the owning class. Care must also be taken when deciding when and where to add parts. We will also need to look at other concepts such as flows that can take place on connectors between parts as well. A block (class) does not just contain its parts, attributes, etc. it can also be involved in interactions when used in conjunction with other blocks in context. Figure 3 shows a more complex example diagram.

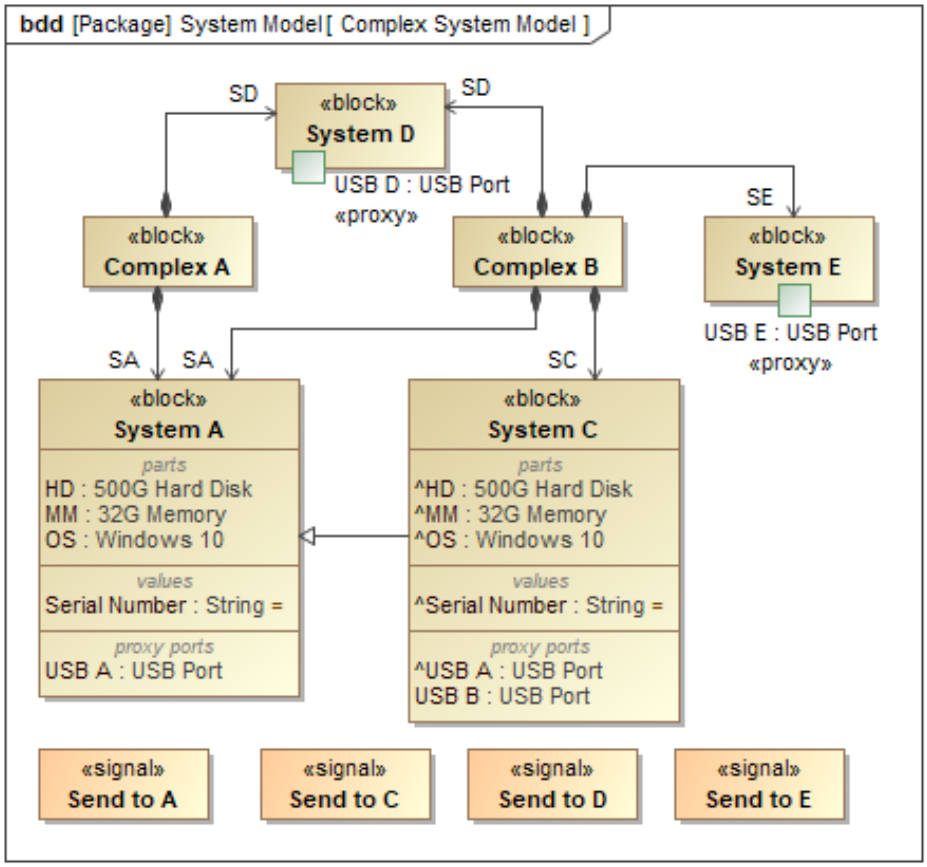

Figure 3. Complex Systems A and B

Complex Systems A & B include System A & System D. Each of these have their own parts and attributes. Complex B adds System C, which is a subclass of System A. and also adds System E. A set of signals has also been defined. Their names correspond to the System that they will be sent to. The internal block diagram for Complex A shown in Figure 3 shows the populated parts of system System A System D. The interchanges signals have also been added.

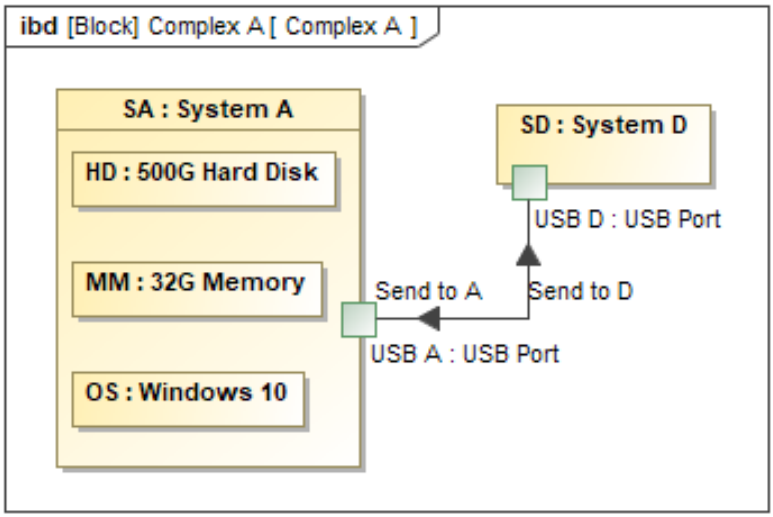

Figure 4. Complex System A IBD.

Remember that System C is a Child Class or Sub-Class of System A so it will inherit its parts, ports, relationships, and attributes. Will it also include the item flows from System A as well? Figure 5 shows an ibd of Complex B which contains both System A and System C and others.

