ENTERPRISE ARCHITECT

User Guide Series

Guide to MBSE with SysML

Author: Sparx Systems
Date: 2026-05-27
Version: 17.1

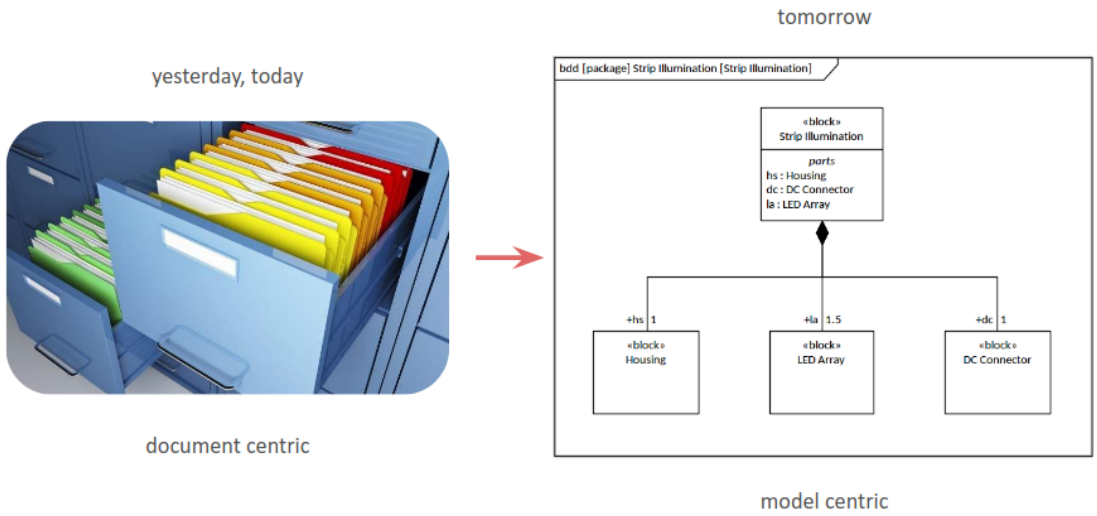
CREATED WITH ENTERPRISE ARCHITECT

Table of Contents

Guide to MBSE with SysML

Model Based Systems Engineering (MBSE) has emerged as a valuable approach to the management and engineering of complex systems. It is a shift away from the document-centric approach and allows models to be developed and used for a wide range of purposes, including Requirement specification, design, trade-offs, architecture, verification, validation, simulations and support, and more.

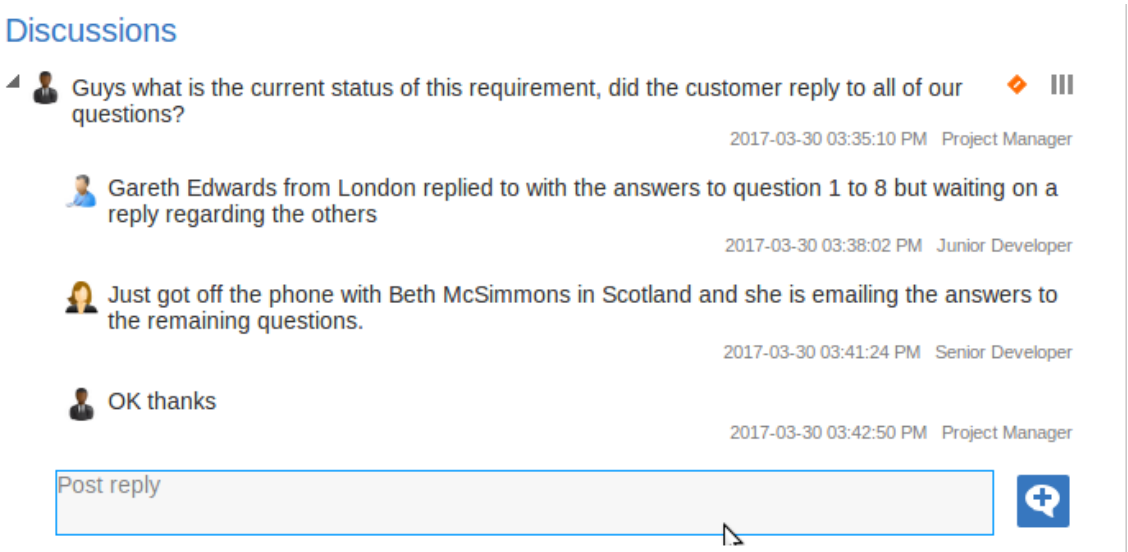
The models act as an 'insurance policy' against catastrophic engineering errors and help to reduce the high cost of failure at the specification, design, test, implementation and support phases.



When you use Enterprise Architect for MBSE you open the door to a completely new way of thinking and working. Enterprise Architect is a rigorous collaboration platform that allows ideas, problems, solutions and implementations to be shared by a wide range of stakeholders including:

- Customers
- Executives
- Engineering Managers
- Engineering Team leaders
- Architects and Designers
- Systems Engineers
- Software Engineers
- Testers
- Suppliers
- System Integrators
- Support Staff
- Users or their surrogates

The models are also available in real-time for view, contributions, reviews and discussions through a secured Browser interface, utilizing the power of collaboration to create robust and well formed architectures and designs.



In this guidebook we will explore many of the features of Enterprise Architect that can be harnessed to take an individual engineer, a team, an organization or an entire industry segment to the level of practice and performance that is required in an age dominated by innovation and unprecedented technical change. The guidebook will provide background for the Systems Modeling Language and show how the language constructs can be created in Enterprise Architect. It is intended to give a newcomer to both the language and the tool exposure to all that is possible in the field of Systems Engineering using this engineering and collaboration platform. In the words of the famous 17th Century physicist, Sir Isaac Newton:

'If I have seen further than others it is because I have stood on the shoulders of giants.'

Enterprise Architect provides a platform where this vision can take place, where through the Collaboration features and facilities such as simulations and automation an engineer can see opportunity, design solutions, and architect the future.

Enterprise Architect includes major features for systems engineers, technical architects and others wishing to couple their modeling and simulation work in Enterprise Architect with MATLAB, Octave, OpenModelica and more. 'Solver' classes and an extensive Math Library in the JavaScript engine provide a significantly expanded Simulation capability. For more information, see the Mathematical Simulations Help topic.

An Equation with Four Variables

The title of this guidebook implies that there are only two disciplines to learn:

- Model Based Systems Engineering (MBSE) and
- The Systems Modeling Language (SysML)

However, there are four aspects to be considered when taking on this approach. This is like an equation with four unknowns, each of which must be addressed before a team can be successful with an MBSE project or initiative. The variables in the equations are:

- The engineering method or process
- The discipline of modeling
- The Systems Modeling Language (SysML)
- Enterprise Architect, the modeling tool of choice

It is not imperative that these four aspects be mastered straight away, but it is important that they are known and worked on and that the team develops the skills to understand them individually and how they relate to each other. For example, how to create a SysML Requirements diagram using Enterprise Architect and what properties should be included, when should it be done and what other parts of the model should it be related to.

$$MBSE\ Approach = Process + Language + Modeling + Tool$$

The guidebook will address all of these concerns, and by the end the reader will not be confronted with unknown variables, but the equation will have values based on the learning, thus solving the equation with the four variables that we started with. At this point the reader will be well on their way towards a sound engineering modeling practice.

The Engineering Method or Process

The Systems Modeling Language is process agnostic and can be used with any method or process. This point is sometimes not understood by newcomers to the language who expect that it should be prescriptive and give clear guidelines as to what elements, diagram and models should be created and when. This agnostic position provides great flexibility and allows the language to be used in ways that are applicable to the process and the underlying problem or solution domain.

The elements, connectors, diagrams, and language definitions defined as part of the Systems Modeling language have all been created with the express purpose of allowing engineers to create models of the:

- mission
- stakeholders
- requirements
- measures of effectiveness
- structural and behavioral aspects of a system such as the components that ultimately implement the requirements

The process that is used by a team to create, manage and disseminate the Artifacts is completely arbitrary and must be defined at an organization or team level.

System Engineering typically requires a collaborative or multidisciplinary approach where teams work together to produce a result that meets the stakeholders' needs. There are two important aspects to any process:

- A management process - which governs stakeholders, risk, schedule, budget and quality
- A technical process - which manages architecture, analysis, design, integration and testing

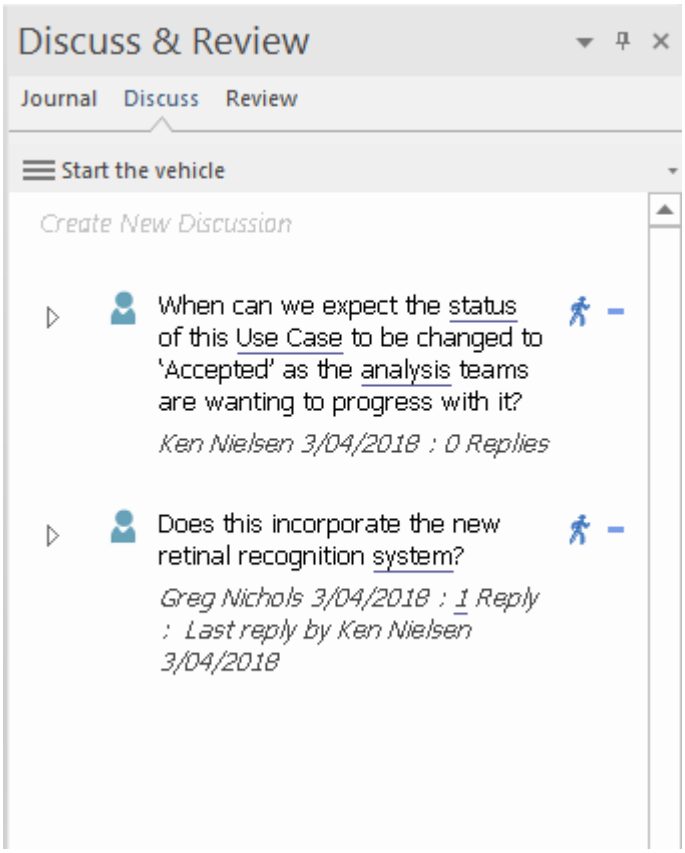
The two processes, however, clearly require touch points to ensure that the overall mission and the goals and objectives of the project are being met.

Enterprise Architect allows you to use any type of process regardless of whether it is formally defined, part of a standard or crafted in-house. There are also facilities within Enterprise Architect that allow you to define, publish and share a bespoke process.

A Well Supported Team

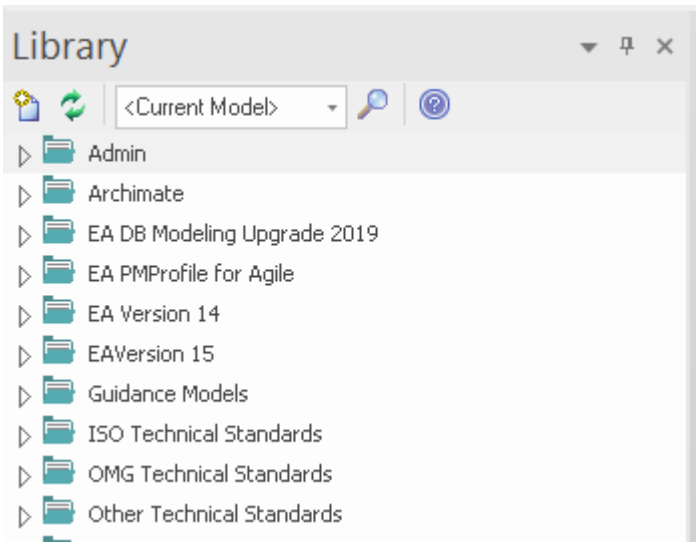
Enterprise Architect provides a large range of tools that will help teams collaborate regardless of where they are geographically located or how they are separated by time and distance. The product has been built as a collaborative platform from the ground up, allowing engineering and non-engineering, technical and non-technical stakeholders to work together in a collaborative and integrated structure.

The repository can be Cloud-based, and users can connect securely from anywhere on the globe, effectively creating a virtual team. This is important for a number of projects where expertise is not available locally or where the project itself is global. The users and teams can use the collaboration features such as Discussions, Chats, Reviews and Model Mail to work together. The result will be collaborative architecture and design that is not the result of one engineer's work, but the output of many minds, and the work will be more than the sum of its parts.

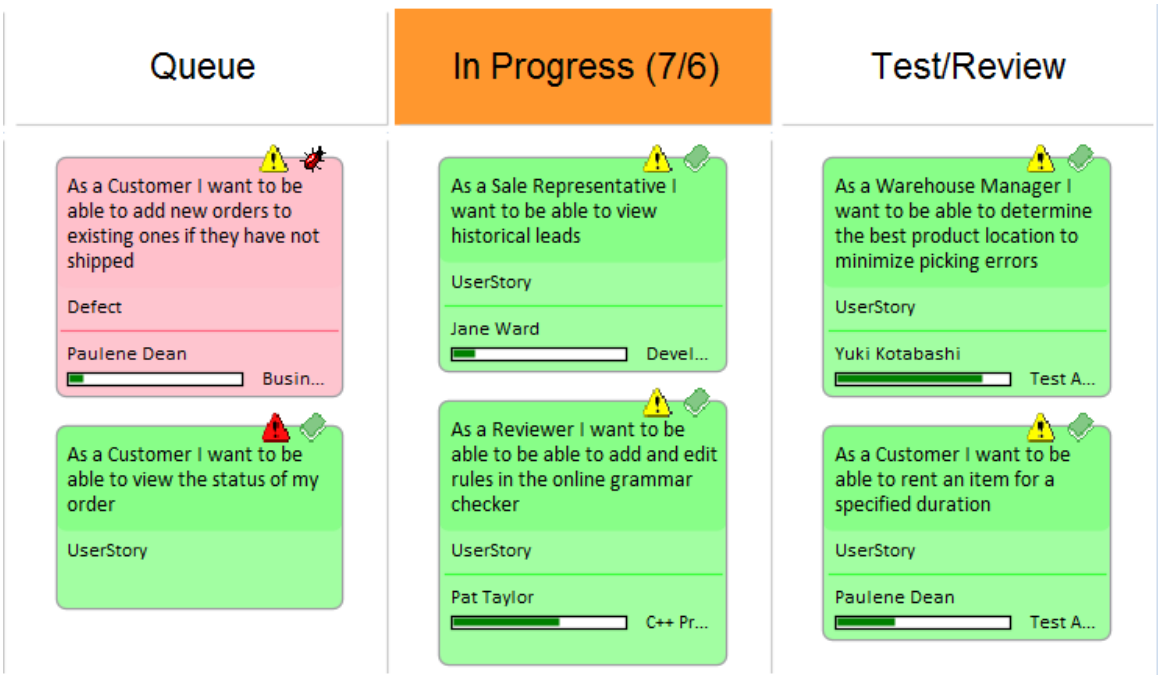


These tools are effective because they can be used to annotate models, elements and diagrams, allowing users to work together as though they were working collaboratively on a whiteboard in the same room.

The Model Library is another handy collaboration tool that allows any type of file to be either included in the repository or listed with a hyperlink and/or URL reference to its external location on a web site. Documents such as standards, specifications, guidelines, guidance, examples, mentors and other material can all be catalogued in the Model Library.



There is a wide range of other tools that can be used to facilitate team work, including the Image Manager, Calendars, Publishing, Kanban, Project Management features and many more. This example shows a Kanban diagram that can be used to visualize what is being worked on in an Agile team developing physical or software components of a system. For more information see the The Modeling Team Help topic.

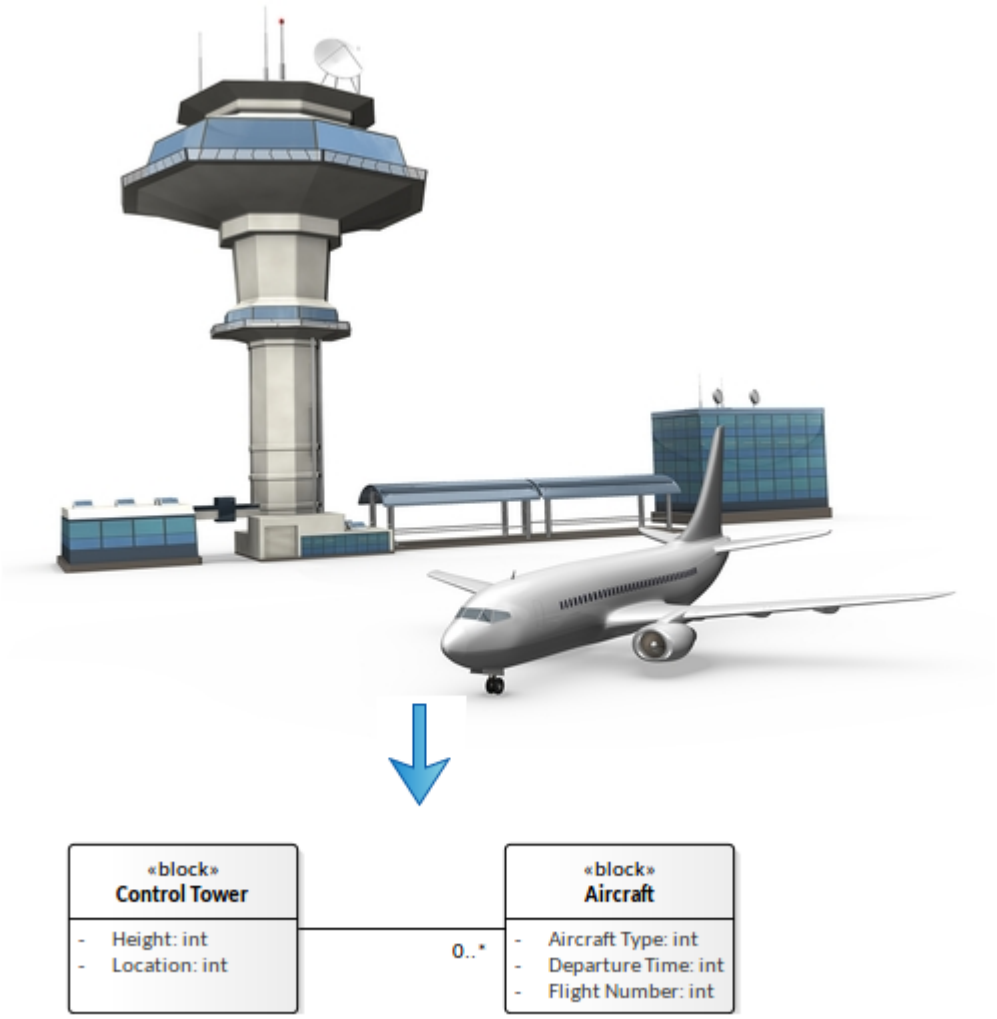

Queue
In Progress (7/6)
Test/Review
As a Customer I want to be able to add new orders to existing ones if they have not shipped
Defect
Paulene Dean
Busin...
As a Customer I want to be able to view the status of my order
UserStory
As a Sale Representative I want to be able to view historical leads
UserStory
Jane Ward
Devel...
As a Reviewer I want to be able to be able to add and edit rules in the online grammar checker
UserStory
Pat Taylor
C++ Pr...
As a Warehouse Manager I want to be able to determine the best product location to minimize picking errors
UserStory
Yuki Kotabashi
Test A...
As a Customer I want to be able to rent an item for a specified duration
UserStory
Paulene Dean
Test A...

Modeling as a Discipline

Most people, systems engineers included, typically find it easier to write a long description of a topic rather than a succinct, concise summary - this is analogous to the challenge of modeling.

The question isn't so much what to include, but rather what to leave out.

One of the advantages of model based system engineering is precisely this - it encourages engineers to create models that are descriptive, crisp and concise. The long (and sometimes rambling) sentences of document based processes are replaced with clear and laconic diagrams that unambiguously describe the requirements, the structure and the behavior of the system.



There are those who describe modeling as a hermetic discipline and speak of it as one of the 'dark arts' practiced by alchemist engineers robed in purple gowns. This underlies the issue that modeling is seldom taught as a subject in our universities, nor are there vast quantities of literature on the topic, making it appear to be a mysterious art rather than what it is - a science that can be learnt.

There are a number of different types of model including:

- Scale Models
- Physical Models
- Abstract Models

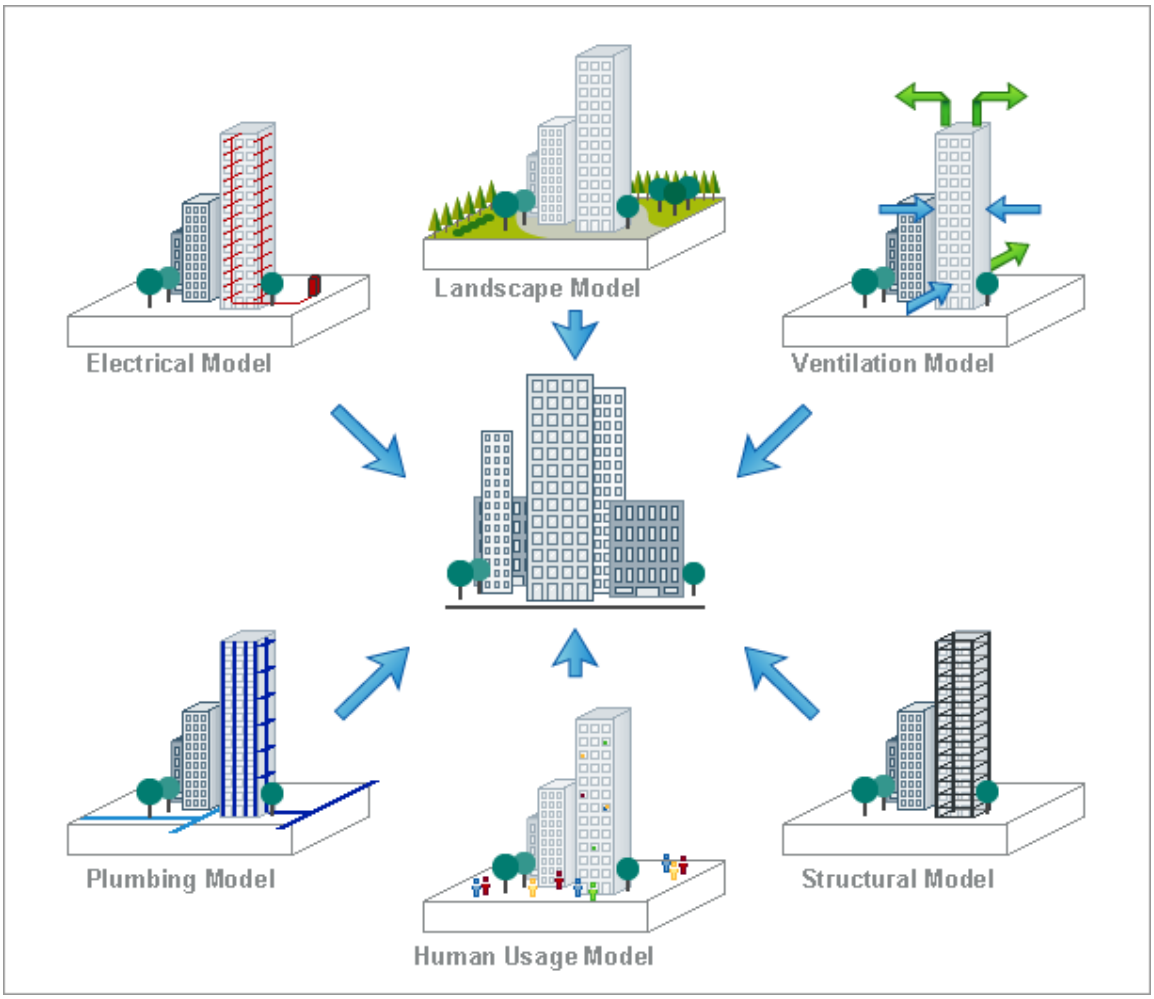
In this guidebook we are most interested in Abstract models, as they are the models we will typically be creating using Enterprise Architect and the Systems Modeling Language.

These models are - as the name suggests - abstractions of reality that seek to highlight the most important aspects of an entity, subsystem or system, while leaving out the things that are not important or are irrelevant from that viewpoint. For more information see the Models Help topic.

Getting Started

An abstract model is a representation of a real world thing in a way that helps us to reason about it without needing to view the real thing. Typically a model is much smaller and is a simplified view of a system or one of its parts. A model can also be created that focuses on just one aspect or facet of a system; for example, the communication or navigation system of an aircraft.

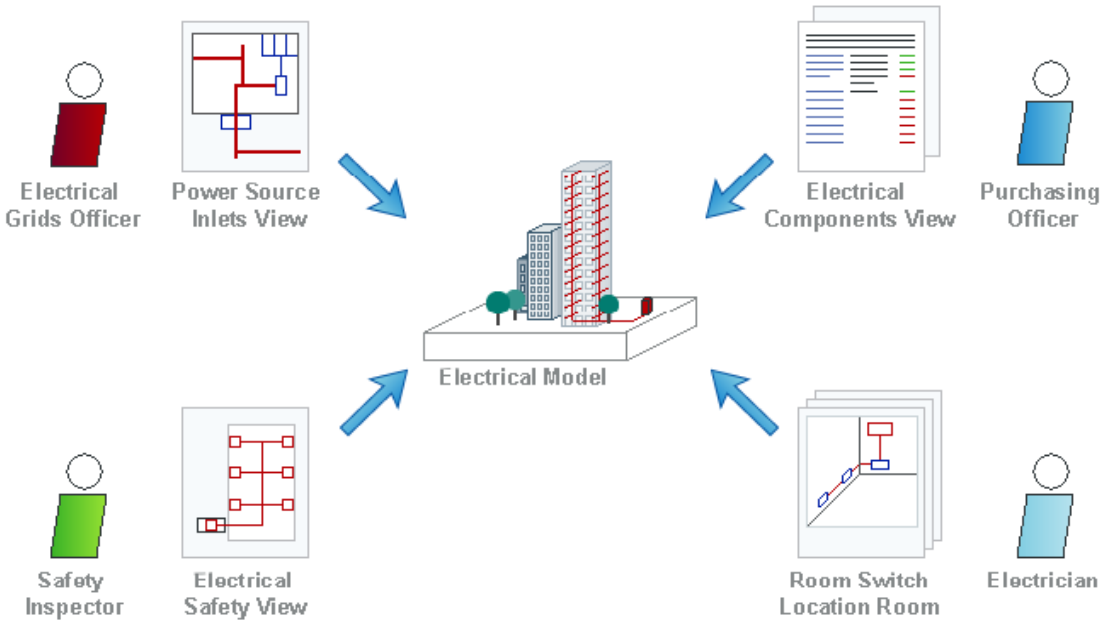
A building is a complex structure that has a number of different systems including Structural, Electrical, Ventilation, Plumbing, Landscape and more.



By constructing a number of models we are able to create a simplified view of each of the subsystems, which makes it easier to understand that aspect of the building. The models themselves also need to be resolved against each other. For example, it is critical that the power system represented in the electrical model provide electricity to the air-conditioning equipment modeled in the ventilation system. The Human Usage model needs to be resolved against the Landscape model to ensure that the gardens and landscaping meet the recreational needs of the occupants and their visitors.

A model will typically be viewed by a number of different stakeholders who commonly have quite disparate roles with respect to the part of the system being modeled. To ensure the model is useful to a particular stakeholder, views can be created representing what is seen when looking at the model from a particular viewpoint.

The electrical model of the building is useful to a number of different stakeholders, all of whom have a different viewpoint with respect to the system, including the Electrical Grid Officer, the Safety Inspector, the Electrician and the Purchasing officer.


Electrical Grids Officer
Power Source Inlets View
Electrical Components View
Purchasing Officer
Electrical Model
Safety Inspector
Electrical Safety View
Room Switch Location Room
Electrician

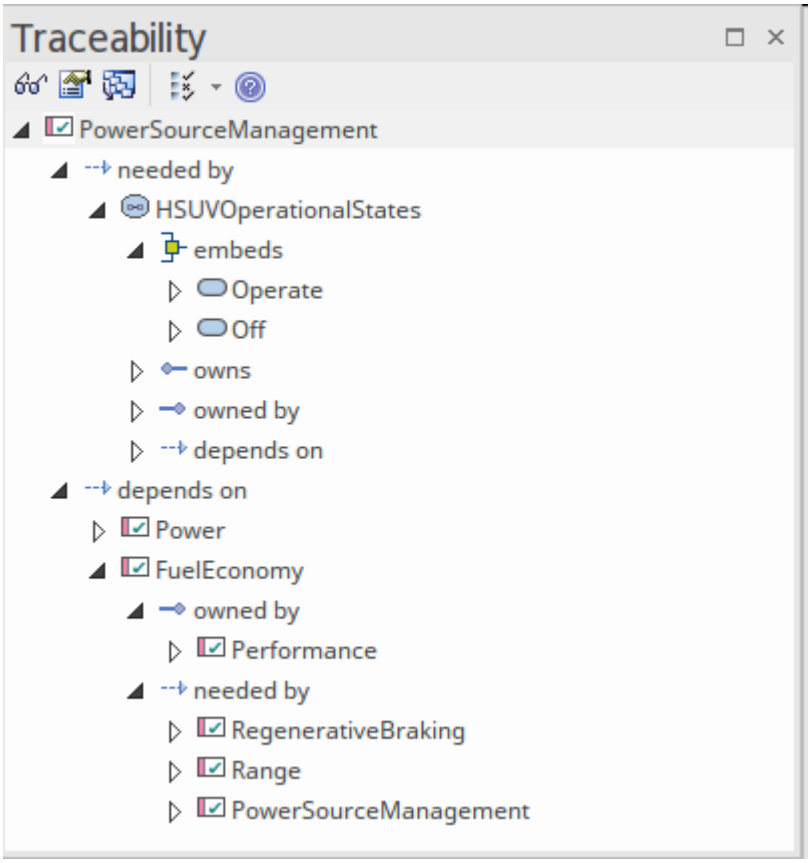
Defining a Model's Purpose

Moving to Model Based System Engineering brings with it a number of challenges and traps for teams who are more accustomed to working with document-centric methods. Probably the commonest trap is to start modeling without having a clear understanding or definition of the purpose of the models.

In comparison with document-centric modeling approaches, it is more difficult to define the purpose of a model than it is to define the purpose of a suite of documents. The model is orders of magnitude more useful and effective than a document and can be used to perform work that is unimaginable with a document-based system. Some of the advantages of the model based approach are:

- Consistency checks can be easily applied
- Alternative views can be readily created and kept consistent
- If documentation is required it can be generated automatically
- Models are interlocking and consistent
- Change impact can be visualized and automated
- Models can be kept compliant with an underpinning metamodel
- Models can be versioned and baselined
- Requirements traceability can be easily managed
- Models can be easily manipulated and changed
- The model can be used to generate code and standards
- Models can be simulated producing rich visualizations
- Models can be transformed from one level of abstraction to another
- Parts of the model can be easily reused creating efficiency

This diagram shows how traceability can be visualized and managed in the tool, allowing you to view the way that parts of the models interlock and how elements form a graph of connections, helping you to describe and comprehend your model.



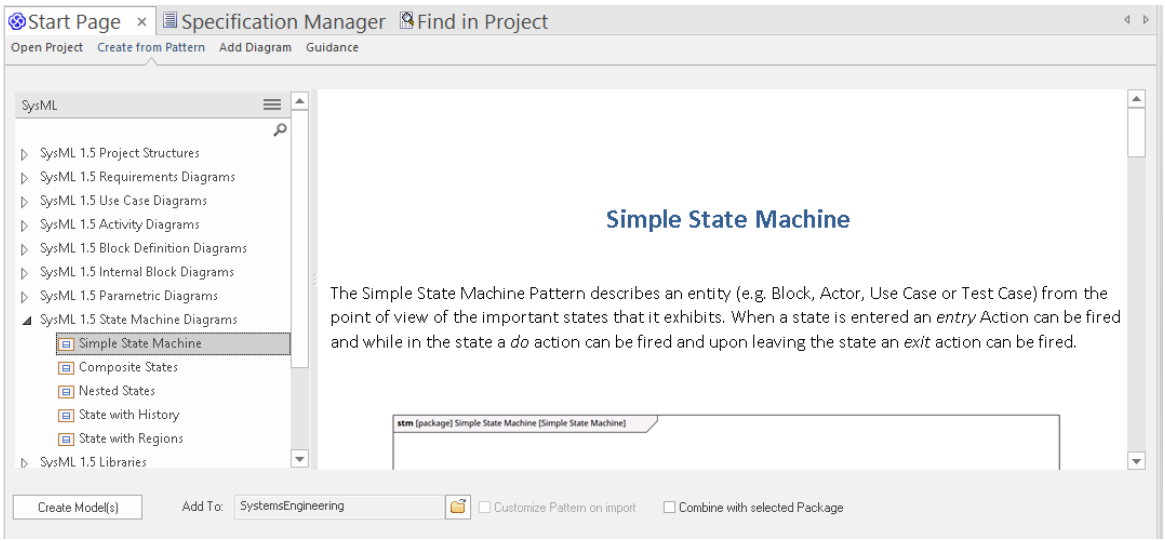
Enterprise Architect harnesses the power of the SysML, along with a large set of tools built with System Engineering Managers, Systems Engineers and other stakeholders in mind, providing simple but effective ways to take advantage of the Model-Based Systems Engineering approach.

There are other profound benefits that can be achieved by moving to a model-based approach, including ensuring that projects and programs of work are performed with rigor, productivity and efficiency using a tool that encourages excellence and collaboration. For more information see the Why Enterprise Architect? Help topic.

Deciding Where to Start

The process of modeling can be quite daunting for engineers new to Model Based Systems Engineering. More than anything else is, apparently, the issue of where to start modeling - the engineer's equivalent of the artist's 'blank canvas' inertia.

Enterprise Architect provides a welcomed solution to this issue, by providing a series of patterns that can be used to create starting points for an initiative or project, including all of the SysML diagram types with a number of patterns for each type. For more information see the The Model Builder Help topic,

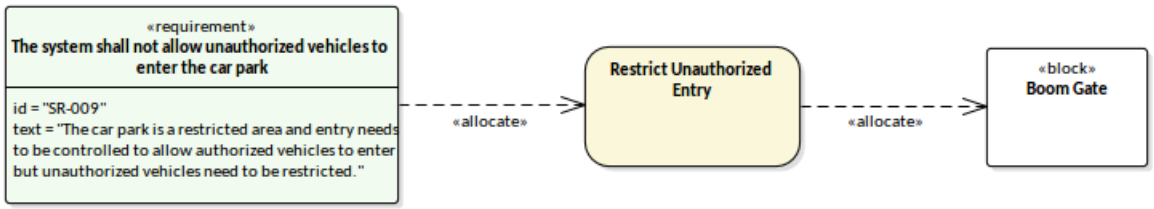


Textbooks typically describe a series of steps that should be carried out in a prescribed order, but in practice these recipes don't work because projects are substantially more complex than the generic ways described in the books, and complex project and resource dependencies mean that tasks cannot be performed in a prescribed order.

The starting point will typically be determined by the engineering method or process being used for the project, which could be a waterfall, an iterative, a combination of both or another type of process. Regardless of the type of process being used, having a clear understanding of the mission is often a good starting point, and defining the affected stakeholders and their concerns and requirements is often a good next step.

Connecting Parts of the Model

The Systems Modeling Language encourages engineers to create a series of models, which will seem to the novice or newcomer to model-based systems engineering to be fragmenting the view of the system. In reality the SysML describes a network of models, each addressing particular concerns but connected together to describe the system and its parts as a whole.



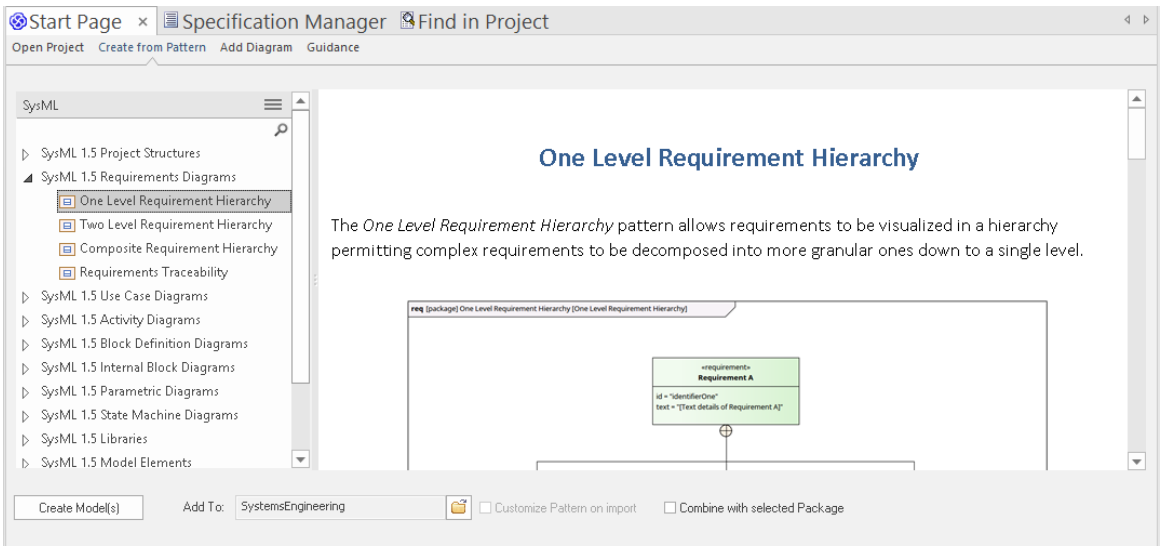
In this illustration we see a compelling Enterprise Architect diagram that depicts the connections between parts of the model using specialized elements and connectors, namely a Requirement, an Activity and a Block using the Allocate relationship. The elements can be reused in any number of diagrams, and changing their properties in one location will update them in every context. Diagrams can be created quickly and easily using a number of features, and they can be visualized in a wide range of ways such as lists, tables and spreadsheets. The diagrams can be filtered, and elements can be replaced with graphic icons to create more interest for non-technical audiences. For more information see the Model Diagrams Help topic.

This is the real power of the Model Based Systems Engineering approach, as it allows a system to be viewed in a multitude of ways, from a complete and high level view down through multiple layers of decomposition or hierarchy. Each level is connected, gaps or breaks in the models can be easily identified, and remedies can be found.

Ensuring a Model's Quality

The quality of a model will ultimately be reflected in the quality of the system that it represents. Enterprise Architect has been designed to provide a platform for the creation and management of high quality collaborative models. There are a number of features that assist the modeler attain the required level of quality, including facilities such as:

- *Metamodel* - that can be defined by a user to effectively create a grammar for the model, ensuring that users create 'compliant' modeling sentences; for more information see the Developing Profiles Help topic
- *Model Validation* - that allows the model to be checked for compliance with the underlying metamodel
- *Discussions and Chat* - that allow modelers to work collaboratively on a problem or solution
- *Reviews* - that allow internal or external experts to view and critique models
- *Model Patterns* - that provide expertly created model Packages to provide a starting point for modeling
- *Searches* - that assist in finding particular problems in the model



This diagram shows the Model Patterns in action, where novice and experienced modelers alike can, with a single button press, create well formed models and diagrams using a productive corpus of industry best practice models - all SysML compliant. There is also a detailed explanation and discussion on how to use the pattern, where to get further help and more. For more information see the The Model Builder Help topic.

The Systems Modeling Language (SysML)

The Systems Modeling Language (SysML) has been defined for the purpose of representing the Artifacts of Systems Engineering problems and solutions or programs of work in a consistent, efficient and robust way.

SysML is designed to provide simple but effective constructs for modeling a wide range of systems engineering problems and solutions. It can be used for a variety of purposes but is particularly effective in specifying requirements, structure, behavior, allocations, and constraints on system properties to support engineering analysis including parametric analysis and simulation. SysML can be used with multiple processes and methods such as structured, object-oriented, iterative, waterfall and many others.

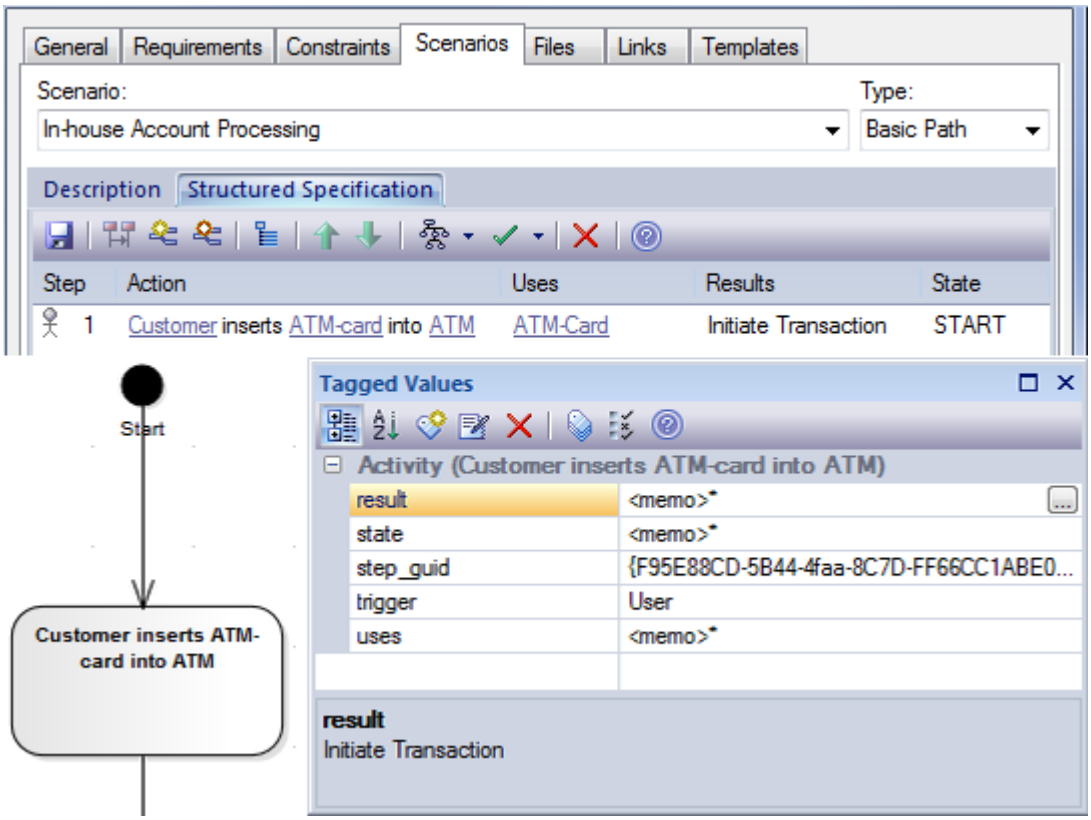
The language has been designed and augmented over more than ten years to be suitable for modeling systems of an ever increasing complexity. These changes have seen a relatively compact and concise language become broader and more diverse; nevertheless, the majority of systems engineering projects can still be modeled with a smaller part of the language, which we might term 'Core SysML'. For more information see the Modeling Systems in Enterprise Architect Help topic.