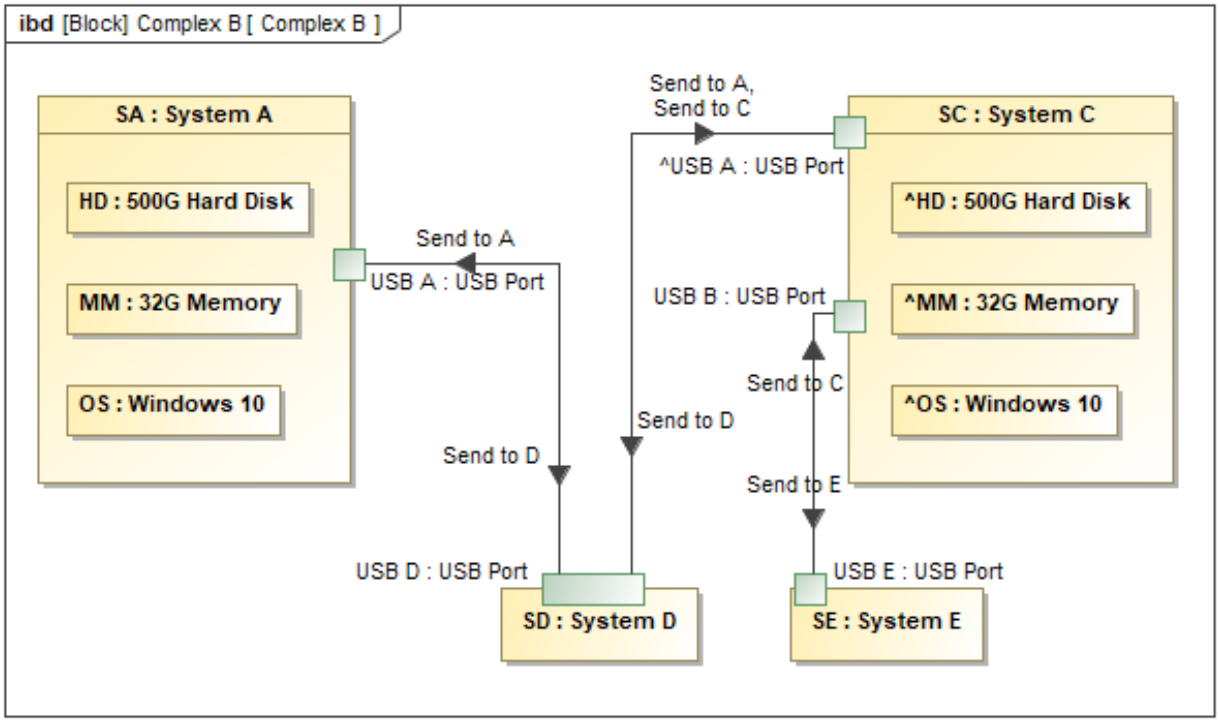

Figure 4. Complex Systems A and B

In Figure 4, the item flows defined between System D and System A in Complex A are populated in the Complex B ibd. Also, the inherited USB A item flows to USB D can be populated onto the diagram. In this diagram, we have also added the Send to C item flow. When we try to populate this message onto the connector between System A and System B, the message is not added as it is not part of that context. In other words, the new item flows remain within their contexts, but can be reused from the inherited block.

System Deployment over Time

UAF provides the ability to model systems and their evolution over time. As mentioned earlier, the project view is used to show when systems are deployed over time. When linked to the capabilities, these can be used to show capability coverage as well as gaps using the CV-3 report. Also bear in mind that when modeling systems, adding new parts to a system configuration creates a new system type. For example, one can model a computer with a specific set of hardware and software. When an instance is created of this system, it will contain all these parts. Also, adding or removing parts to the system means a new configuration and therefore a new structure. The example shown in Figure 5 demonstrates this.

- The Base Configuration is made up of Systems, A, B & C
- The internal resource flow diagram configures them as shown below.
- The different systems each contain two resource ports and connectors.

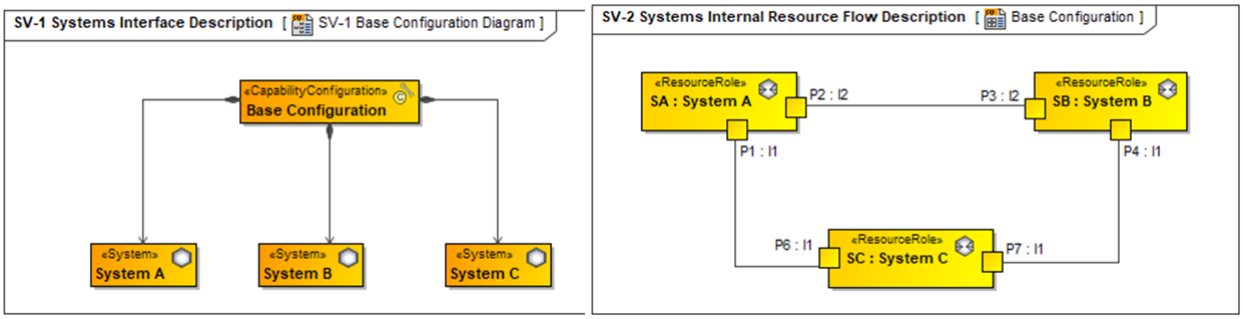

Figure 5. Base Configuration

The Base Configuration will be modified over 3 phases as shown in Figure 6.

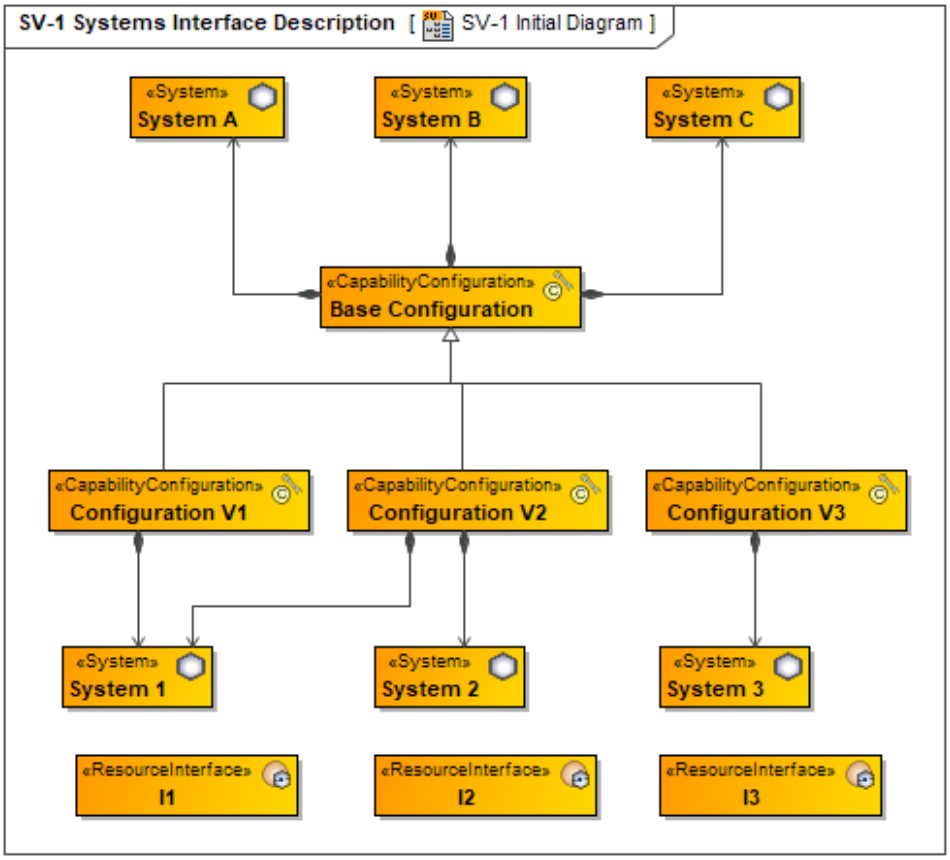

Figure 6. Variations on the Base System Configurations

- Configuration V1 adds System 1
- Configuration V2 adds Systems 1 & 2
- Configuration V3 adds System 3

Systems A, B & C are in all 3 configurations, so we have V1-V3 child classes of the Base Configuration. This relationship as well as the connections are inherited as shown in the Figure 7 diagrams.