Enterprise Architect the Modeling Tool

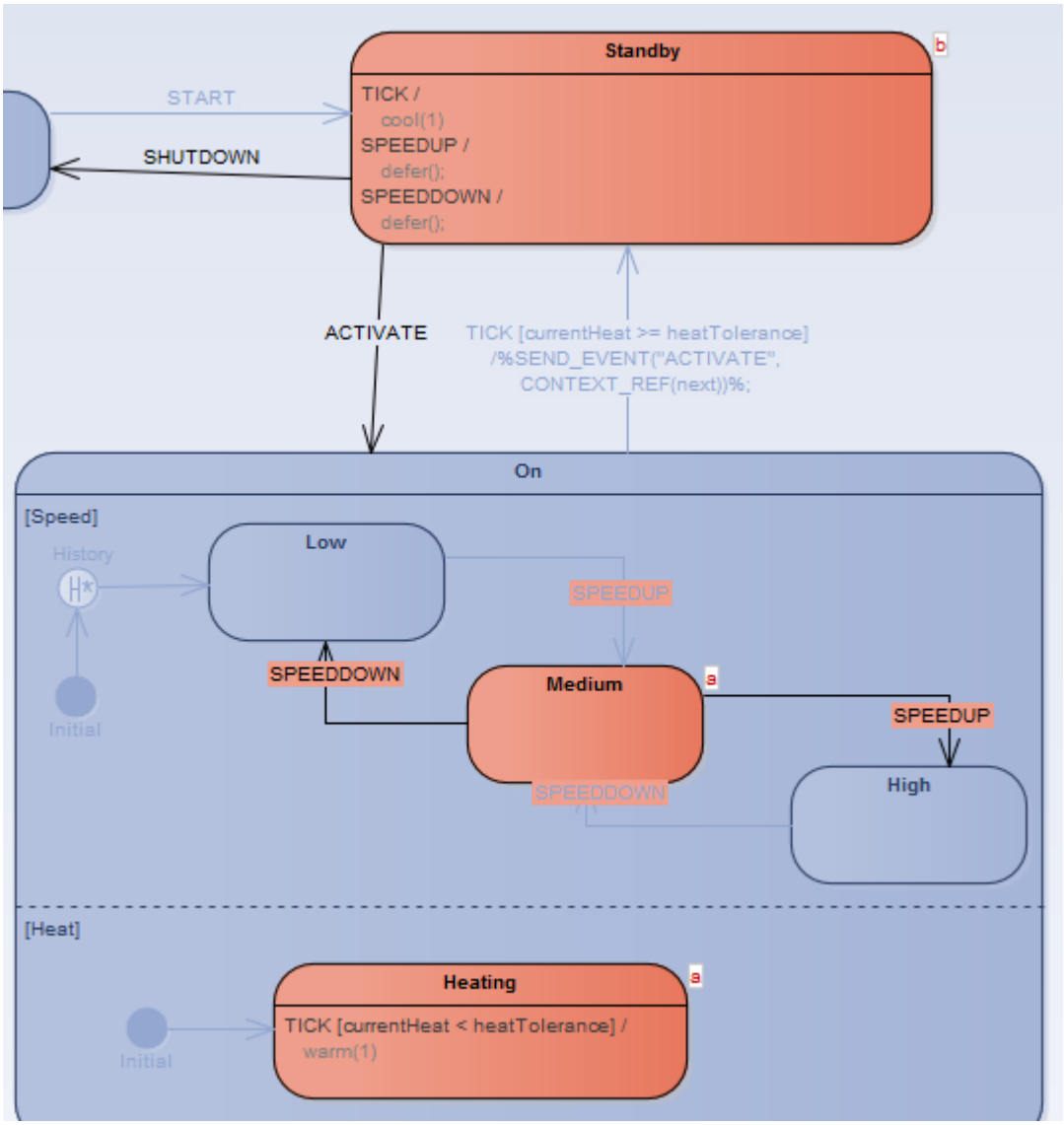
Enterprise Architect is both a Model Repository and a Collaboration Platform, making it an effecitve tool for Model Based Systems Engineering projects. It enables team members - including project sponsors, engineering managers, customers and engineers - to collaborate on projects in a rigorous and productive environment. Using WebEA and Prolaborate, the collaboration can continue on mobile devices such as mobile 'phones, Tablets and Notebooks.

In the information and innovation age, a tool is required to do a lot more than store information or allow users to view diagrams and models. Enterprise Architect has taken up this challenge and propelled its Systems Engineering offering to another level, with tools such as the:

- *Scenario Builder*, which automatically creates Activity diagrams from Use Case steps, and generates Test Cases from Scenarios; for more information see the Scenario Builder Help topic

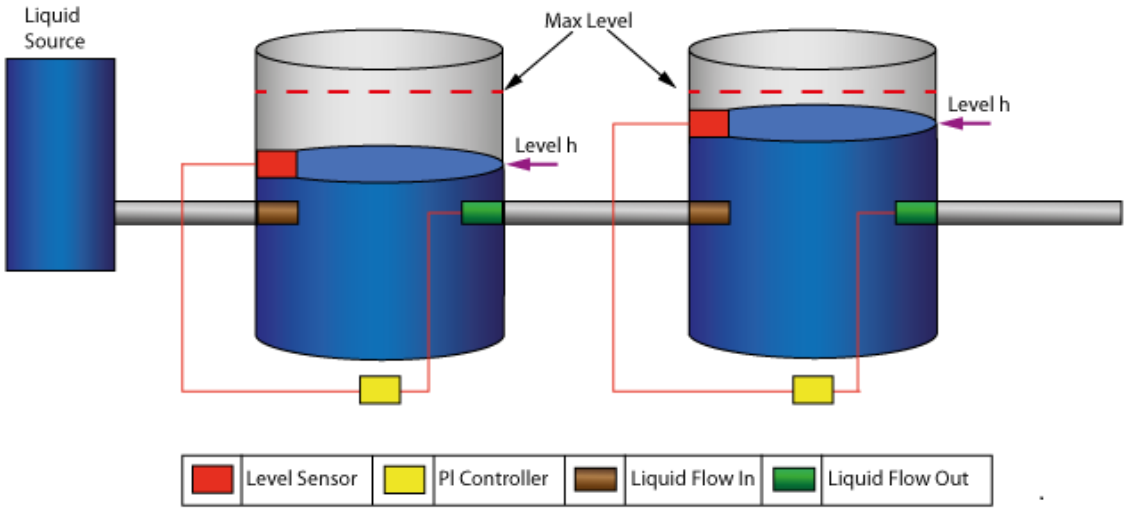


- *Executable StateMachines,* which allow programming code to be automatically generated from StateMachines; for more information see the Executable StateMachines Help topic

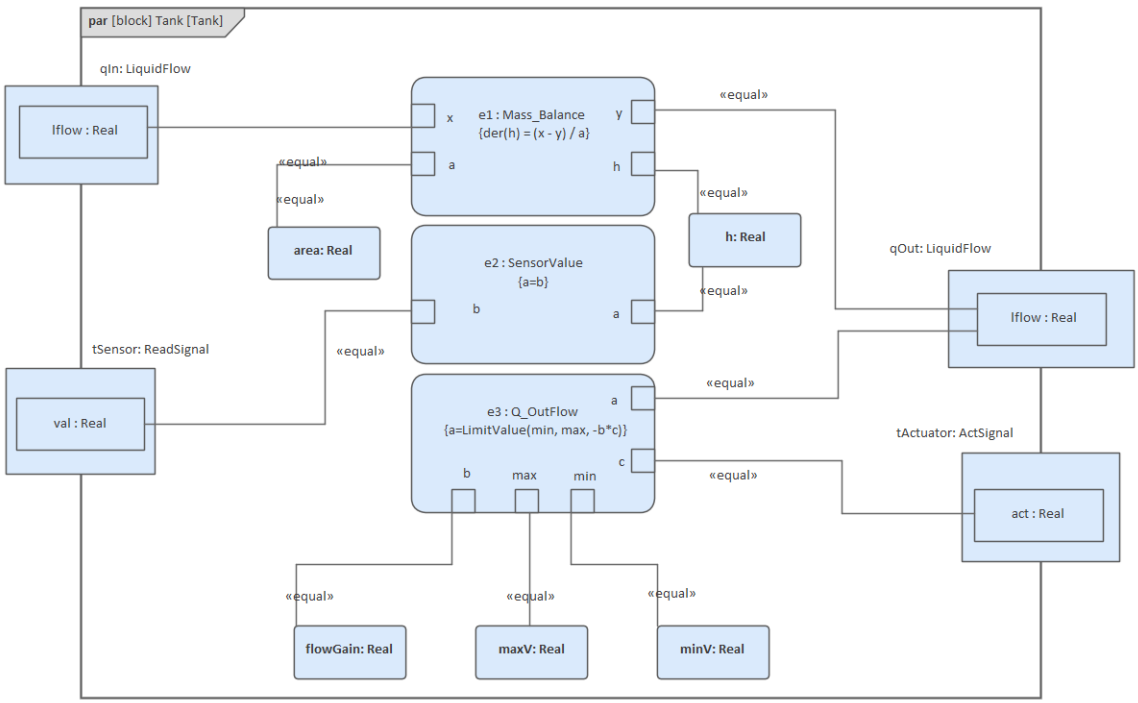


- *Parametric Simulations* using OpenModelica or Simulink, bringing models to life and allowing complex and often intractable problems to be visualized and analyzed to support trade-off analysis and engineering investigations; for more information see the Simulation Help topic

In this example the relationships between the factors controlling a fluid flowing between two tanks are defined:

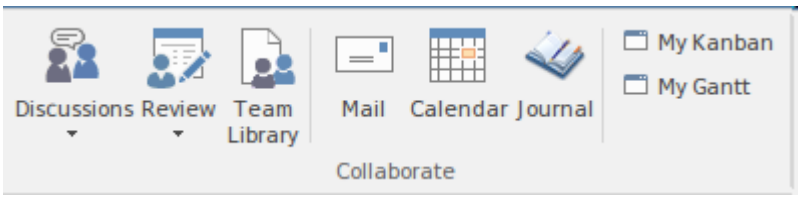


The models can then be simulated using the advanced OpenModelica simulation features.



Collaboration Platform

The information age has been transmuted to the innovation age almost while we were sleeping, and now more than ever there is an imperative that teams will work together in new and cohesive ways. Sharing documents and files on disk and working with static diagrams are now things that we expect to see in museums. Responsive, robust and innovative solutions can only be achieved by teams working with exceptional tools that not only allow models to be constructed and facilitate collaboration, but also perform work. Enterprise Architect is a multi-featured toolkit that allows teams to collaborate, bringing together the best minds and most experienced hands from a wide range of interlocking disciplines. The people that contribute to update and view the models might be dispersed geographically, operate in different time zones, be from different organizations or even speak different natural languages.

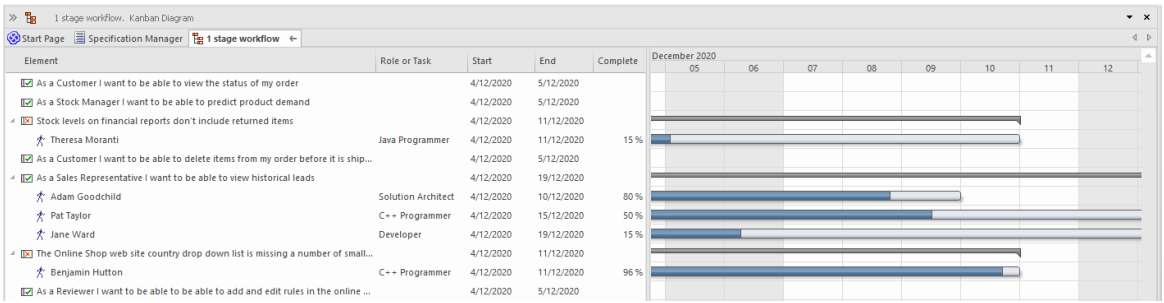


This image shows some of the useful collaboration features available from the Start ribbon. Discussions and Reviews are also available from WebEA and Prolaborate, allowing modeling and non-modeling staff to collaborate, resulting in more robust and fit-for-purpose solutions. For more information see the Teams & Collaboration Help topic.

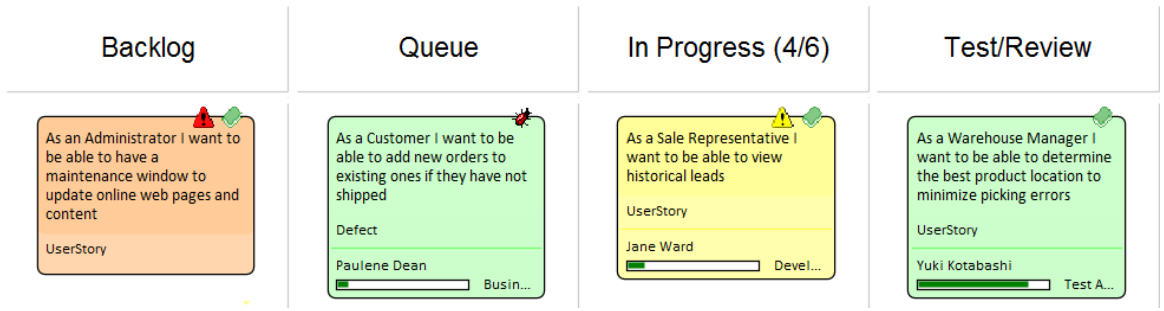
Project Management Workbench

Enterprise Architect provides a wide range of tools for managing a Model Based Systems Engineering project. In this way it can act as a project management workbench that can be used to manage an engineering project. The entire Systems Life Cycle can be modeled in the tool, from the conceptualization of business needs through design, implementation, utilization, support and ultimately to the system's retirement or disposal.

There are Gantt charts, Calendars, Model Libraries, Risk, Defect, Task, Effort and Metric Registers, to name a few. Roadmaps are another valuable feature that allow a project manager to visualize the development of a project over time from a current state to any number of transition or future states.



A team can also work cohesively using the built-in Kanban boards which allow items such as Requirements, User Stories, Defects and Changes and more to be visualized as they are actively being worked on. Resource allocation and Properties such as Priority and Status can be viewed through the board items and over-fill limits are displayed.



This proven technique, which has its origin in the Japanese Automotive sector has been implemented in Enterprise Architect in a way that will greatly enhance the productivity of your team and its project management. For more information see the Project Build & Deploy Help topic.

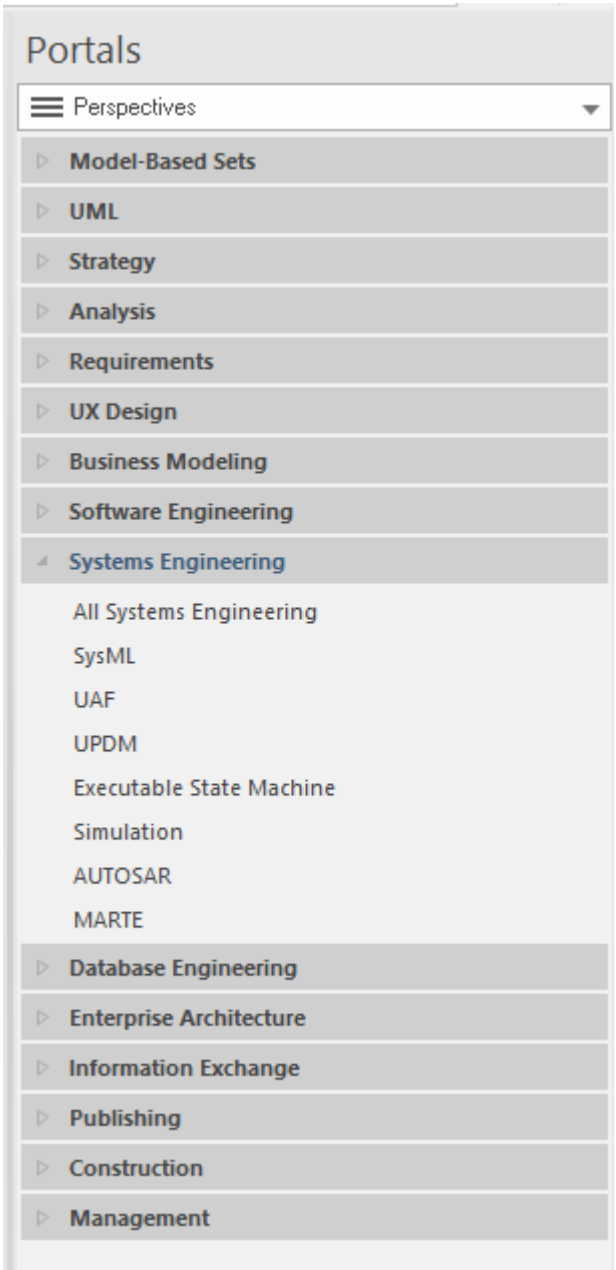
Model Repository

Enterprise Architect is primarily a model repository that allows models to be managed from their creation through to their retirement. The repository is stored in a relational database that can be hosted in a client server configuration or as part of a Cloud services facility, either on or off premise in a Cloud environment. So even though modelers will be working with diagrams and visual elements, these diagram are all codified and stored in the repository database. The repository can contain any number of models and can be organized for reuse and for enterprise and project models. For more information see the The Model Repository topic.

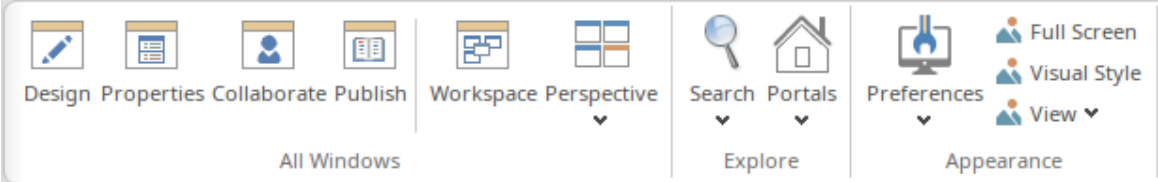
Getting Started

Getting started with a new tool is often one of the most difficult challenges, but Enterprise Architect makes this easy by providing a number of facilities to assist the newcomer to the tool. Enterprise Architect is a large, multi-featured application and the breadth of its coverage might overwhelm a person new to the program, but fortunately a solution to this has been built into the design.

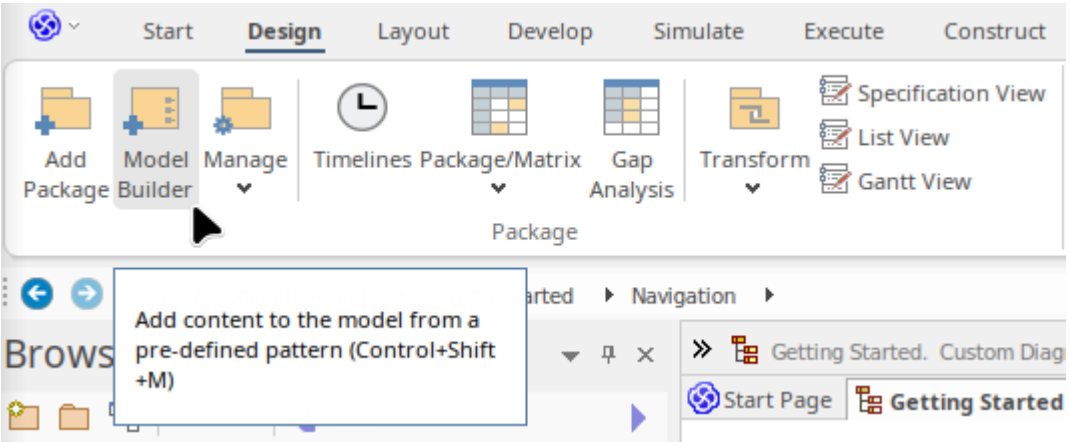
Perspectives can be used to limit the functionality to a specific area, such as System Engineering, making it easy for a System Engineer or Manager to get started. A user still has the ability to utilize other functionality that might be useful, such as Strategic Modeling, Mind Mapping, Code Engineering and more, simply by changing Perspectives, all without having to open a different tool. It is worth noting that Perspectives exist for a wide range of modeling disciplines that Enterprise Architect supports. For more information see the Model Perspectives Help topic.



A user also has tremendous flexibility in tailoring their own environment and the user interface by setting preferences and selecting workspaces and visual styles. For more information see the Advanced Customization Help topic.



Setting up a new project is straightforward with the use of the Model Builder Patterns (with accompanying documentation) that can be used to automatically create an MBSE project structure to get you started. The Model Builder can then be used to create any number of SysML diagrams as the model is developed and the problem and solution spaces are fleshed-out.

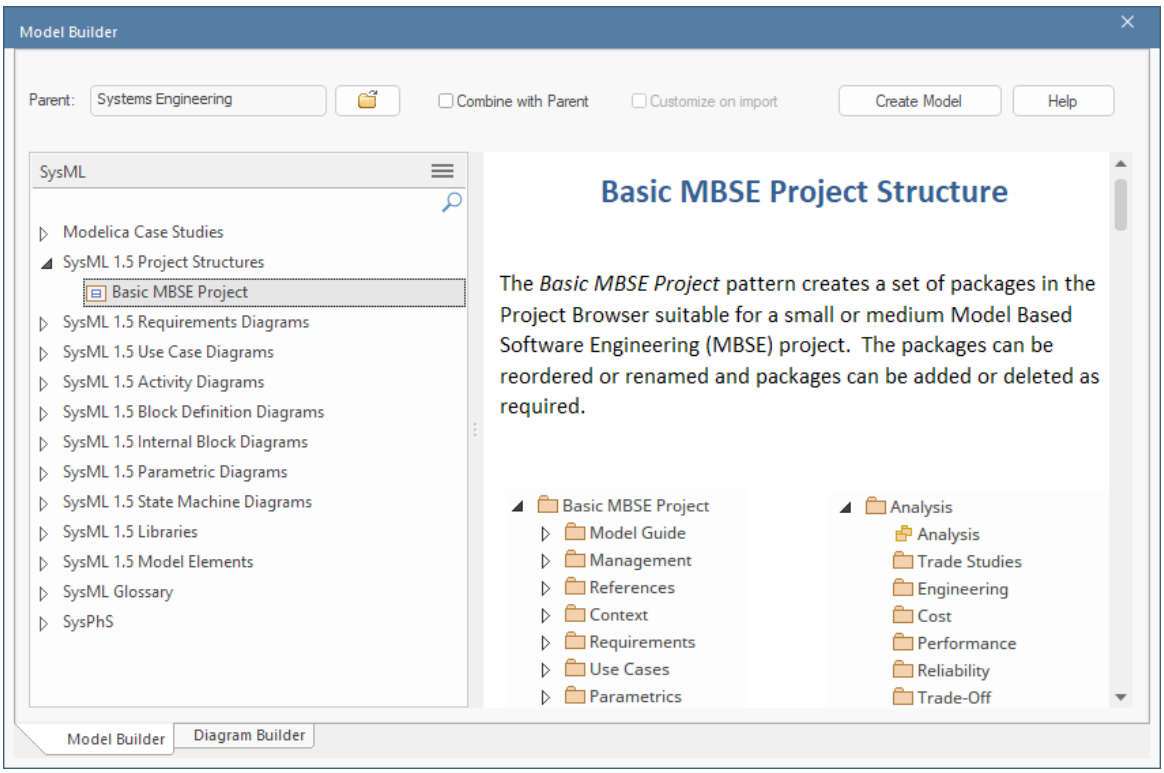


All of these facilities make it easy for a newcomer to get started, helping them to become productive members of a team and start contributing to models quickly and without any delay. A novice engineer will be surprised at how productive they can be when compared to working in text-based or other more rudimentary modeling tools. There will be challenges along the way as you push yourself and the tool to new limits but an in-depth Help system, a large community of users, comprehensive forums, a community site and first-class support services will make the journey easy and informative.

Setting Up a Model Structure

Enterprise Architect has been designed as a productivity tool from the ground up, and setting up a model structure - sometimes a daunting task for the beginner and tedious for the experienced user - is made simple in Enterprise Architect by the use of the Model Builder.

The structure for a new initiative (project) can be created using the Model Builder, which will create an entire project structure that can be tailored on import, providing all the Packages ready to start the project.



The structure of the repository is a subject that is explored in a later topic, because it is critical to the success of a model-based engineering approach to Systems Engineering. We will learn later that Packages are important units in the organization and maintenance of a model repository, and there is an entire topic dedicated to the subject of using Packages to structure the repository. For more information see the The Model Builder Help topic.

Tailoring the Application for MBSE

Enterprise Architect is a tool with a vast amount of functionality, which is one of the reasons why it is so popular as a tool for modeling systems of any kind. To ensure that the tool provides the most benefit to an organization, team, project or individual, some tailoring of the interface to suit the modeling intent will ensure that all parties achieve the best outcomes. Most of the settings can be changed by a single button click, transforming the tool to be fit for purpose, which - for us - is collaborating on Model Based Systems Engineering projects.

We will look at a number of places where we can change the application from a generic modeling tool to a systems engineering tool. We will look at these topics.

Selecting a Perspective

Selecting a Perspective is similar to putting a filter on an optical lens. It allows a modeler to just see application facilities relevant to that Perspective - in our case the Systems Modeling Language (SysML).

Selecting a Workspace

Selecting a workspace is important because it allows the user to ensure that the windows, ribbons, toolbars and other visual elements provide an efficient working environment and easy access to the important facilities that are needed.

Setting Visual Styles

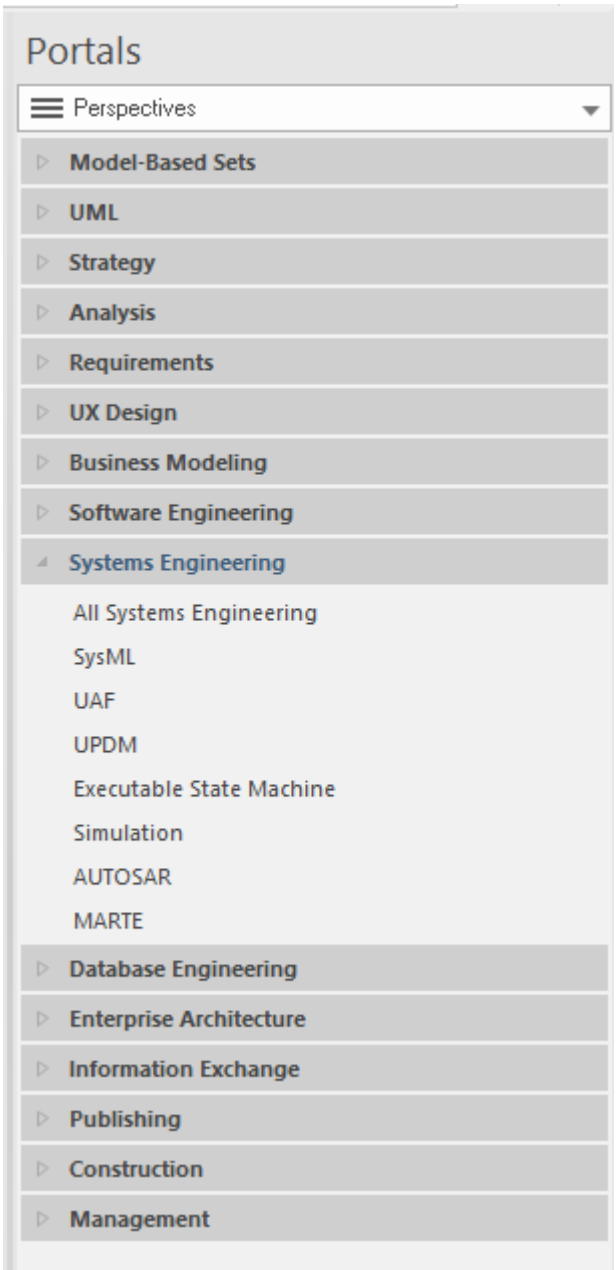
Visual styles provide a series of options for the look and feel of the application, including things such as colors and tab positions.

Setting Preferences

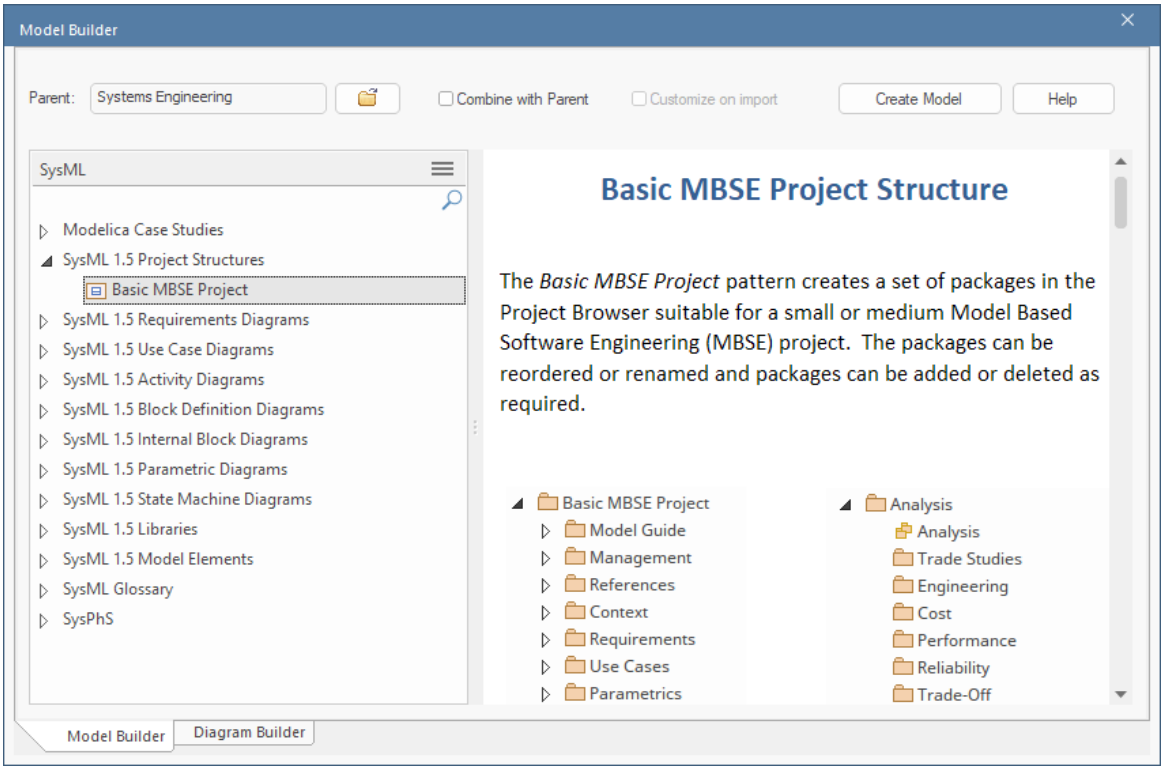
Preferences provide a wide range of options for tailoring how Enterprise Architect looks and functions from General settings such as Browser window options to Diagram, Objects and Engineering options. Many of the options apply to an individual user with others pertaining to the entire repository.

Setting a Perspective

Enterprise Architect is a tool packed with features for a wide range of disciplines, methods, languages and frameworks. Perspectives provide a way for a user to select a facet of the tool that allows them to focus on a particular subset of the tools features and facilities. The Systems Engineering group of Perspectives provides a natural starting point for Systems Engineers, but at any point if you decide to use other facilities in the tool you can simply change Perspectives and the tool will change to provide a focus on the selected area.



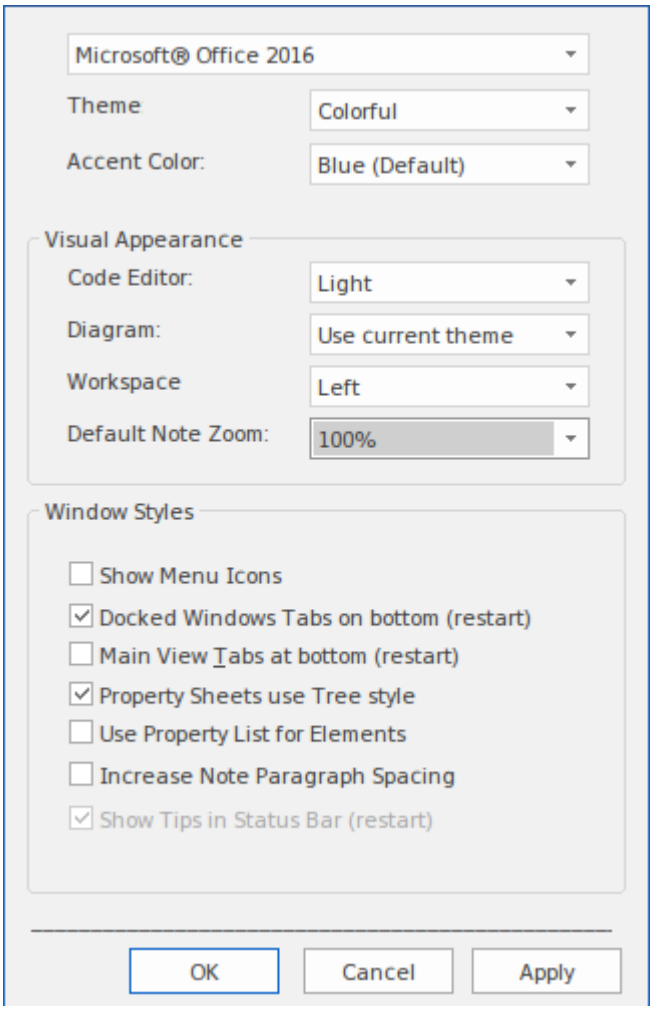
Selecting one of the Systems Engineering Perspectives will change the tools to focus on the selected aspect of Systems Engineering. For example, choosing the SysML Perspective will display a series of model patterns giving a user a jump start by being able to load a pattern for a standard model fragment or diagram. The 'New Diagram' dialog will also just display SysML diagram types.



There is also the convenient facility for a user to create any number of their own Perspectives, adding sets of technologies to each Perspective. This allows a modeler whose primary concern is SysML diagrams to add other facilities such as strategic models, Kanban diagrams and dozens of other useful diagramming and modeling mechanisms. For more information see the Model Perspectives topic.

Selecting a Visual Style

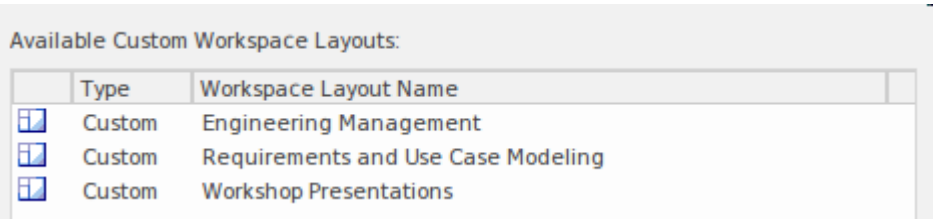
Every modeler will have their own preferences about the color scheme and style of the user interface, and Enterprise Architects allows these to be set and saved for each user, making the application more appealing. For example, some modelers will want a dark color scheme and others will prefer a light or colorful scheme.



There is a range of options here, including setting the position of the main window tabs, the size of the text in the notes window, and much more. Setting the visual style will assist in personalizing the modeling environment and making individual modelers feel comfortable while maintaining consistent and rigorous models. For more information see the Visual Styles Help topic.

Selecting a Workspace

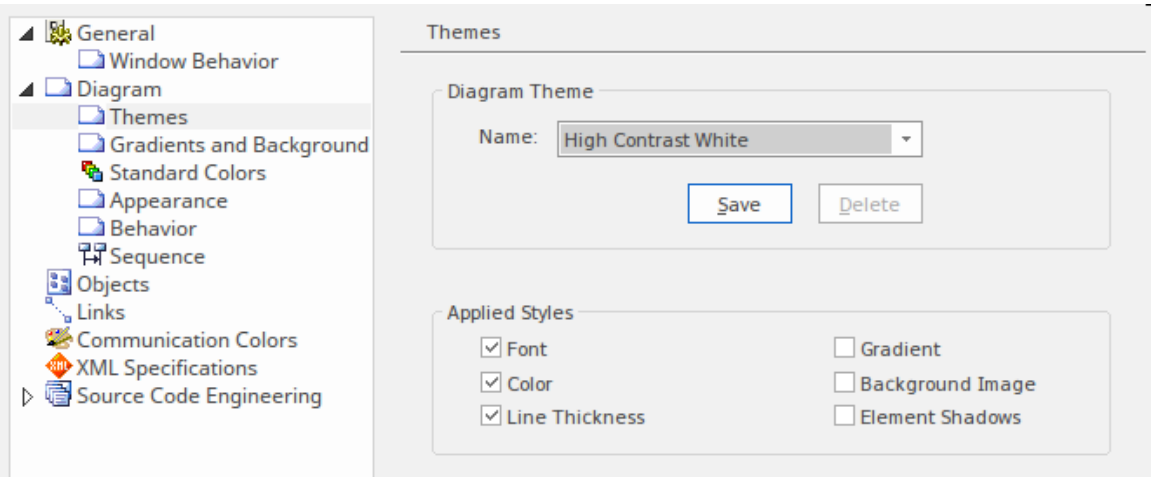
Enterprise Architect has a helpful way of quickly changing the layout of the User Interface to facilitate particular tasks or ways of working. This is achieved by simply selecting a workspace that will change the visible windows and tools, to provide the most efficient way of working to suit the task. For example, there is a workspace defined for Systems Engineering Simulations, one for Use Case Modeling, and another one for Testing. You can also define any number of your own workspace layouts that you find useful, by opening windows and tools and positioning them in an arrangement that facilitates working on a particular task or set of tasks, and saving them. In this example, a modeler has defined three custom workspace layouts. For more information see the Workspace Layouts topic.



Setting Preferences

Enterprise Architect has a formidable set of preferences, some of which can be set for the entire repository and others for each user. These allow the application to be tailored to suit an individual engineer or an entire team. For more information see the User Preferences topic.

This diagram shows how diagram themes can be set and elements of the style can be specified, including fonts, colors, line thickness and gradients.



Importing Existing Material

When Enterprise Architect has been set up for use it is likely that you will still have some existing project artifacts in the form of diagrams, documents, spreadsheets and items in other formats. Many of these can be conveniently imported into Enterprise Architect or referenced from within the tool. The tool also provides extensive server-side connectivity to a wide range of other tools, including Requirements Management tools such as DOORS Next Generation, project management tools such as Wrike, and project implementation tools such as Jira, via the Pro Cloud Server (a separately licensed server-side component).

Importing Spreadsheets

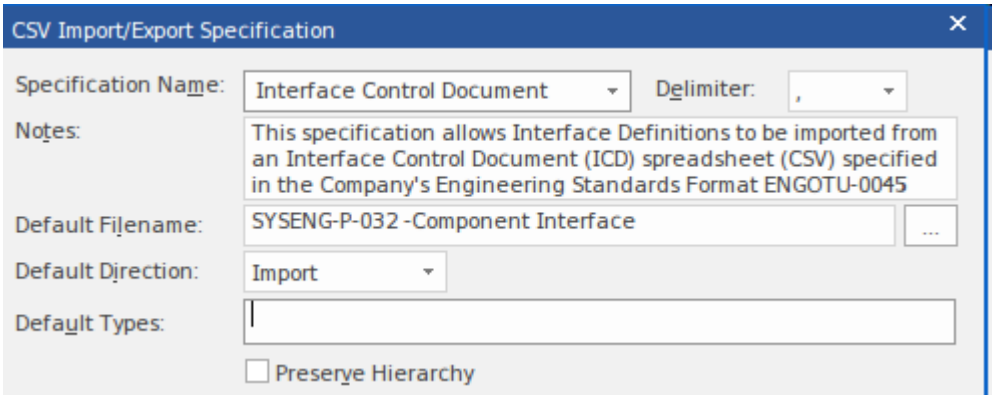
Spreadsheets are commonly used as a general purpose container for a wide range of numerical and textual project data. Uses include:

- Requirements
- Stakeholder Analysis
- Planning
- Roadmaps
- Subsystems
- Components
- Interface Definitions
- Task Management

While the spreadsheet is a very familiar tool it lacks many of the rigorous and useful features of an information management platform such as Enterprise Architect, including:

- Collaboration,
- Diagramming
- Traceability,
- Baselines,
- Visualizations,
- Simulations
- Versions and more.