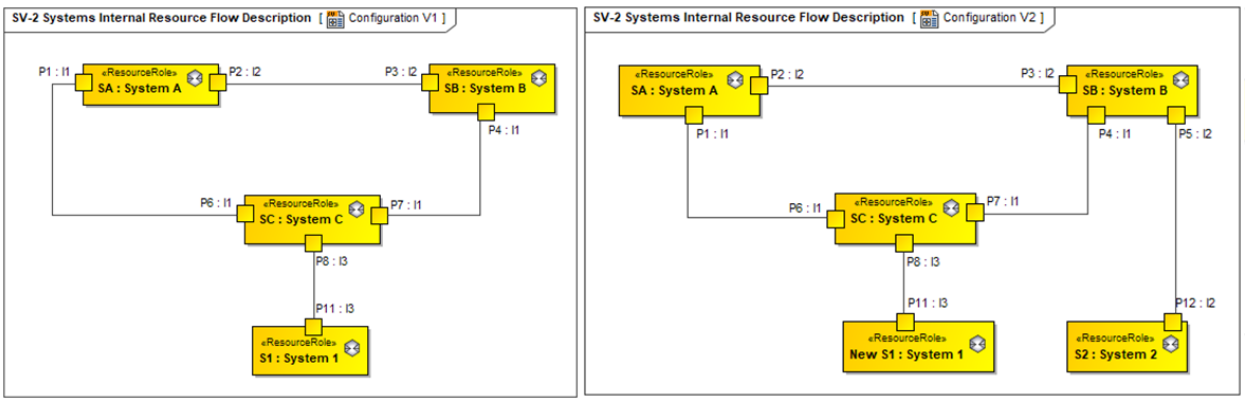

Figure 7. System Configurations V1 and V2

The new configurations have modified Systems B & C. System is 1 added to the configuration V1, Port P8 is added to System C to provide the connector. Systems 1 and 2 are added to the configuration V2, Port P5 is added to System B. What about configuration V3?

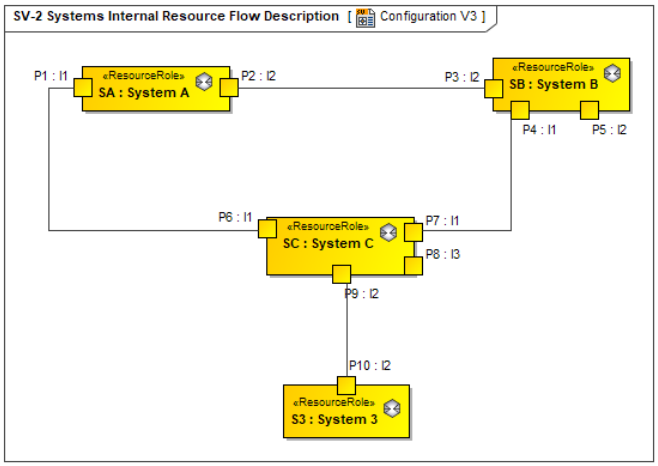

Figure 8. System Configuration V3

System 3 has been added and systems 1& 2 have been removed. Ports P5 & P8 added in configurations V1 and V2 are still parts of Systems B & C. An extra port in the model is unimportant but could be substantial when dealing with physical systems. What if the unneeded ports or systems were:

- Large/heavy systems?
- Safety critical systems?
- Energy hungry systems?
- Unwanted software?
- Creating an access vulnerability?
- Etc.

Consequently, we need to modify the structure to maximize commonality and specialize differences. We need to review where the different parts are and are not necessary. And we also need to review the existing part structure as shown in Figure 9.

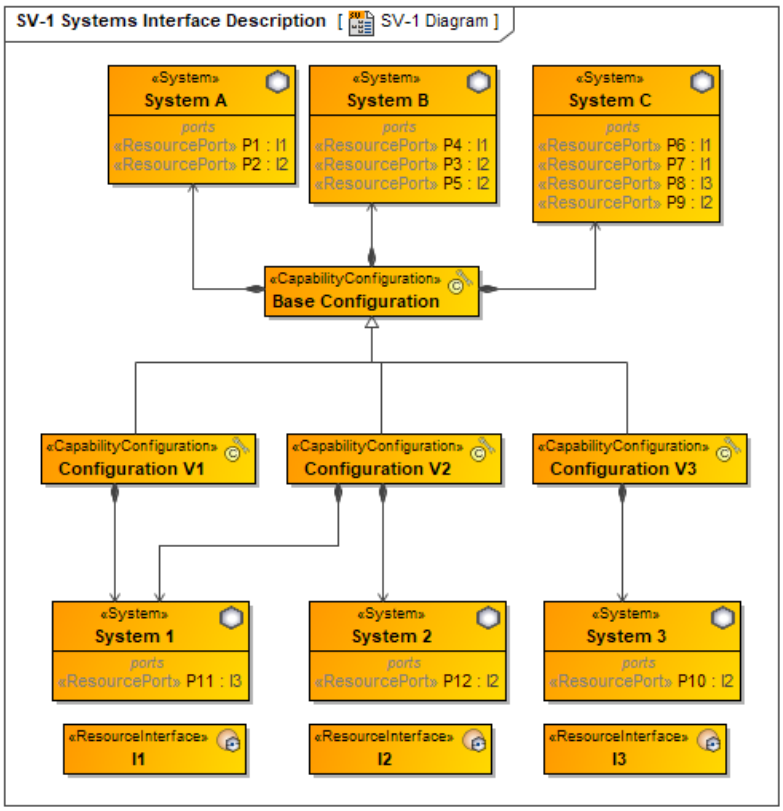

Figure 9. System Inheritance Relationships with Compartments

Figure 9 shows the different components and their parts and ports. Figure 10 shows the repopulated Base Configuration diagram with unused ports on System B and System C.

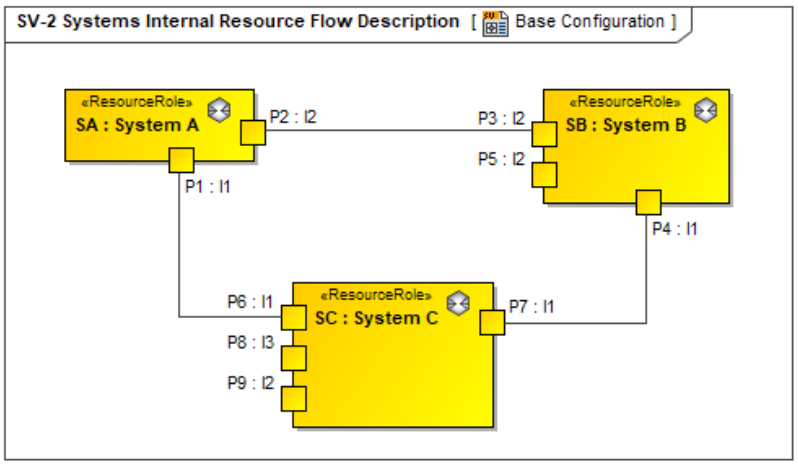

Figure 10. Base Configuration with Ports

We need to reconfigure the systems using inheritance to avoid having unwanted ports and to increase commonality as shown in Figure 11.

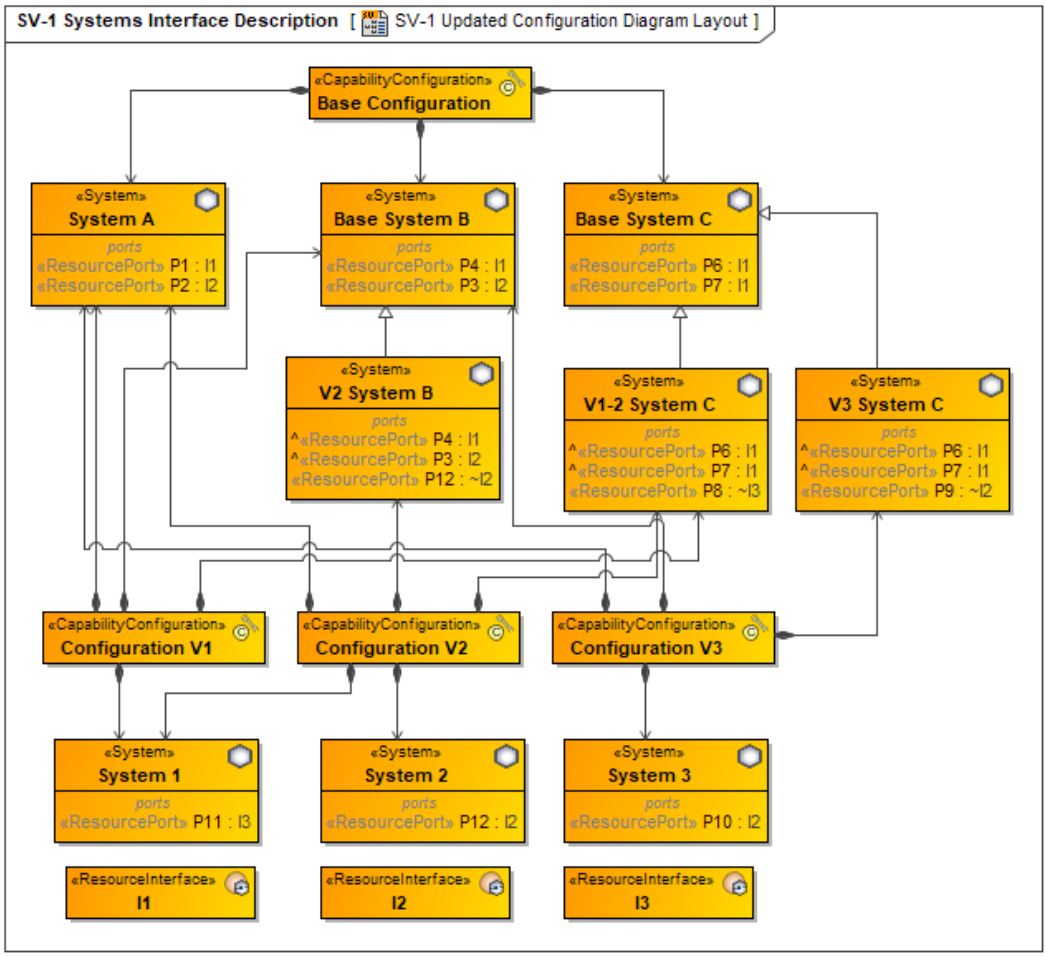

Figure 11. Reconfigured System Structure

Inheritance from the base configuration has been removed for clarity. System A is the only common system between the 4 configurations, so, inheritance is of no benefit and in fact complicates things. However, systems B & C can benefit as their sub elements contain common parts. Again, common elements are in the parent class, and child classes add new ports.

- The Base configuration contains the “original” versions of Systems A, B and C.
- Configuration V1 contains System A, Base B, and V1-2 system C
- Configuration V2 contains System A, V1 System B, and V1-2 system C
- Configuration V3 contains System A, Base B, and V3 System C

This modified structure avoids unwanted parts, maximizes reuse and creates fit for purpose structures.

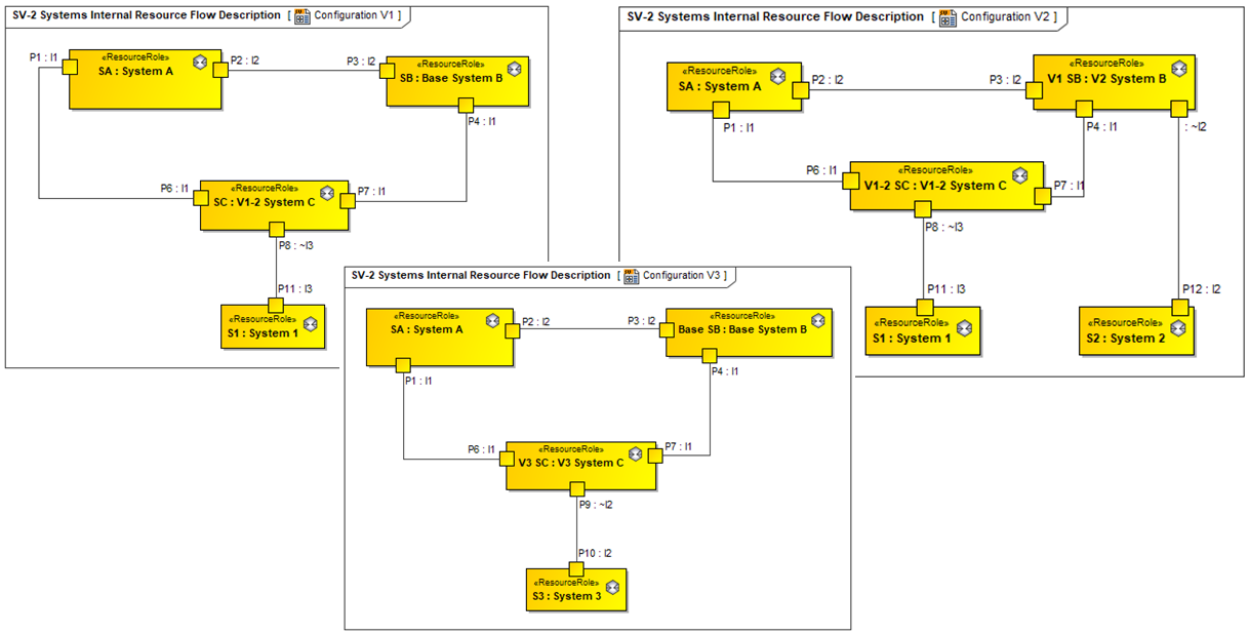

Figure 12. Reconfigured Systems

Having redefined the different systems and added inheritance where necessary, we have a simplified, exact, and explicit view of each system, Again, for a physical system this will have an impact on spare part, maintenance, maintenance procedures, overall system costs as well as through life costs and configuration management,

The Power of Redefinition

By using the concept of redefinition, additional handling of variability can be dealt with. Redefinition allows modelers to redefine the type of an inherited part to a different type. It also allows for the redefinition of multiplicity of an inherited part. This can include redefining the multiplicity to zero when necessary to remove the part from the hierarchy. This can help when doing reasonably simple hierarchical structures but can become difficult when the structures become to complex. A hypothetical complex system of this sort is described in this section. It may be that the system configurations are so complex that it variation management or Product Line Engineering becomes necessary.