Enterprise Architect has built in support for all these and many other forms of information that are commonly stored in Spreadsheets. The tool also conveniently comes with facilities to import and export the spreadsheet data using the CSV file format.



The most typical scenario is for the information in the Spreadsheets to be imported into Enterprise Architect and then the spreadsheet can be decommissioned and the information assets can be managed in Enterprise Architect from that point forward.

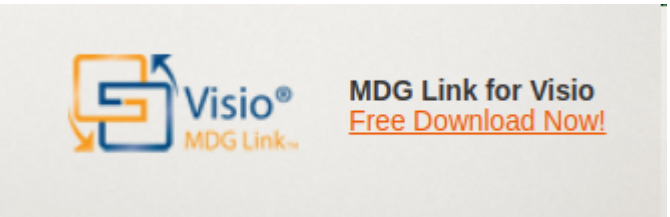
There might be situations where using the numerical analysis aspects of a spreadsheet would be useful, and Enterprise Architect conveniently provides a tool to export information to CSV file format for import into a spreadsheet. For more information see the Import and Export Spreadsheets topic.

Importing Visio Diagrams

Microsoft Visio is commonly used by engineering teams, often because there are no other more sophisticated tools available and it serves the purpose of creating general purpose diagrams. It is common for a team that adopts Enterprise Architect as their engineering platform of choice to have a collection of preexisting Visio diagram. All these diagrams can be imported into Enterprise Architect but the results are more effective when these Visio diagram have been constructed with consistency or using standard industry palettes rather than free-form geometric shapes such as square and circles that have no shared meaning. Once imported the diagrams can be massaged and updated to form part of the repository, and the original Visio diagrams can be decommissioned.

Enterprise Architect provides a free tool that can be used to connect to an MS Visio engine and import selected diagrams into the repository.

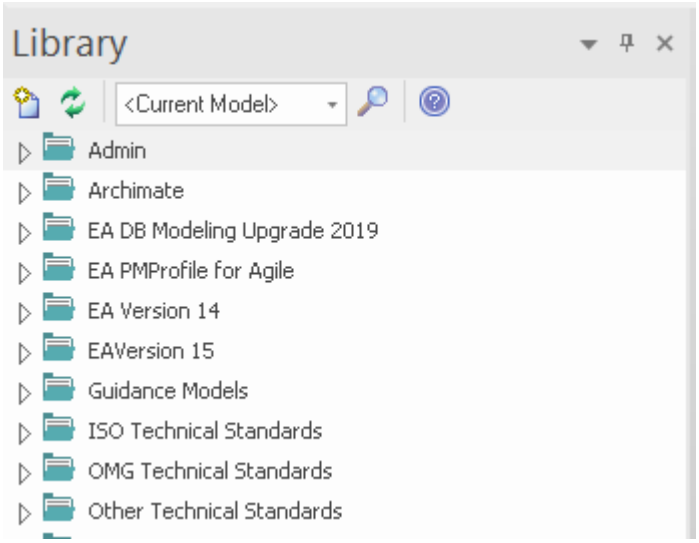
A decision has to be made whether to decommission the diagrams in Visio and to allow Enterprise Architect to manage the diagrams from this point on. Having the diagrams in the repository provides great power, as elements on the diagrams can be related to other elements in the repository. For more information see the Extensions - MDG Technologies the topic.



Using the Model Library

Even if an engineering team has transitioned to Model Based Systems Engineering it is likely that there will still be a range of documents and web based material that is critical for the management and development of engineering solutions. Enterprise Architect provides a pragmatic approach to this need by incorporating a Model Library feature where documents and web resources (both local and remote) can be collected together as references.

Any of the references catalogued in the Model Library can be included on a diagram as an Internal or External Artifact but more conveniently they can also be imported or referenced. For more information see the The Model Library Help topic.



Microsoft Office Integration

Enterprise Architect has the ability to integrate with the Microsoft Office suite of applications using the MDG Link for Microsoft Office, making it easy to exchange information between any Enterprise Architect model and MS Powerpoint, MS Word and MS Excel. There are options to import, export and synchronize the content.

Microsoft PowerPoint

PowerPoint integration provides easy access to Enterprise Architect's model repository within PowerPoint presentations. You can insert references to the model, use hyperlinked model element names, insert diagrams as images and tabulate Package contents on slides.

Microsoft Excel

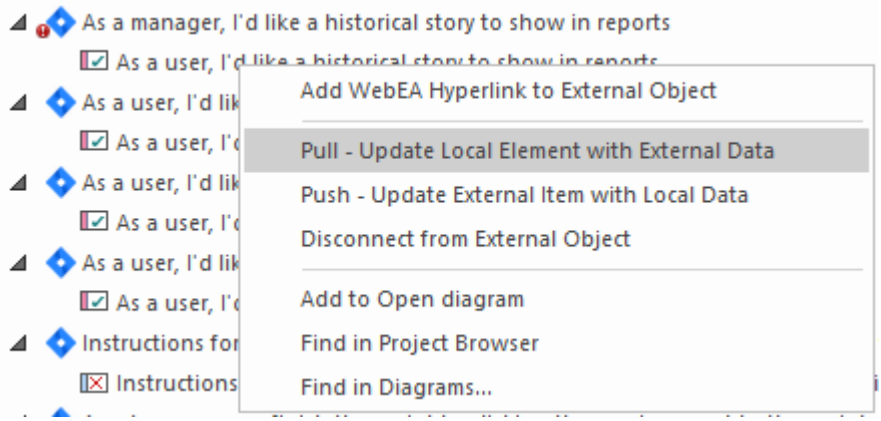
The Microsoft Excel Importer tool allows you to import contents from Microsoft Excel workbooks into Enterprise Architect as model elements. This includes importing spreadsheet data as UML elements, connectors, attributes and operations.

Microsoft Word

The Microsoft Word Importer tool brings Requirements, Use Cases, Processes, Classes and other data from Microsoft Word documents into Enterprise Architect as model elements. The Microsoft Word Importer provides a step-by-step approach that helps you map items such as sections, tables and delimited name-value pairs to Enterprise Architect elements and properties – including defining custom Tagged Values.

Integration with External Tools

Enterprise Architect provides an interface (as part of the Pro Cloud Server) for connecting your model repositories to external tools. This enables Enterprise Architect to synchronize elements in external tools with views of the elements in Enterprise Architect, which is particularly useful if Enterprise Architect and another tool share an interest in particular types of information. An example is the integration with the DOORS Next Generation (NG) product, where requirements modeled in DOORS can be viewed inside Enterprise Architect, and local surrogates of the elements can be placed on diagrams and related to any number of other modeling elements, including strategies, trade studies, Use Cases and Components. (There is also an in-model facility available to connect to older versions of DOORS.)



There is a wide range of integrations available, and teams can create their own integrations using the Open Services for Lifecycle Collaboration (OSLC) facility, available as part of the Pro Cloud Server. For more information see the Integrate Data from External Providers Help topic.

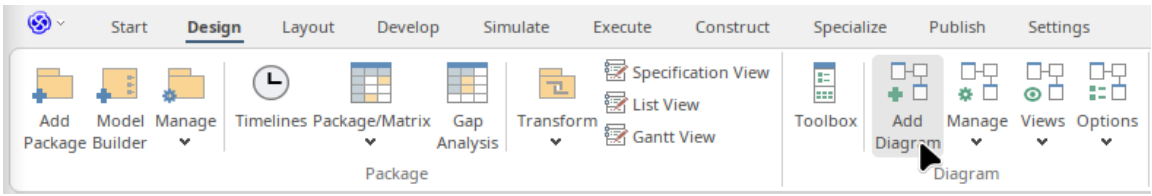
Creating Diagrams Elements and Relationships

Once a model structure has been set up and the application has been tailored to suit your needs, including the selection of the SysML Perspective and an appropriate workspace, you are ready to start creating elements and diagrams. While it is possible to create an element without first creating a diagram it is common practice to first create a diagram as a canvas for how the elements will be visualized. The first thing you will need to do is choose a location for the diagram in the Browser window. For example, you might be defining the fundamental architecture of your system and have defined a Package called 'Subsystems'. By selecting this Package you are telling Enterprise Architect that this is where you want a new Subsystems diagram inserted.

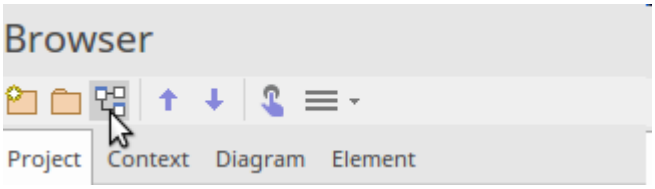
Creating a New Diagram

Enterprise Architect is a flexible tool and provides a number of ways of inserting a new diagram, including:

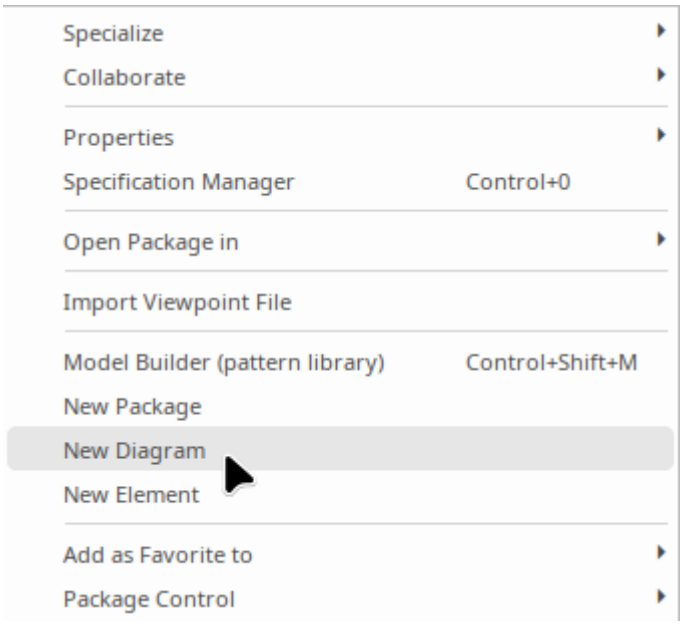
Selection from the Ribbons:



Selection from the Browser window header bar:

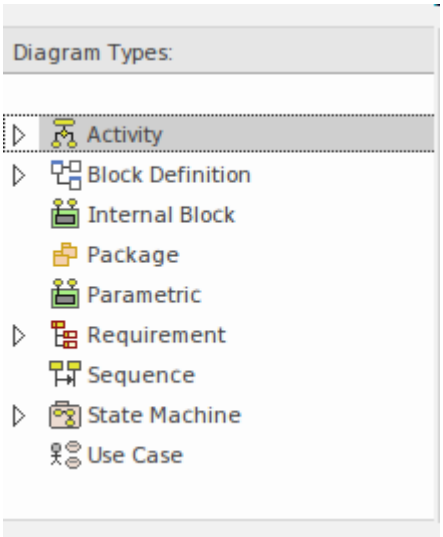


Selection from the context (right click) menu:



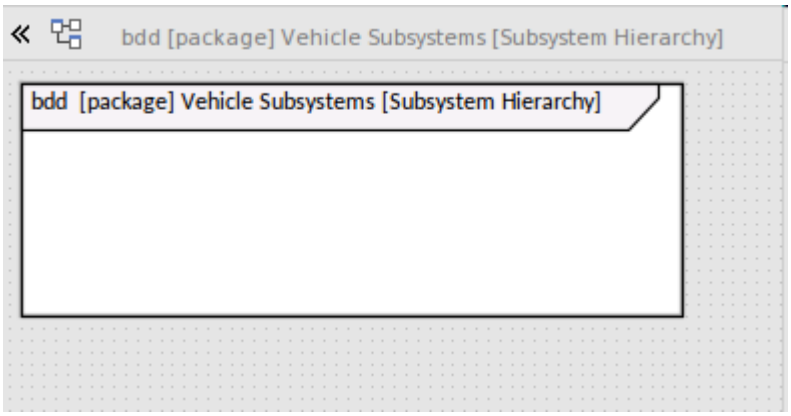
Regardless of the method you choose you will be able to select the SysML diagram type from the 'Diagram Types' panel

of the 'New Diagram' dialog.



Let's continue on to create a Block Definition diagram to represent the Subsystem. Select the Block Definition diagram as the diagram type and enter an appropriate name. Once you click on the OK button, a new (blank) BDD diagram will be created and the Block Definition Diagram Toolbox will be displayed ready for you, or a member of your team, to create elements and relationships.

Enterprise Architect will create a diagram canvas with a visible frame that represents the border of the diagram. The diagram frame is included because some users prefer to see it, but it can be hidden with no loss of meaning or compliance; once hidden the canvas then becomes the frame and the header information is contained at the top of the canvas. The frame can be included in saved or published diagrams by choosing this option in the 'Preferences' dialog.



Adding Elements to a Diagram

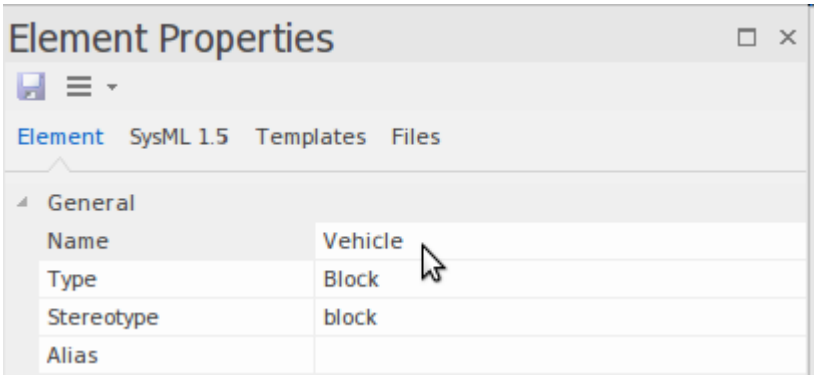
With the new diagram opened you are ready to start creating elements and relationships to describe the subsystems. There are essentially two types of Object that can be added to a diagram:

- New elements - *Created by dragging an item from the Toolbox and dropping it onto the diagram canvas*
- Existing elements - *Placed on the diagram by drag-and-dropping an element from the Browser window*

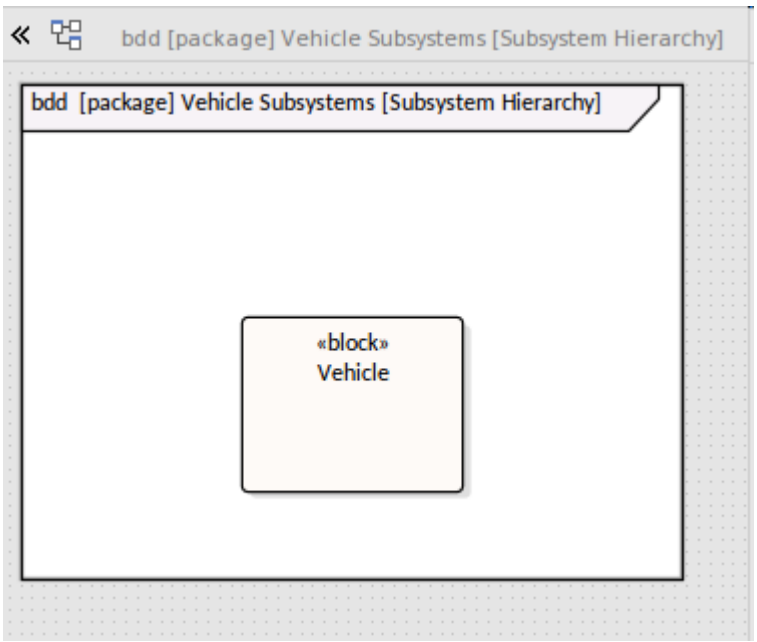
If you are starting a new project and have just set up your repository, you will not typically have elements in the Browser window so you will make more use of the first option and create elements from the Toolbox. As your project progresses it will become more common to use the second option and drag existing elements from the Browser window.

We will create a number of Blocks. Firstly we need a Block to represent the entire vehicle, so we will drag and drop a Block item from the Toolbox onto the diagram canvas. The tool will resize the frame to include the Block regardless of where you placed it on the canvas. The element will be given a default name of 'Block1'. Now using the Properties

window, typically docked on the side of the diagram, change the element's name to 'Vehicle' by typing over the default name 'Block1'.



This will change the element's name in the Browser window and the diagram. Returning to the diagram you will see the newly added Block with the name 'Vehicle' enclosed in the diagram frame.



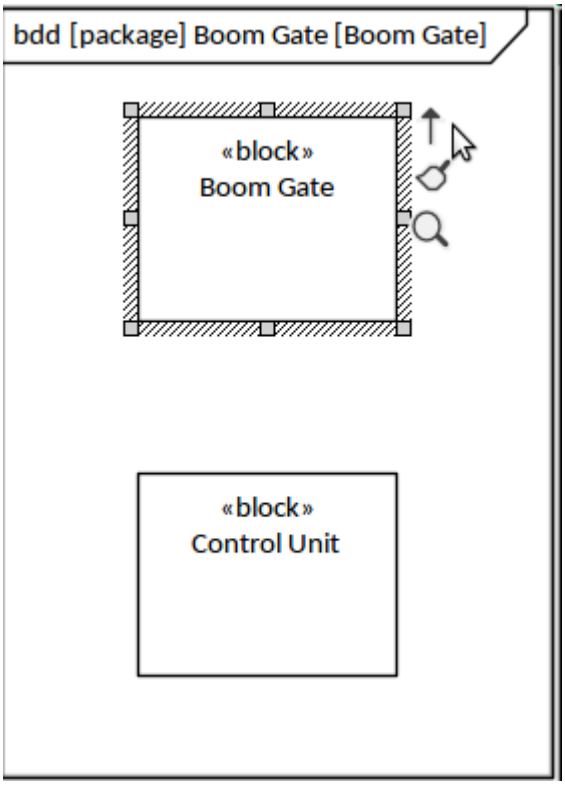
We could now use the same method to add a series of Blocks to represent each of the Subsystems.

Adding Relationships to a Diagram

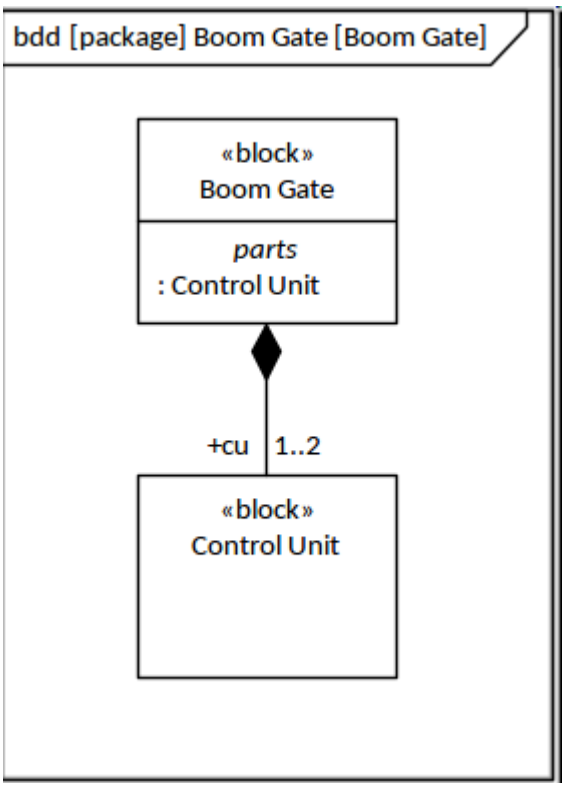
Once you have added two or more elements you can connect them with relationships, which provide the semantic glue between the different elements in the model. For example, a Block element can connected to another Block element using an *Part Association* relationship. There are two primary ways that connectors can be added to a diagram:

1. Quick Linker - *an intuitive diagram device initiated by dragging a link between the Quick Linker arrow (at the top right of the element) and another diagram object*
2. ToolBox Items - *connectors can be selected in the Toolbox and then dragged between two diagram objects.*

Either method will result in the specified connector being drawn between the two elements. Care needs to be exercised to ensure you are dragging in the right direction; the Part Association relationship, for example, should be dragged from the Block that is at the *part end* to the Block that is at the *whole end*. This will ensure that the little diamond marker at the end of the relationship is positioned at the correct end, indicating the whole-part relationship.