Product Line Engineering

Product Line Engineering (PLE) is the engineering and management of a group of related products using a shared set of assets and a means of designing and manufacturing. PLE can include both system and software assets and involves all aspects of engineering including electrical, electronic, mechanical, chemical, etc. As this whole of system approach is also essential for systems engineering, PLE is becoming more relevant to systems engineers. In addition, PLE is being investigated particularly in the automotive arena, but also in rail, power systems, manufacturing and MBSE in general. These are all industries looking to adopt PLE and leverage the capabilities to achieve economies of scale and drive down product costs. (Hause, Korff, 2016)

Orthogonal Variability Modeling (OVM)

There are several enabling technologies and standards that enable PLE and MBSE. Orthogonal Variability Modeling (OVM) provides the ability to model systems and software products, their variation points, mutual exclusions, and product dependencies resulting in product lines. OVM was developed by the University Duisburg-Essen, PALUNO Institute and is now ISO standard ISO 26550: 2013, Reference Model for System and Software Product Line Engineering and Management (Paluno, 2005). Through this modeling technique, users can see their options and conflicts, (if any exist), and to pick their end desired product. (Hause, Korff, 2016) This technique involves creating a model with all possible variations and annotating the model with the variation points, mutual exclusions, and product dependencies resulting in product lines. A product model is then generated from this model resulting in a single configuration. This is also the case for other PLE methods. It has the advantage of allowing you to define the commonality in a single model. It has the disadvantage that the product model needs to be generated to perform trade-off analysis on the model or to view a specific configuration, In the case of UAF architectures showing phased configurations, individual configurations to be deployed over time would need to be carefully modeled to ensure that they exist in the product line model. Otherwise, the roadmap diagrams could not be created. See also Hause, Hummell (2015) for more information.

Feature-Based Product Line Engineering

Feature-based PLE relies on a managed set of features to describe the distinguishing characteristics that set the products in the product line apart from each other. A specialized software tool called a configurator applies the features that describe a particular product to the product line's engineering artifacts (requirements, designs, tests, documentation, models, and so forth) and produces the artifacts specific to that product. Each product is described by giving a list of its features: "A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems". Features are used to express product differences in all lifecycle phase artifacts. In the PLE Factory paradigm, the shared assets (requirements, designs, code, test cases, user manuals, etc.) need to be configured appropriately. Rather than adopt a different "language" and mechanism for each artifact (for example, compiler directives for code, attributes for requirements, text variables for documents, and so forth), we use a small and consistent set of variation mechanisms for all of the artifacts. The PLE Factory configurator automatically derives products from a wide variety of shared assets. Features are used to describe the products' variations, and to tell the configurator the germane characteristics of the product it is to derive by configuring the assets. (Excerpts from https://www.productlineengineering.com/index.html) Multiple references and resources are available here.

This way of working has the advantage in that a separate tool defines the features variants. Links are created to different artifacts in the digital thread from requirements to MBSE to Product Lifecycle Management (PLM). This is a significant benefit. However, the variations and links are not visible in the MBSE model which makes it difficult to visualize. For more information see (INCOSE, 2021)

Redefinition and Product Line Engineering

The intent here is to show how far inheritance and redefinitions is workable and when to transfer to variant management. Figure 13 shows a fairly complex system definition. It can be assumed that Configuration A is valid for a period and that Configuration B then replaces Configuration A. Not all of the Configuration A elements will need to be changed but only some parts. The configurations include different systems, and the systems include a set of resource artefacts. The resource artefacts contain different pieces of software as will be shown in the later Figures.

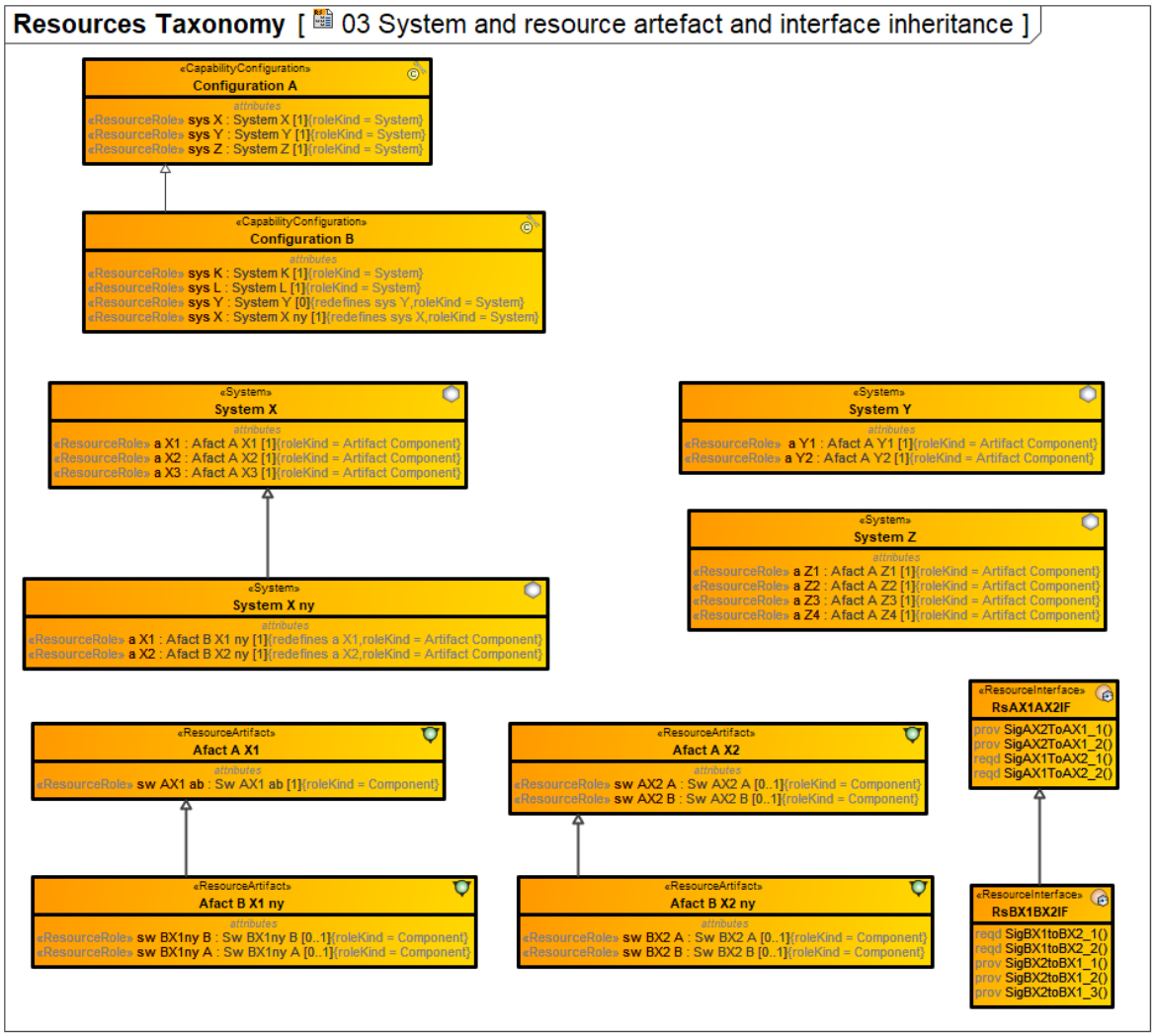

Figure 13. System and Resource Artifact and Interface Inheritance

Figure 14 shows that Configuration A contains three different systems X, Y and Z and that each of these systems contain a set of resource artefacts. The resource artefacts in turn contains a set of software components. There will likely be different usages of the software components during the lifetime of configuration A. Using inheritance and redefinition here is counterproductive since this would complicate the model significantly. At this level it is therefore felt to be a lot better to use variant handling to deal with the different deployments of configuration A during its lifetime.

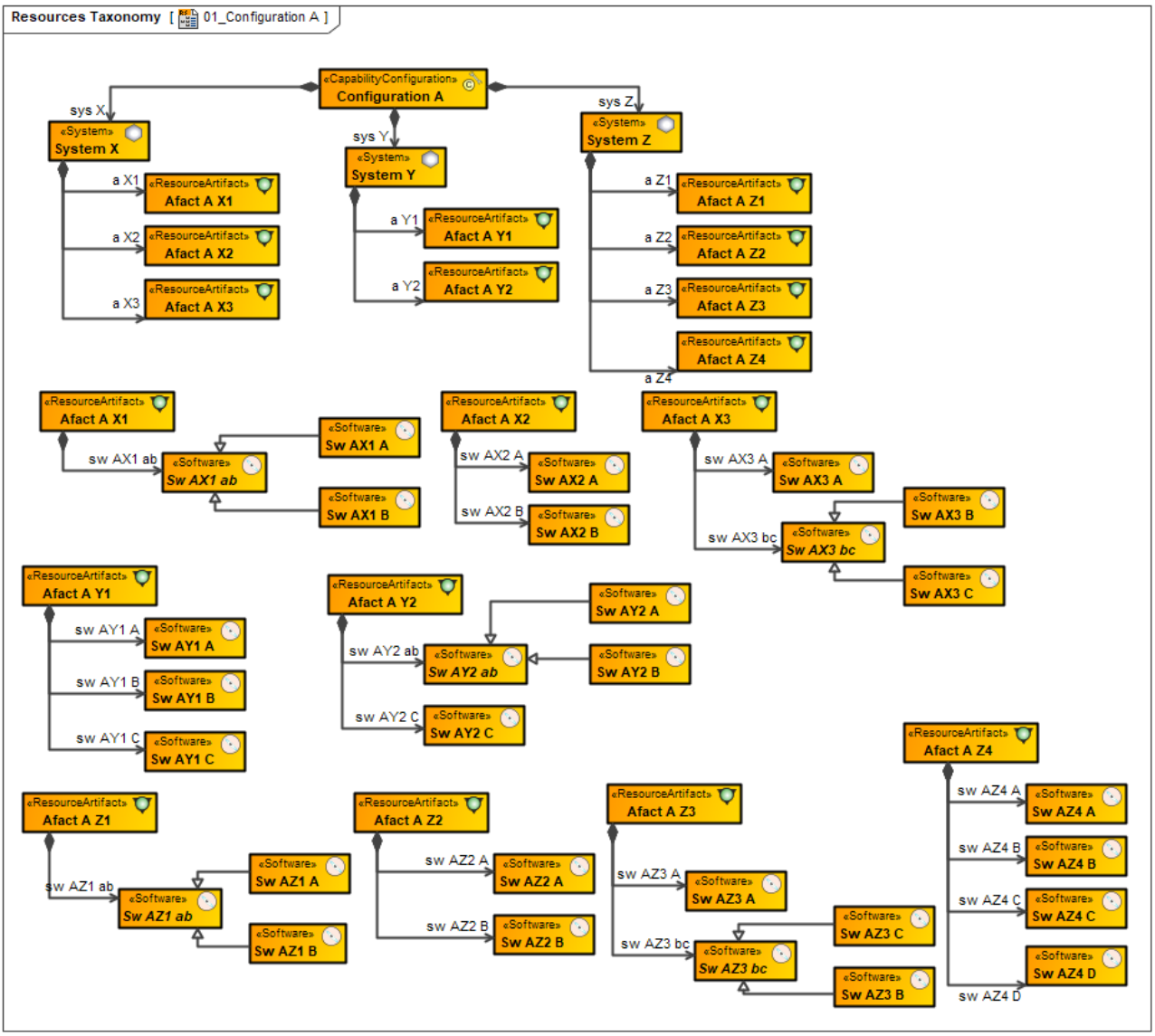

Figure 14. Configuration A Current and Future Elements

Figure 15 shows that Configuration B contains four different systems X ny, Y, Z, K and L and that each of the systems contain a set of resource artefacts. The resource artefacts in turn contain a set of software components. System X ny is a redefinition of system X. System Y has been redefined to itself and the multiplicity changed to 0, i.e., it has been removed. System Z does not show up but it is inherited directly from Configuration A since it is retained unchanged in Configuration B. Also here different usages of the software components that could exist and also here variant handling should be used rather than inheritance and redefinition due to the complexity of the model and the need for exact system specifications.