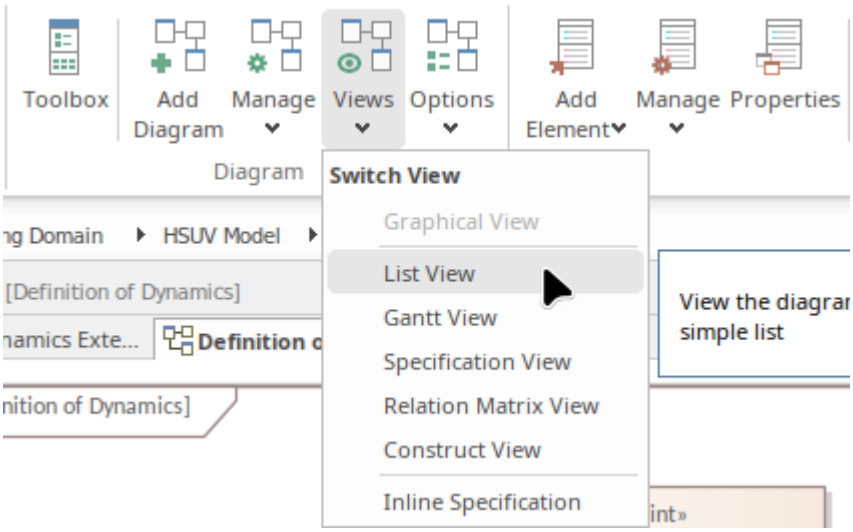
Regardless of the method that is used the result will be an Information Requirement relationship connecting the two Blocks. The direction and style of the connector can be altered, and any number of way-points can be added to route it differently as the model is developed. This diagram shows the added relationship where the modeler has also added a role name (+cu) and a multiplicity (1..2), indicating that a Boom Gate must have at least one control unit but could have as many as two. If a modeler were to inadvertently add the connector in the wrong direction it can be conveniently reversed by accessing options from the *Advanced* submenu of the connector's context menu.



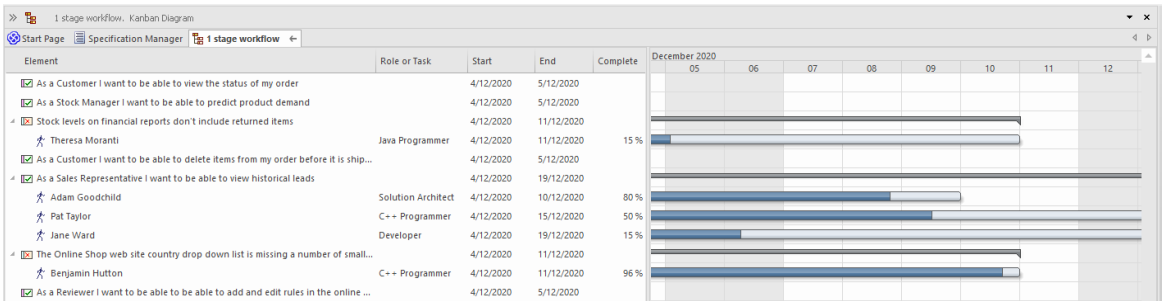
Visualizing the Models

You might have been introduced to the SysML by reading introductory material in textbooks or in lecture notes as part of a tertiary qualification or in online pages. Much of this content presents the SysML as a language of diagrams, but Enterprise Architect expands the horizons of the language - and modeling in general - by providing a variety of ways to visualize information in the repository. The diagrams still play an important part, but in many circumstances there are more compelling ways to visualize or work with the repository content, including:

- Specification View - a spreadsheet or document view for those who are more familiar working this way
- List View - a table-based view similar to a spreadsheet where the properties of multiple elements can be viewed
- Traceability View - a hierarchical and graphical view where element relationships can be viewed to any level of nesting
- Gap Analysis Matrix - shows source and target elements and gaps in architectural models
- Relationship Matrix - a grid based view with two axes containing sets of elements
- State Table View - a view available for StateMachine diagrams to display the state transitions in a table
- Gantt Chart View - view a timeline when resources have been assigned
- Searches - view element lists that comply with a built-in or user defined set of criteria
- Publications - view elements in a publication using built-in or user defined templates
- Graphical View - view the elements as a diagram (this is the default view)



This diagram shows a Gantt Chart that has been automatically created from the resource information entered against elements that were visualized on a Kanban diagram. For more information see the Gantt View Help topic.



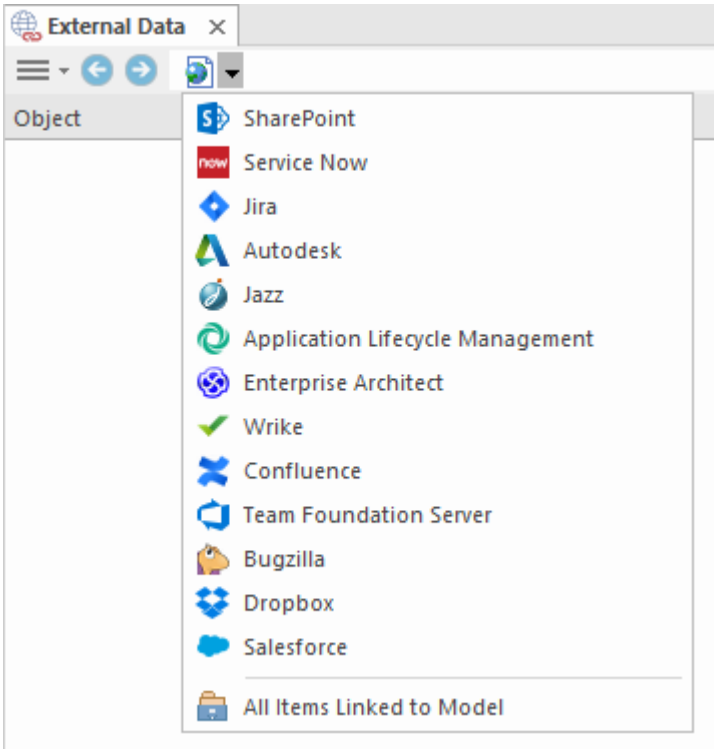
Synchronizing with External Data

While Enterprise Architect is a central tool for the management of Model Based Systems Engineering projects, there are likely to be a number of other tools that an engineering office will have in place or will acquire for the purposes of ensuring that an endeavor meets its outcomes. These might include Project Management tools, visualization tools, Requirements Management catalogues, configuration management systems and issue tracking software.

Enterprise Architect, through the Pro Cloud Server (a separately licensed server-side component), provides bi-directional integration with a large number of tools including:

- Doors Next Generation - used for Requirements Management
- Wrike - used for general purpose Project Management
- Jira - used for issue tracking
- ServiceNow - used for Configuration Management
- Share Point - used for document management
- Team Foundation Server - used for Version Control in the software discipline

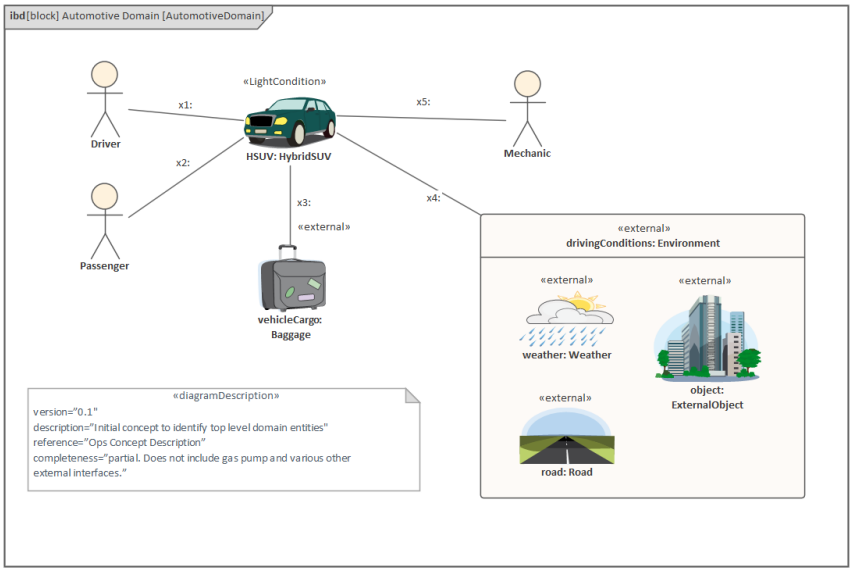
Representations of the elements from these external data sources can be included in diagrams and related to other elements in the repository. This allows Enterprise Architect to become an engineering information hub, connecting and integrating with a wide range of tools to provide a single view of a project. This image shows a current list of providers.



Where we are Heading

The discipline of Systems Engineering dates back to the early 1900's; the term is thought to have been first coined by Bell Laboratories in the early 1940's. Sixty-odd years would pass before the need for a dedicated language for modeling systems was perceived in 2001, but it wasn't until 2006 that the Systems Modeling Language (SysML) was adopted by the Object Management Group and became the language of choice for describing systems.

In the field of Systems Engineering, Enterprise Architect has become the tool of choice for many leaders in the industry, because of its flexible, extensible and pragmatic approach to modeling complex systems and its strong compliance with the Systems Modeling Language specification.



As a platform, Enterprise Architect offers a unique capability in supporting the integration of strategic, business, engineering and technology models, from motivation models through to the implementation of systems and continuing on to support. The tool helps the System Engineer to create Strategic models - including diagrams such as the Balanced Scorecard - Capability models, Tactical models - such as Gap Analysis and Roadmaps - and Operational models, appealing to stakeholders including the senior executives, engineering managers, solution and implementation teams, and engineers.

How it will help you

Readers will typically come to the topic of Model Based Systems Engineering with some existing knowledge or experience even if it is something that has been learnt in lectures or by on the job training, or perhaps by using a different tool. Readers will benefit by understanding Enterprise Architect's features and the tools that are available to develop and manage Model Based Systems Engineering models in Enterprise Architect. This knowledge will enable them to be more productive as an individual and also as a member of a team. The reader will also learn about the syntax and semantics of the Systems Modeling Language popularly know as SysML and how it can be used to model and even to simulate complex engineering systems.

Who will benefit

Anyone involved in the development, management or support of Model Based Systems Engineering initiatives whether at a:

- Strategic level,

- Business Value level,
- Management Level or
- Engineering level

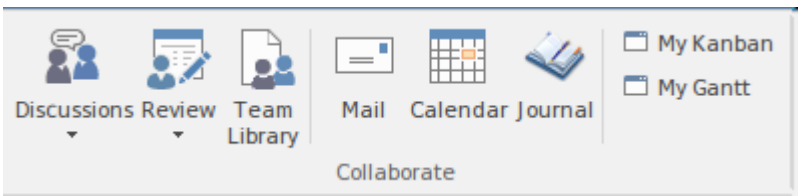
will benefit from reading this information. This covers a wide range of roles including:

- Customers,
- Strategic Thinkers,
- Senior Management,
- Engineering Management,
- System Designers and Architects,
- Software Designer and Architects,
- Systems Engineers,
- Software Engineers,
- Fabrication Teams,
- Implementation Teams,
- Support Staff.

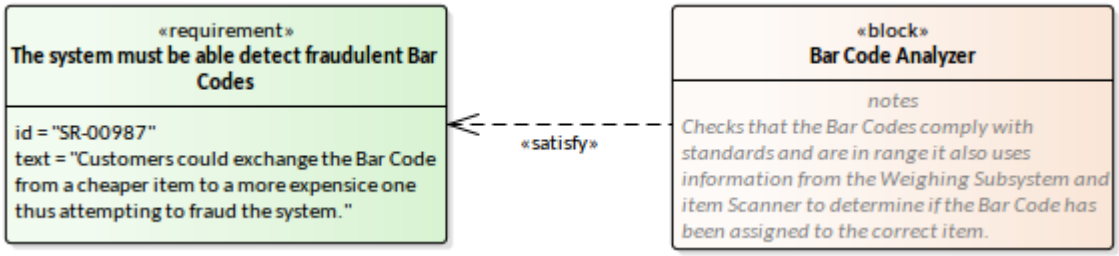
The real power of this way of working and using Enterprise Architect as a tool is in the unification of the interdisciplinary teams. Thus allowing people from a wide range of disciplines, potentially geographically dispersed and working at a wide range of levels from strategy to support, to share a common view of the product or system being developed.

What you will learn

This guidebook will teach you how to use the rich features of Enterprise Architect to develop and manage Model Based Systems Engineering initiatives, to write and read the Systems Modeling Language (SysML), create documentation and to work collaboratively as a member of a team using a formal or informal engineering process.



You will learn what tool facilities are available and which tools should be used to perform a particular technique and using links to help topics how to use them . For example, regardless of the process or framework that is adopted, at some point *Allocation between Structural and Behavioral elements* will need to be performed; this topic will describe the technique and how to best achieve it using the tools and facilities you have at your fingertips by adopting Enterprise Architect.



Overview of the Documentation

This table provides a list and a description of the subjects that are included in this work, giving an overview of the material.

An Equation with Four Variables	This topic describes Model Based Systems Engineering from a mathematical perspective as an equation with four variables namely: *Modeling, Enterprise Architect the tool of choice, the Systems modeling language, and an Engineering Process*. Elementary mathematics would suggest that we need four simultaneous equations to solve such a problem. This topic will introduce these four variables and suggest ways you as an engineer or manager can become proficient in this discipline without the need to solve the equations simultaneously. We will see how Enterprise Architect can play an important part in all four parts of the equation and has been designed to help make your engineering initiatives successful.
Getting Started	This topic introduces newcomers to the tool to the most important aspects of the application, including setting up a model structure, tailoring the application, working with diagrams, other ways of visualizing content and integrations with other tools. It also includes discussions on the user interface, including Ribbons and Perspectives which are fundamental to working with the tool. The tool has an extensive Help system that has been developed over more than twenty years and that can provide answers to almost any question that an engineer or other stakeholder might have while working in the tool.
Where we are Heading	This topic describes the list of the topics in the Guide, giving an overview of each topic.
Getting to Know the SysML Diagrams	This topic introduces the diagram as the preferred and most commonly used method for visualizing models and repository content. It gives a brief introduction to each diagram and allows the reader to build up a mental map of what the diagrams are and how that can be used both to express one's ideas and to interpret the ideas of others.
Systems Modeling Language Overview	This topic provides insight into the anatomy and the physiology of the Systems Modeling Language. It provides a useful synopsis of the notation and describes the language architecture. It also introduces the fundamental concepts including: Elements, Diagrams, Models and Views. While this information can be gleaned from the Specification itself it is summarized in this topic in enough detail to shield the newcomer from the need to wade through the specification. Enterprise Architect provides example models and an extensive help system that will act as useful resources to help the newcomers and experienced practitioners alike gain experience with the language.
Collaborating as an	This topic introduces the formidable collaboration tools where system engineers,

Engineering Team	managers, customers, consulting engineers, regulators and standards bodies can all contribute to models. This includes discussions, chats, model mail, a team library, reviews and more.
Using Packages to Structure the Repository	This topic covers the fundamental aspect of the repository structure and how it is used as an organizing principle to ensure that the repository is fit for purpose. It also covers how the Package can be used as a container that both allows content to be added but also provides a mechanism for the management of the elements, properties, diagrams and views that are added to each Package.
Requirements Definition and Management	This topic introduces the engineer and other stakeholders to this all important and central discipline that forms the basis for all other aspects of a modeling endeavor. Enterprise Architect has a formidable and unparalleled set of tools for developing, managing, visualizing and documenting requirements and these are introduced and practical examples are given on how the tool can be used.
Describing User Goals with Use Cases	This topic describes a user-centric way of articulating requirements where the goal that the user is trying to achieve is the foundation upon which a requirement is written. The users' goals are codified in Use Cases, which are represented simply on a diagram but the details of which are fleshed out in Enterprise Architect using the Scenario Builder. This facility allows the description, constraints such as preconditions and post-conditions, and the steps of the scenarios to be written in a compelling and productive user interface. It also describes how behavioral diagrams can be automatically generated from the tool, and the elements such as Activities can be linked to up-stream modeling elements such as Requirements and down-stream elements such as Components - creating effective and useful traceability.
Using Blocks to Model Structure and Constraints	The Block, which is introduced in this topic, is the fundamental unit of structure in the language and can also be used to model constraints. It is the atom of the SysML language and can have Features, Properties and Interaction Points that describe in detail the anatomy. This topic also describes the relationships that Blocks have with other model elements, including the all-important *Allocation* relationship that relates Blocks to Activities. Allocations tie together the two important pillars of the SysML: *Structure* and *Behavior*. It also introduces devices for modeling *Quantity* and *Value Types* that can be used to model dimensions in the physical world.
Using Properties and Parts to Model Block Usage	This topic follows on from the previous topic and introduces the Internal Block diagram, which is used to visualize how Blocks are used in a given context. These diagrams show how a Block's part properties can be connected together. The owning Block is represented as a diagram frame and the parts that appear at the part end of the Part Association on a Block Definition diagram appear on the Internal Block diagram as a Part element
Coordinating Behavior with Activities	This topic introduces Activities and the more atomic unit, the Action, which are both used to describe the behavioral aspects of a system at different levels. These all-important elements are equivalent to the verbs in our natural languages and, like verbs, have an organizing function in the model. Enterprise Architect has a number of useful devices such as Simulations that can bring these models to life and allow complex real world problems to be simplified and visualized.
Visualizing with Parametrics and Simulations	This topic explores the use of Parametric diagrams in connection with Block Definition diagrams, which define ConstraintBlocks that model mathematical equations and the parameters they use. The topic describes how these constraints and parameters can be represented on the Parametric diagram, which is a cousin of the Internal Block diagram. Simulation of the Parametric diagrams is also introduced and you will learn how to install and work with the OpenModelica interface. You will learn how advanced plots of equations can be visualized without

the need to leave the Enterprise Architect environment.

Modeling Change with StateMachines

This topic introduces the StateMachine as a method of describing the discrete conditions (States) that an entity such as a Block can exhibit. This behavioral device can bring great clarity to a model and solve otherwise intractable problems. Enterprise Architect has a number of tool features that can bring these models to life, namely the State Table and Executable StateMachines that allow the States and the Transitions to be active in compelling visualizations.

Interactions as a Sequence of Messages

In this topic the Sequence diagram is introduced as a way of modeling messaged based behaviors. You will learn how to use this diagram to model a range of engineering concepts. The diagrams model the interaction between Blocks that are internal to the system, or between the system itself and its environment, and they can be used to model the communication that occurs with the steps of a Use Case. Lifelines and Activations are studied and Message orchestration is introduced with the use of Fragments.

A First Example SysML Model

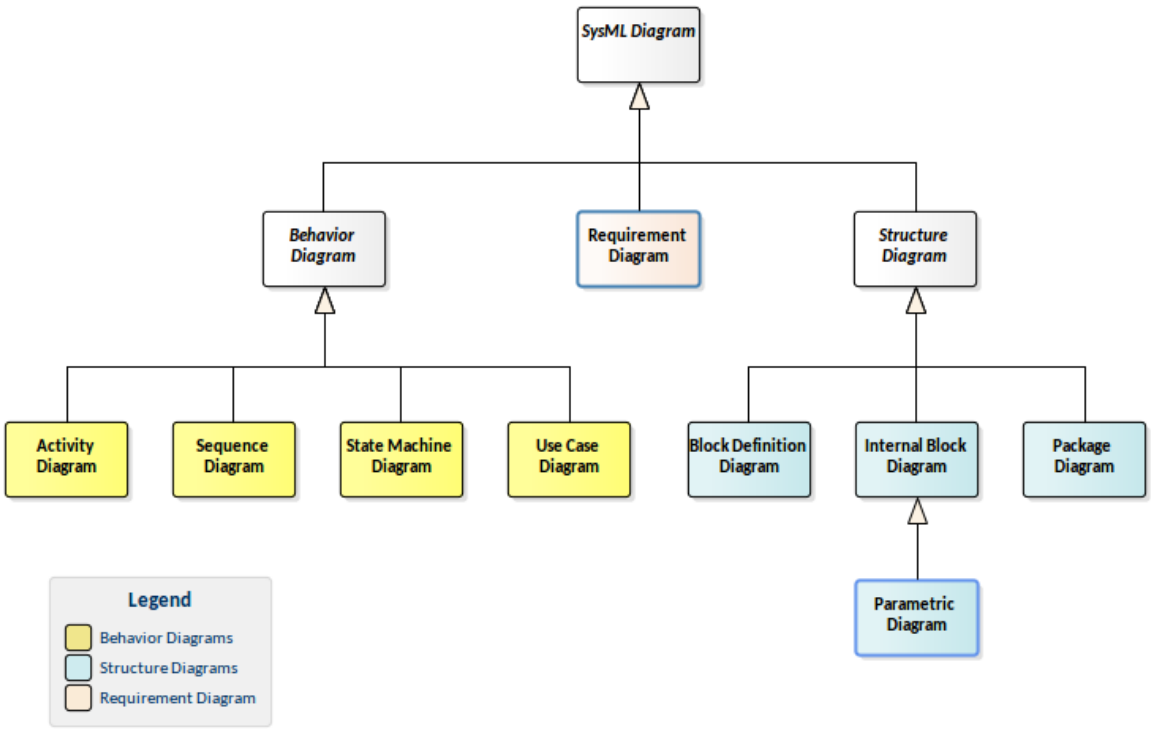
In this topic we explore the diagrams that specify, design and test a Hybrid SUV vehicle that utilizes fuel and electricity as power sources, featuring mechanisms like regenerative breaking to meet its fuel efficiency requirements. The example model is published in an annex of the SysML specification; in this topic we explore the diagrams that have been created in Enterprise Architect, explaining significant language and tool features.