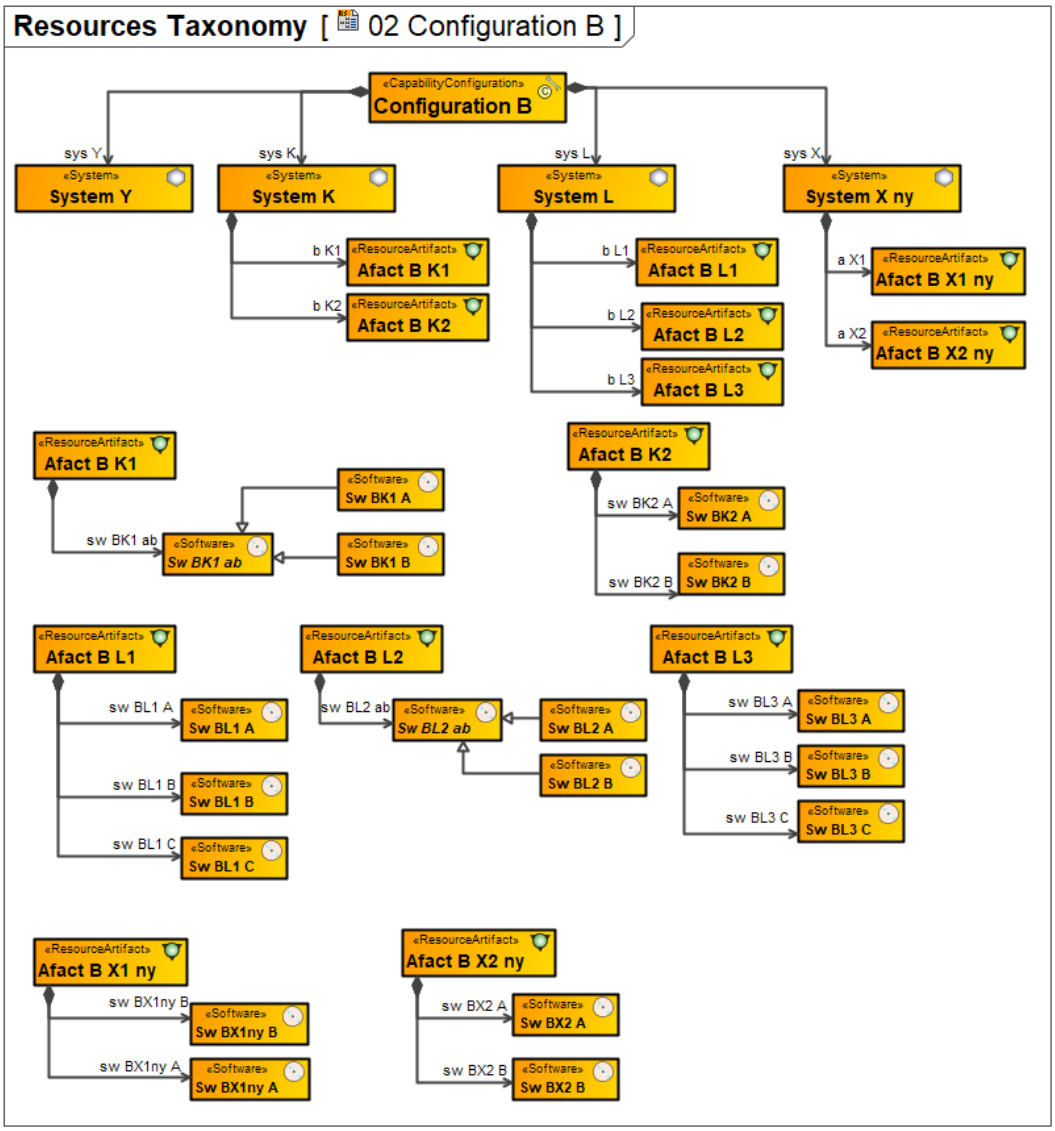

Figure 15. Configuration B

As shown in the previous examples, this will also have an impact on the internal structure of these systems. Figure 16 shows the Resources Internal Connectivity diagram for configuration A that identifies the interactions between the systems. These internal structures were created based on the definitions defined above.

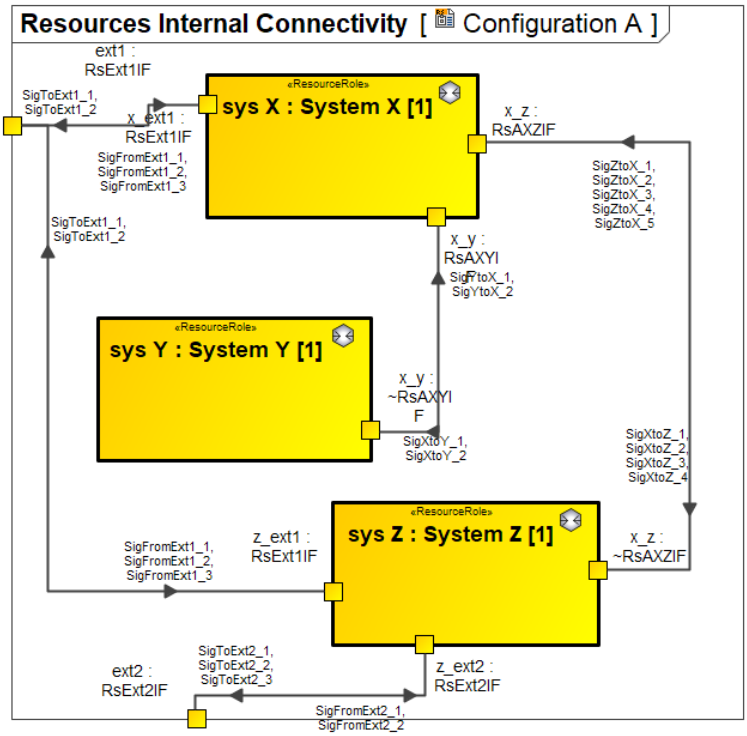

Figure 16. Resource Connectivity Diagram for Configuration A.

Figure 17 shows the internal block diagram for configuration B. As stated, system Y does not show up since its multiplicity has been redefined to 0. System K and L are new additions. System Z is inherited directly from Configuration A and the system X part has been redefined to System “X ny”.

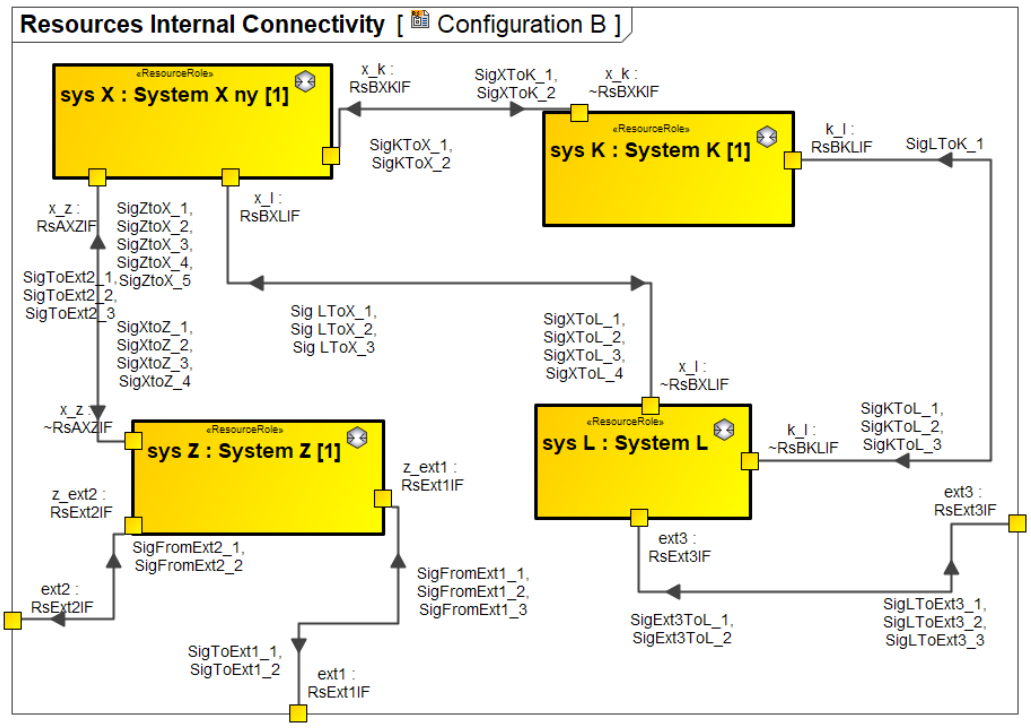

Figure 17. Configuration B Internal Connectivity Diagram

Once again, the parts and interactions between the connectors is inherited and can be accessed by the configuration. Figure 18 shows the internal connectivity diagram for the original system X showing its associated software components, note that some points to abstract representations of two possible software that can be deployed. This is another advantage of the use of inheritance.

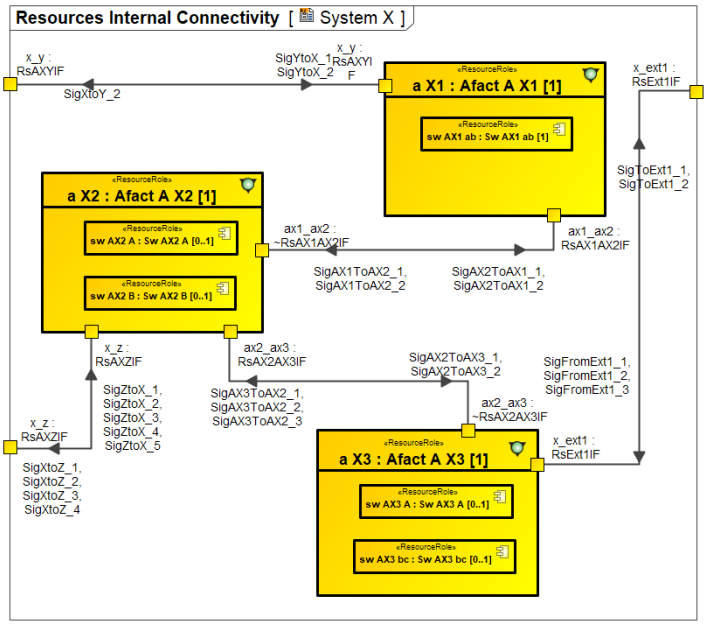

Figure 18. System X Internal Connectivity

(Note that the internal diagrams for system Y and Z have been omitted for reasons of space.) The internal block diagram of the redefined system X is shown in Figure 19. As may be noted, two of its parts have been redefined and one remains from the original system X. The interactions between the aX1 redefined and aX2 redefined has also been changed compared to the original. The ax1_ax2 port in both of these elements have been redefined and now is typed to another Resource interface that inherits from the original resource interface used in System X. A set of new signals have been included within the interface block that inherits from the original one.

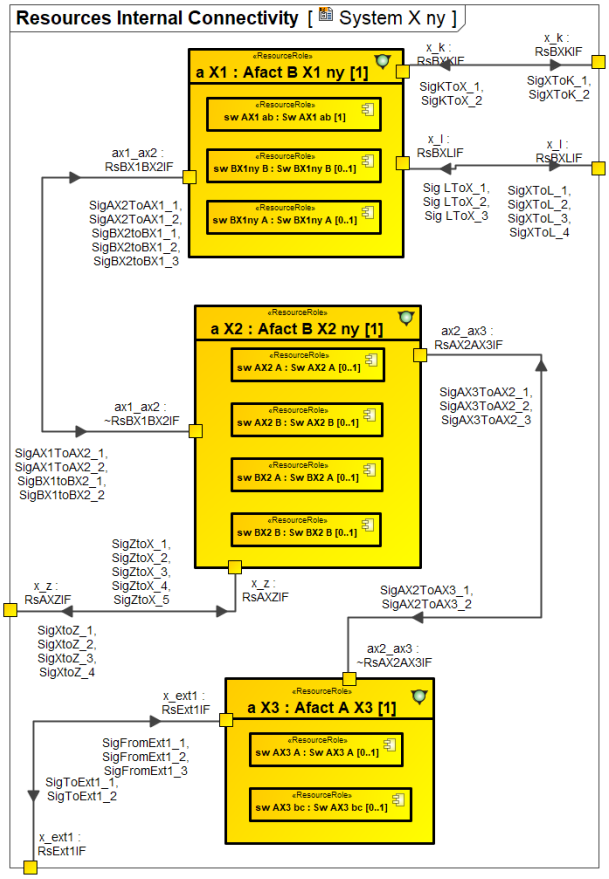

Figure 19. Redefined System X Diagram

Variant management using Product Line Engineering (PLE) is useful in the example used in this section as it would provide a means of specifying which parts are in each configuration and provide a means of generating a solution architecture which contains only the essential elements. This is useful for system test as well as for generating bills of material (BOM) and other system aspects. As stated, the disadvantage is that it is not possible to show the unique system changes over time as is necessary in enterprise systems. Each modeler will need to determine which MBSE method to use at the appropriate time.

Conclusion

Systems always have and always will change over time for the various reasons given in this paper. The job of the engineer is to keep track of those changes as well as the variations and the purpose of each of them. It is the job of the systems engineer using model-based techniques to capture these variants and configurations, often in SysML. As instructors in SysML and UAF, we have found that most modelers do not understand the implications of inheritance hierarchies or how to model different configurations other than in different models or by "clone and own". This paper has spelled out various techniques for modeling configurations, including the pros and cons for each. Hopefully this will assist people in modeling configurations for their products and investigating these techniques.

References

Friedenthal, S., Moore, A., Steiner, R. Practical Guide to SysML: The Systems Modeling Language, Morgan Kaufman September 2008

Hause, M., Hummell, J., 2015, Model-based Product Line Engineering – Enabling Product Families with Variants, presented at the 25th Annual INCOSE International Symposium in Seattle, Washington, USA

Hause, M., Korff, A., 2016, Decision-Driven Product Development, presented at the 26th Annual INCOSE International Symposium in Edinburg, Scotland

Hause, M, Kihlstrom, L, Architecting in the Fourth Dimension - Temporal Aspects of DoDAF, presented at the 23rd Annual INCOSE International Symposium in Philadelphia, Pennsylvania, USA

INCOSE, 2021, Welcome to the Product Line Engineering International Working Group!, available from https://www.incose.org/incose-member-resources/working-groups/analytic/product-lines

ISO 15288:2008 Systems Engineering standard covering processes & life cycle stages.

ISO 26550:2013 for Software & Systems Engineering – Reference Model for Product Line Management & Engineering.

Kang K. , et al., 1990, Feature Oriented Domain Analysis, Software Engineering Institute (SEI)

Object Management Group (OMG). 2013. OMG2013-08-04:2013. Unified Profile for DoDAF/MODAF (UPDM) V2.1, http://www.omg.org/spec/UPDM/2.1/PDF

Object Management Group (OMG), 2019. OMG2012-06-01.OMG Systems Modeling Language (OMG SysML™), V1.6, http://www.omg.org/spec/SysML/1.6/PDF/.

Object Management Group (OMG), 2019, The Unified Architecture Framework, (UAF) Version 1.1, Available from https://www.omg.org/spec/UAF

Object Management Group (OMG), 2022, The Unified Architecture Framework, (UAF) version 1.2, expected date of publication, March 2022.

PALUNO, The Ruhr Institute of Software Technology Software Product Line Engineering (Pohl et al - Springer 2005)

Biography

Lars-Olof Kihlström. Lars-Olof Kihlström is a principal consultant at CAG Syntell where he has worked since 2013, primarily in the area of MBSE. He has been a core member of the UAF group within the OMG since its start as the UPDM group. He was involved in the development of NAF as well as MODAF. He has worked with modelling in a variety of domains since the middle of the 1980:ies such as telecommunications, automotive, defence as well as financial systems. He is specifically interested in models that can be used to analyze the behavior of system of systems.

Matthew Hause. Matthew Hause is a principal consultant at SSI, a member of the UAF group, and a member of the OMG SysML specification team. He has been developing multi-national complex systems for almost 40 years as a systems and software engineer. He worked in the power systems industry, command and control systems, process control, SCADA, military systems, and many other areas. His role at SSI includes consulting, mentoring, standards development, specification of the UAF profile and training.