Meet the Systems Engineering Tools

This topic introduces some of the most important tools for working with Systems Engineering models, describing what they are, where they can be found, and how they can be used, including options and where to find more help. There are many other tools that modelers might find useful; these are introduced at relevant points in the document.

Getting to Know the SysML Diagrams

The diagrams of the SysML can be regarded as types of canvas, where an engineer will create visual representations of the engineering concepts that form part of the model. There are nine SysML diagram types, each focused on a particular aspect of the problem or solution. While the diagram types typically contain different types of elements, they all conform to a standard representation composed of: a Frame that contains, a Header and a Contents Area.



This section describe some of the most useful tools and features that can be used when working with diagrams, but there are many others that can be helpful. For more information see the Model Diagrams Help topic.

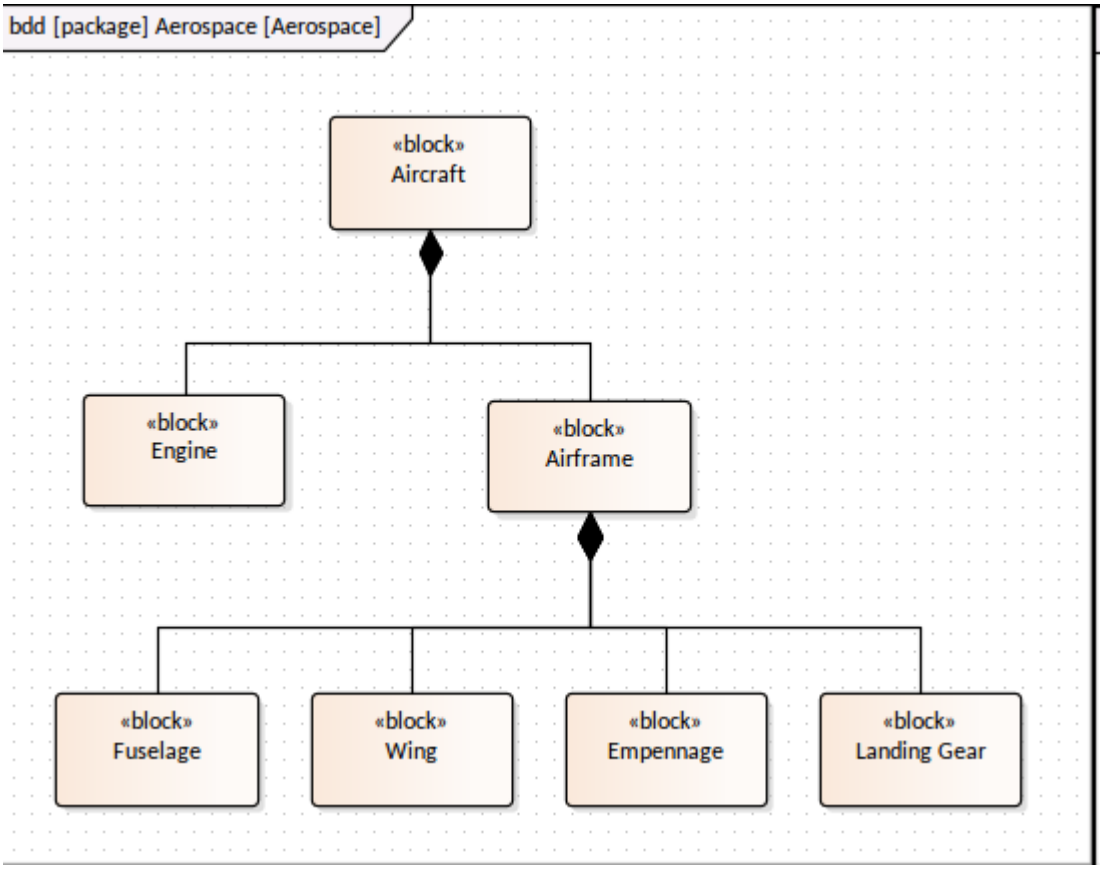
Diagram Frames

A Diagram Frame is a visual device that encloses the elements and relationships on a diagram. The frame has two parts:

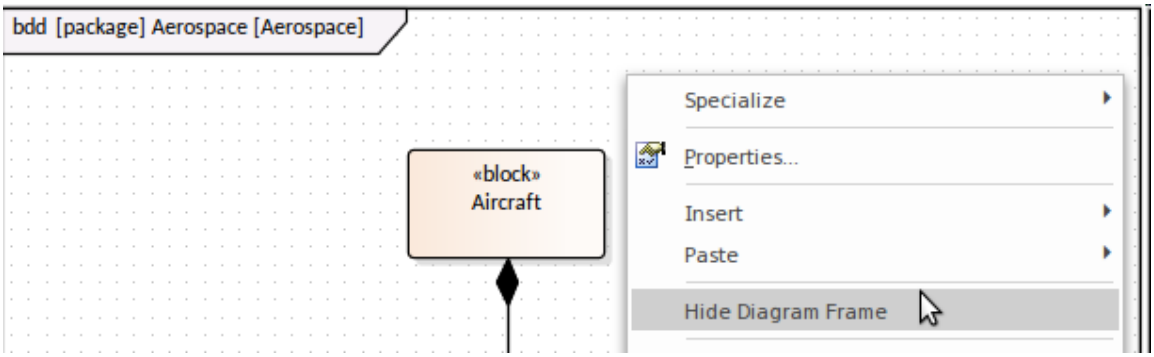
- A *Header Area* that contains a qualified name for the model element within the frame, which is provided if it is not contained within default namespace associated with the frame; it has the form:

  diagramKind [modelElementType] modelElementName [diagramName]

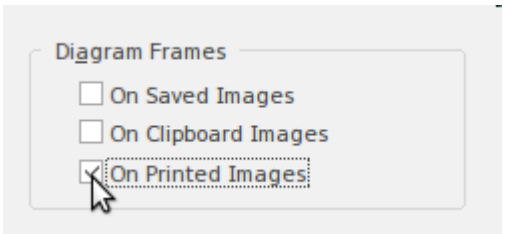
- A *Contents Area* that contains the visual elements that make up the diagram



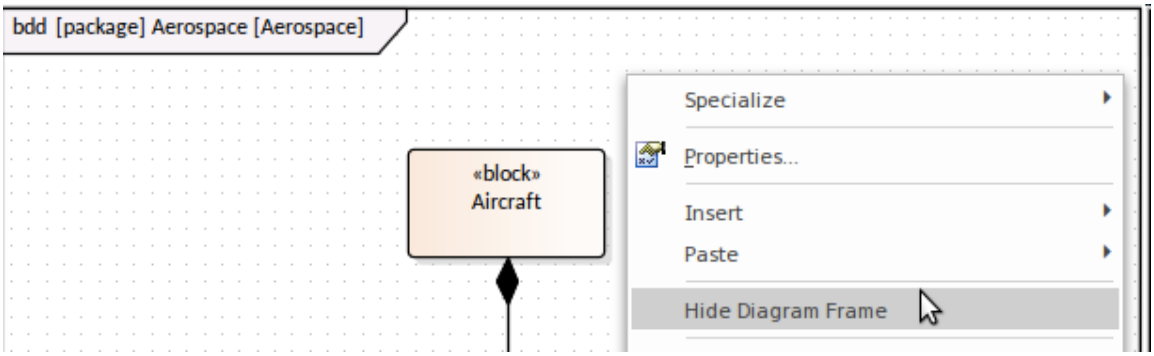
Enterprise Architect by default displays the frames within a diagram window, but in compliance with the SysML specification the frame can be suppressed to create a less cluttered diagramming interface. (Remembering that Enterprise Architect conveniently displays the frame header information in the diagram header).



A frame can be switched back on whenever needed, and diagrams being sent to the clipboard or printer can be configured to display frames regardless of whether they have been hidden in the user interface.



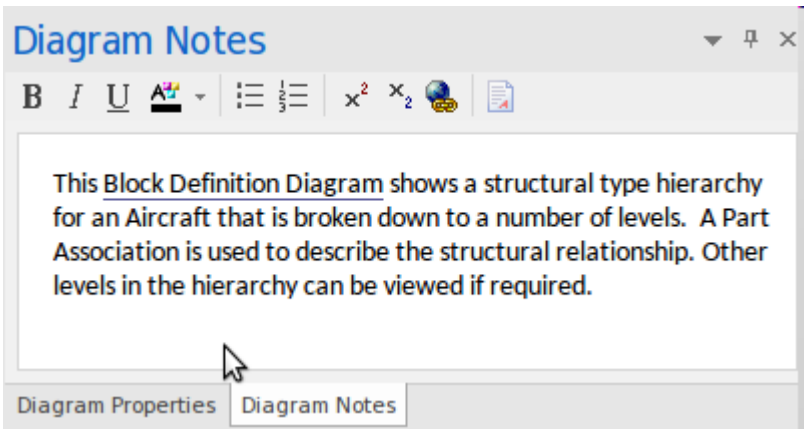
If set to non-selectable, the frame will auto-resize to fit the bounds of the diagram, expanding from its default size but not shrinking smaller. They can, however, be made selectable and adjusted to suit a modeler's preference.



Note that diagrams showing Diagram Frames applied using release 14.0 or later of Enterprise Architect will draw the parent object on the diagram when opened using a release of Enterprise Architect earlier than release 14.0.

Diagram Descriptions (Notes)

In addition to the meta information contained in the Header a diagram can have a description that is useful for newcomers to understand the purpose and intent of the diagram. The description can be added, viewed and maintained in the diagram's notes window,



A diagram is often created to describe aspects of a model or system. While the diagram itself and the elements and connectors it contains tell a story there is often the need to annotate the diagram with some extra information in the form of descriptive text. This text might for example:

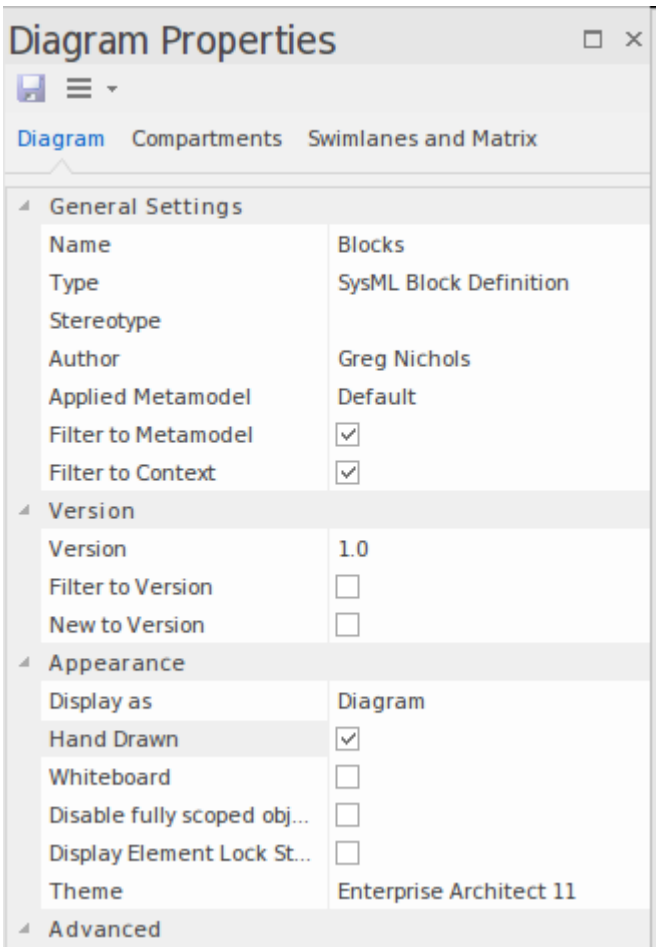
- Describe the purpose of the diagram,
- Highlight how to interpret the diagram,
- Contain link to other information in the model,
- Provide an explanation of the symbols used.

The notes will be generated to documentation and are visible through the WebEA interface.

Diagram Properties

Each diagram has a series of properties that describe the diagram at a meta level including such items as the:

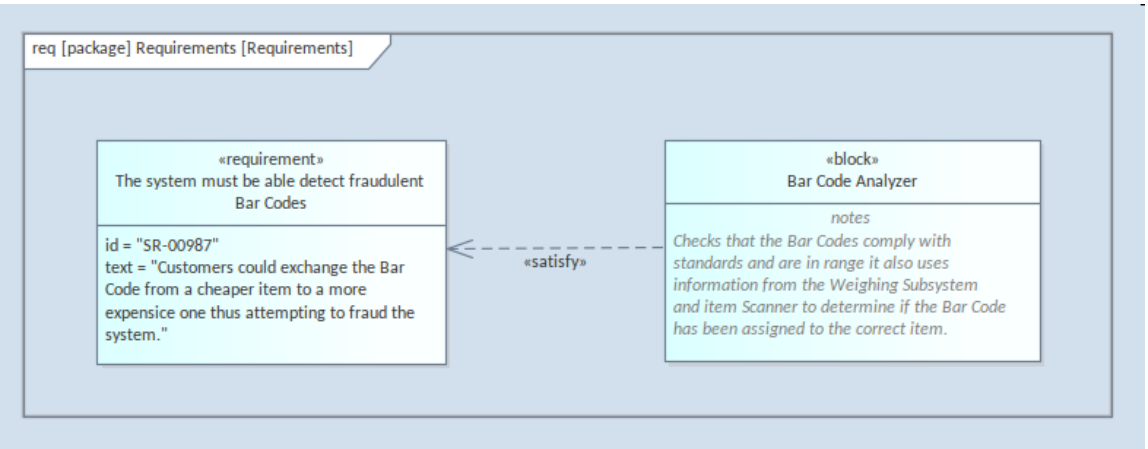
- Date the diagram was created,
- Date the diagram was modified,
- Model author who created the diagram,
- Hand Drawn and Whiteboard Mode
- Visual theme, and many more.



In addition there is a tab dedicated to specifying which compartments are visible in the diagram and another tab used to specify matrices and swimlanes.

Changing Themes and Appearance

Enterprise Architect provides a facility that allows you to apply a selected theme to all diagrams presented on your device. You can use this to create a particular style of the diagram and it can effect color, font, gradient, line thickness and background image (tile). It is a useful mechanism to give your diagram more appeal or to unify their appearance.

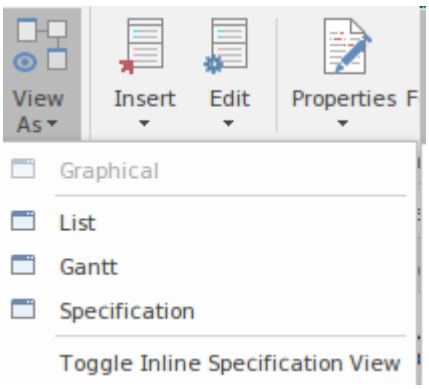


Colors and styles can also be set for each diagram element individually either as a default (every diagram the element appears in) or only for the element on the current diagram.

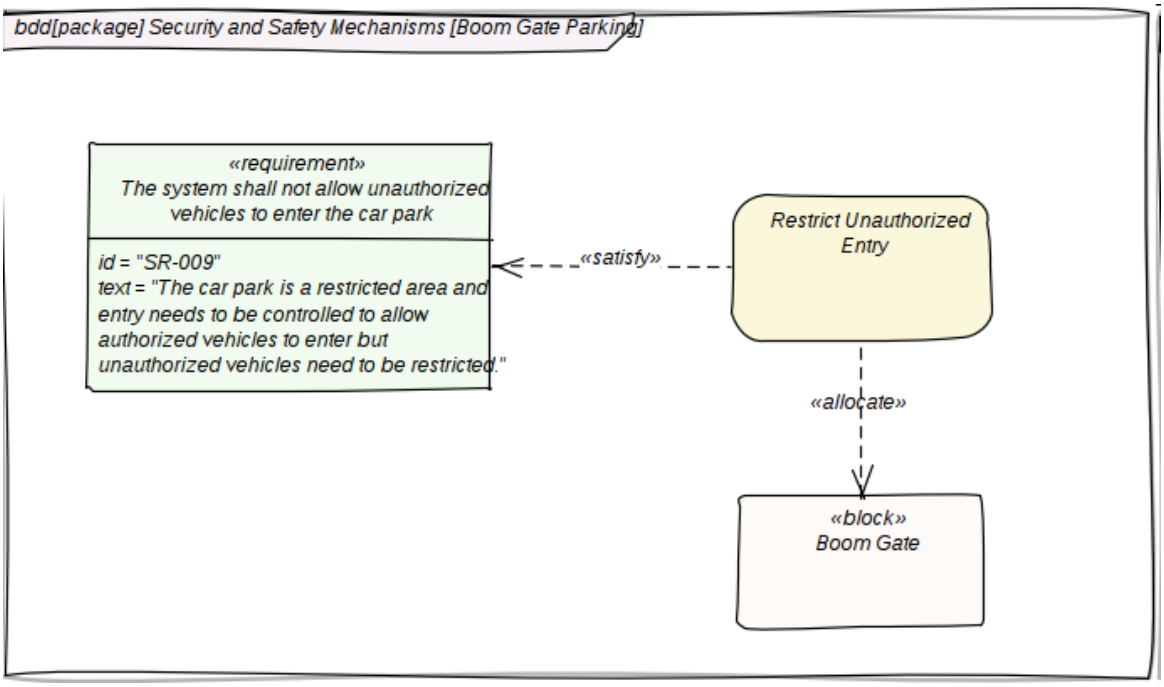
Alternative Visualizations of Diagrams

Whilst being compliant with the SysML specification and providing all of the diagram functionality expected in a modeling tool, Enterprise Architect provides a number of ways for modelers to view the diagram differently, bringing rich visualizations of the diagram and its elements. These include:

- *Specification View* - presents the elements in a familiar word processor or spreadsheet format, allowing elements and text to be updated
- *List View* - presents the elements in a list that can be sorted and grouped, and the fields updated
- *Inline Specification View* - presents the diagram alongside a narrative view similar to the Specification View
- *Gantt View* - presents the elements in a Gantt view showing resource allocation and other temporal information

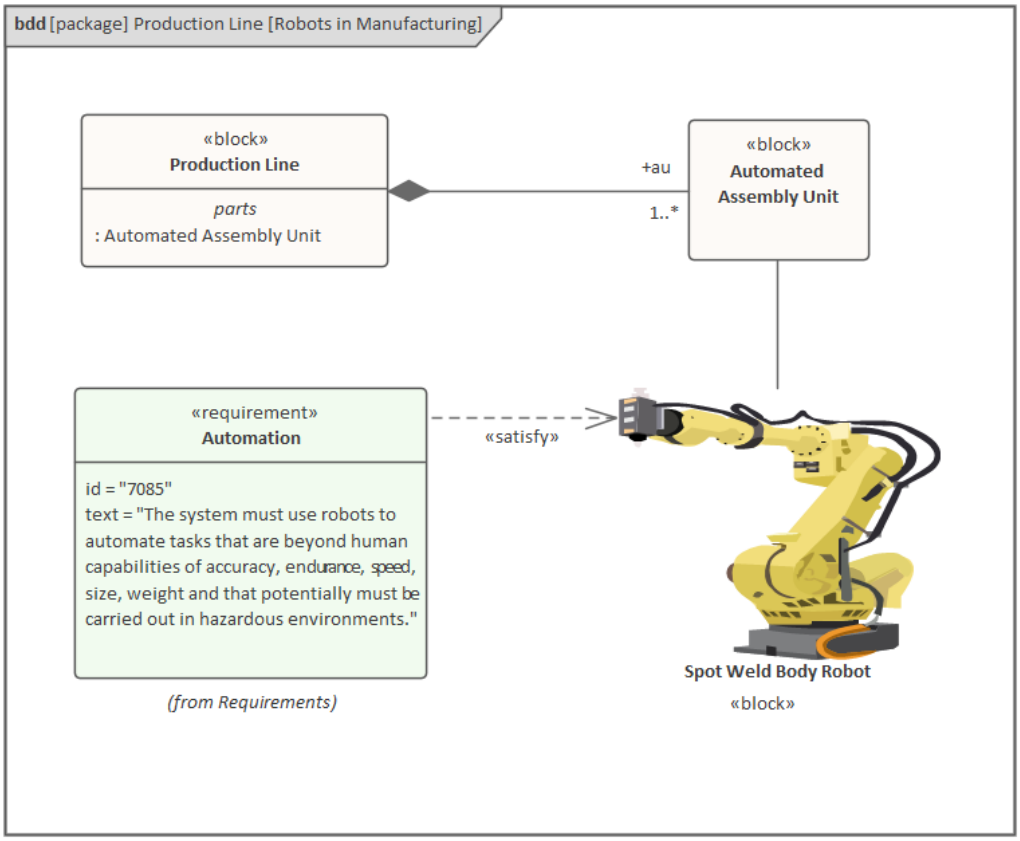


In addition, the diagram can be viewed in two modes that change the appearance of diagrams suitable for presentations, adding great appeal and attenuating the focus on the strictness of the underlying modeling language.



Creating Appeal with Alternative Images

The diagrams created as part of a model are intended to communicate ideas to an audience and the intent of the diagram is often better conveyed to a non technical or business audience by the use of an image. Enterprise Architect provides a mechanism to replace the vanilla (and sometimes) unappealing graphical notation of the Systems Modeling Language with an image in a variety of formats including vector based images.



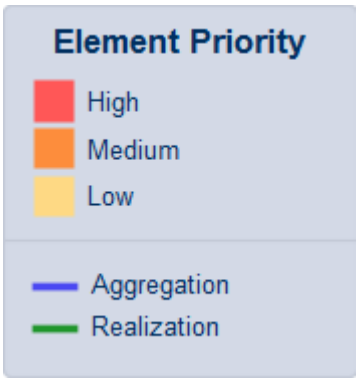
The image can be applied to every instance of the image in diagram or just for a particular diagram. A set of default images can be imported into the Image Manager or a user or team is free to create their own images specific to a particular domain or industry.

Diagram Filters

Diagram Filters provide a mechanism for filtering out parts of a diagram or list of elements that are not of interest, leaving just the elements and connectors that are relevant to the view. The filters can be defined for elements or connectors and there is a wide range of criteria that can be set, such as filter out all elements that don't have a status of 'Validated' and were created since a milestone date. More simply an element type or stereotype can be filtered out. The elements that do not meet the criteria can be hidden, gray-scaled or simply dimmed (faded) so they are visible but not prominent.

Diagram Legends

Diagram Legends provide a way of describing the elements and connectors used in the diagram. The legends can dynamically change the visual aspects of elements and connectors in the diagram, for example by changing fill color, line color and line width based on element properties or Tagged Values.



Any number of legends can be created and they can be applied to one or more diagrams.

Common Aspects of Diagrams

Diagrams are one of the most important ways to visualize the contents of a model and represent a diagram author's expression of what they consider important. Its compelling visual appeal and its ability to act as a narrative telling a story about some aspect of the system being modeled makes the diagram one of the most important views. Each diagram in the SysML has common aspects (or features) including:

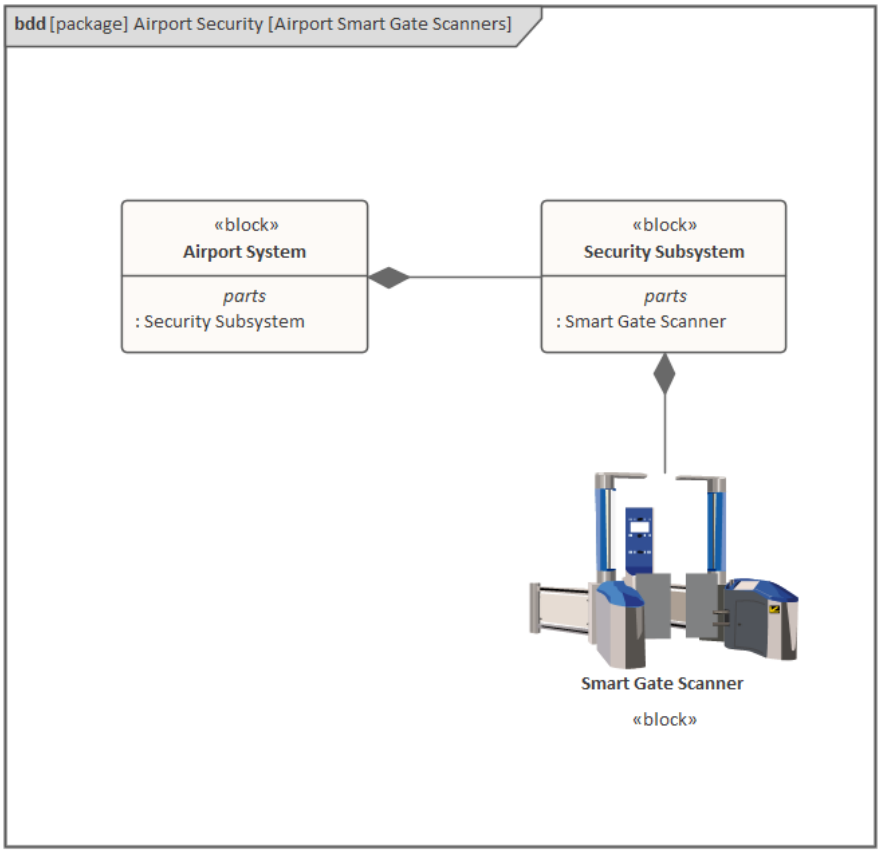
- a diagram Frame
- a diagram Header
- a diagram Contents Area (Canvas)
- a diagram Description (Notes)
- a diagram Properties Sheet
- A diagram Legend
- And more

There is also a wide range of other facilities and that will help the engineer when working with diagrams these include:

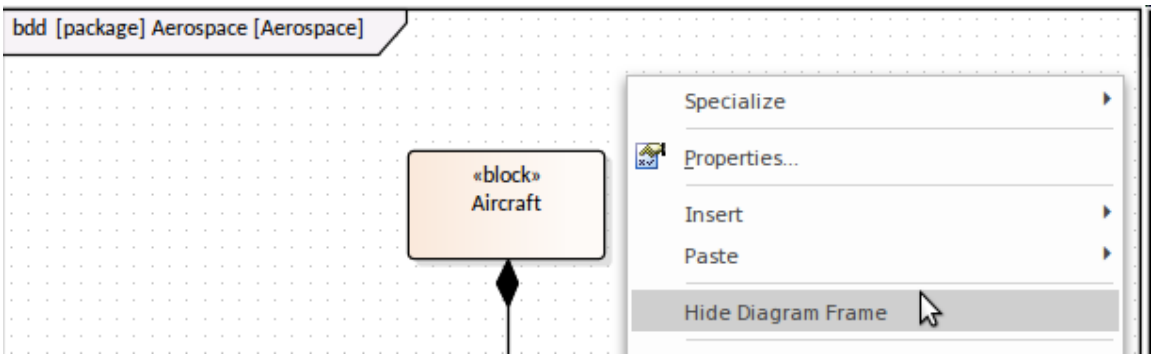
- Traceability Window
- Relationships Window
- Pan and Zoom
- Diagram Layout
- Diagram Filters
- Roadmaps
- Kanban
- Zoom options
- Appearance, Alignment and Style tools

Diagram Frame

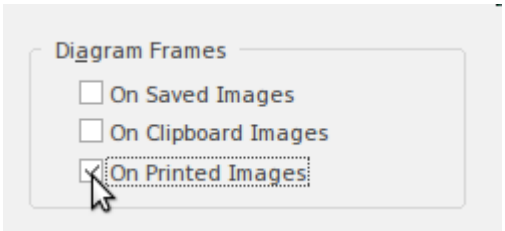
The Diagram Frame resembles a curtain enclosing the elements that form part of the diagram. While it is more important when viewing diagrams in line with written text, some modelers prefer to have the diagram visible when modeling.



The frame can be conveniently shown or hidden for each diagram, and when it is hidden the diagram information - such as the type, parent and the name of the diagram - is still visible in the diagram header.



Preferences can also be set to show the Diagram Frame when diagrams are exported as part of documentation, to the clipboard or in saved images. These options are located on the 'Diagram' page of the 'Preferences' dialog.



The frame contains a header in the top left hand corner, which contains useful information on the diagram. This syntax describes the contents of the header.

diagramKind [modelElementType] modelElementName [diagramName]

1. diagramKind - is a code for the type of diagram, such as bdd (Block Definition diagram),
2. *modelElementType* - is the type of element that is acting as the namespace of the diagram,
3. *modelElementName* - is the name of the namespace element,
4. *diagramName* - is the name of the diagram, provided by the user.

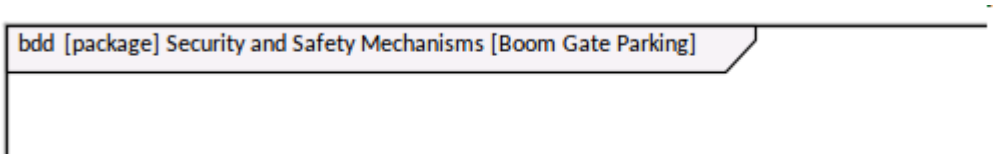
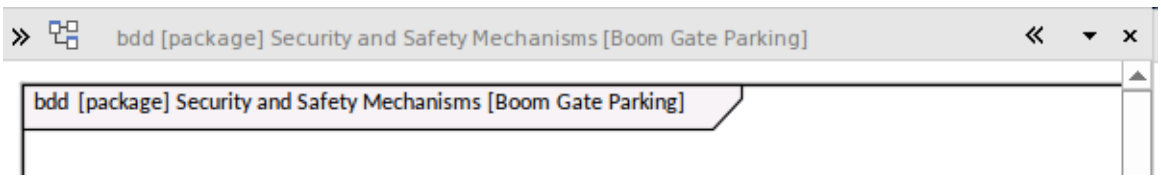


Diagram Header Bar

The diagram header bar provides useful information about the diagram and tools to work with all open diagrams. Even when the frame is not visible, the header will show the diagrams details.

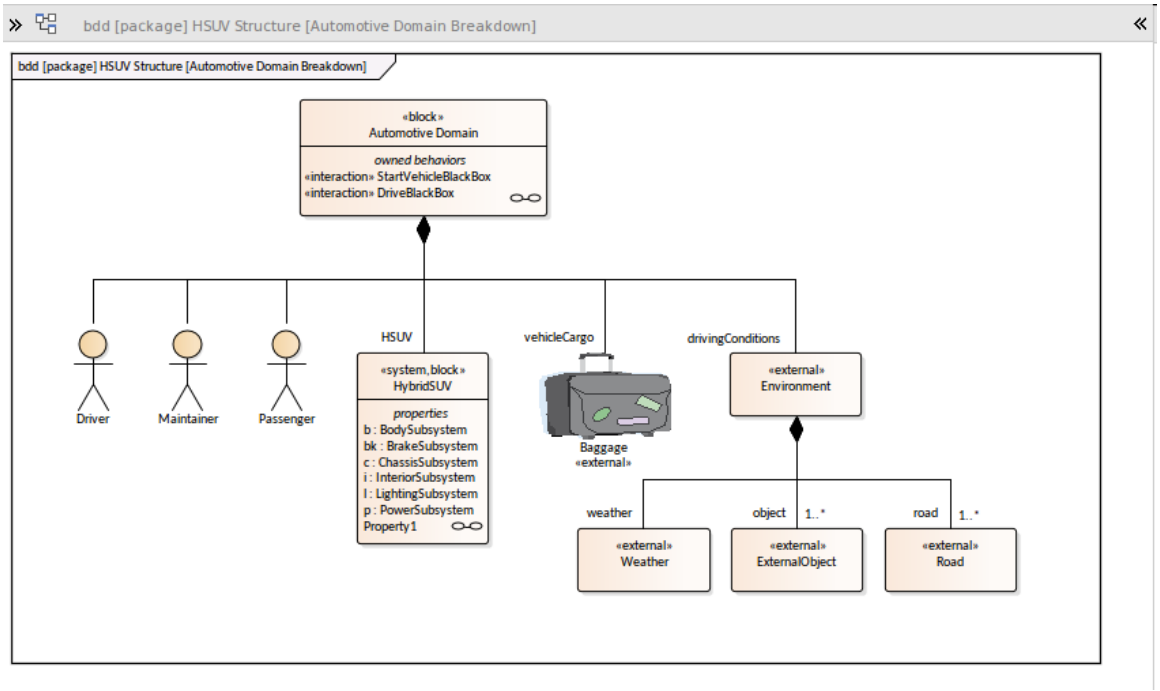


It also has a number of other useful icons that are used to control aspects of the display, including:

- *Toolbox Chevron* - that hides or shows the toolbox for all displayed diagrams,
- *Document Chevron* - that shows or hides the Inline Specification view of the diagram,
- *Open Diagrams Arrow* - which displays a list of open diagrams, indicating the one with unsaved changes,
- *Close Diagram Icon* - that allows the diagram to be closed.

Diagram Contents

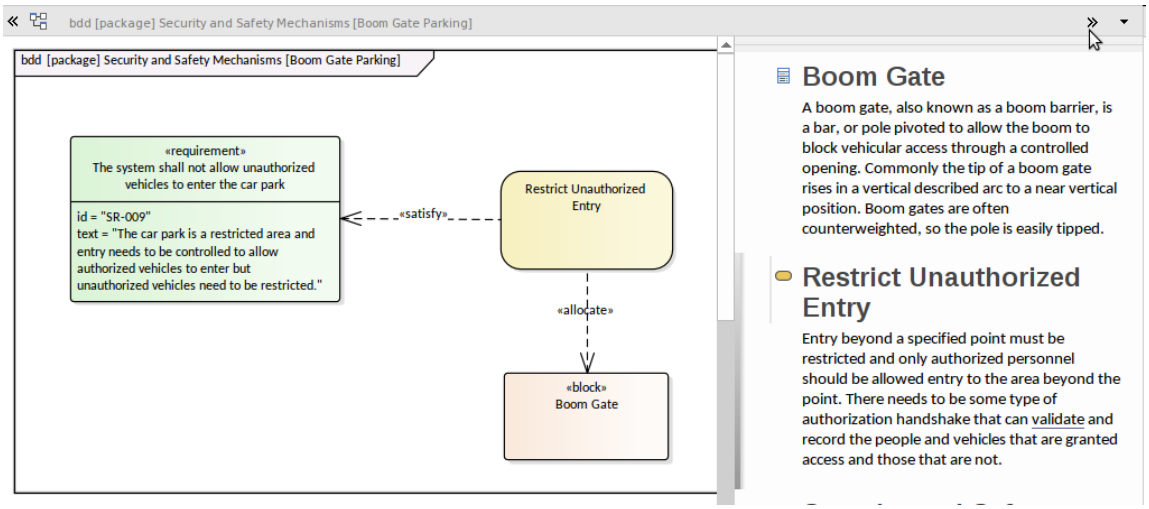
The diagram content is the canvas where you view and work on your diagram. It is a highly flexible work area with lots of useful tools for both creating and visualizing existing diagrams. Pictures can be added and mixed with the standard geometric SysML elements, allowing expressive and compelling diagrams that help convey important engineering concepts to both technical and non-technical audiences.



Enterprise Architect extends the way to view a diagram using a number of visualization techniques. These will provide you with alternative ways to work with diagram content and are welcomed by newcomers who might be more familiar with working with elements in spreadsheets, list and documents. Notice also in this diagram that images can be used as an alternative to the vanilla SysML shapes.

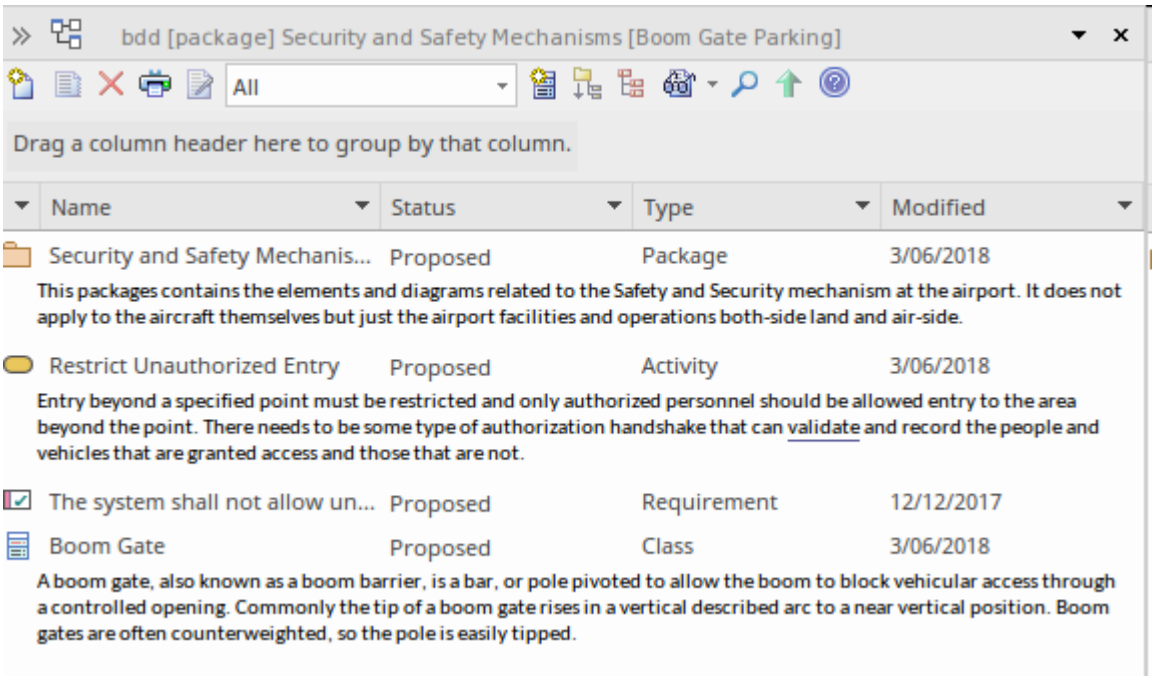
Document View

This is a convenient view that displays the elements on the diagram in a narrative form in a document. It is also known as the 'Inline Specification' panel and is not a separate view but a panel related to the selected diagram. The document is conveniently opened to the right of the diagram so both diagram and document can be viewed simultaneously. Each element present in the diagram will have a heading in the document with the element's notes (description) displayed in position under the heading. Elements and their notes presented in this view can be conveniently edited, all the while viewing both the document view and the diagram, and the two are kept synchronized. The elements are by default listed in alphabetical order but this sort order can be changed from the context menu to follow top to bottom or left to right.



List View

This is a useful way of viewing the elements in a diagram while allowing them to be viewed, updated and created in a familiar spreadsheet-like view. Element properties including standard properties, Tagged Values and notes can be managed, and drop down lists for properties with a discrete list of values are available, providing a welcomed way of viewing these properties across multiple elements.



Specification View

The Specification Manager resembles the inline document viewer, but gives more power and opens in a separate dockable window. It is the perfect tool, designed for engineers and other stakeholders who are more familiar with working with spreadsheets or documents. It essentially allows a modeler or viewer to visualize the contents of a diagram (or Package) as a document or spreadsheet. The document view resembles the familiar word processor document, which can be edited in-line to create new elements and their descriptions. The visualization can be changed to resemble a spreadsheet where properties are displayed in columns.

Item

1 REQ019 - Manage Inventory

The system MUST include a complete inventory management facility to store and track stock of books for the on-line bookstore.

1.1 REQ122 - Inventory Reports

Inventory reports are required that detail the available stock for each item including back orders. Future stock level reports should be able to predict the quantity of stock at a specified future date.

1.2 REQ023 - Store and Manage Books

A book storage and management facility will be required.

1.2.1 REQ022 - Order Books

A book order facility will be required to allow on-line ordering from major stockist's.

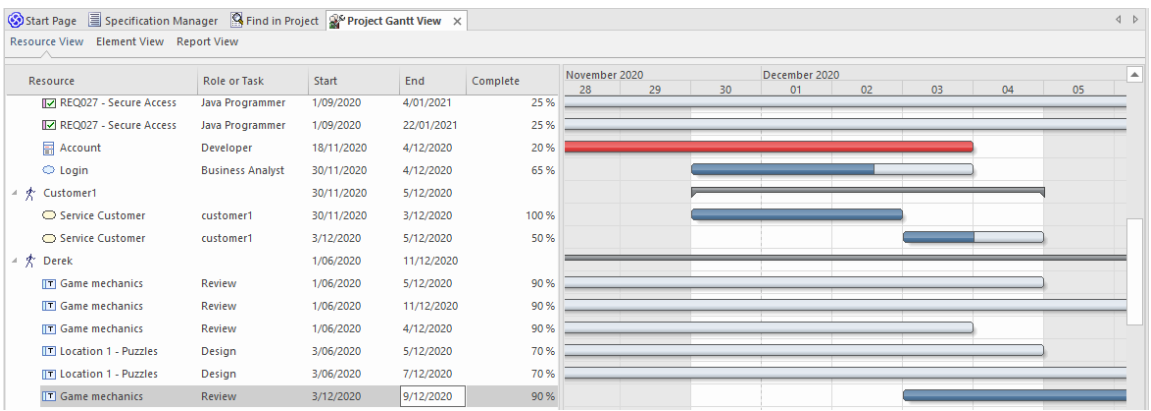
1.2.2 REQ021 - List Stock Levels

A facility will exist to list current stock levels and to manually update stock quantities if physical checking reveals inconsistencies.

It is a highly welcomed view for those transitioning from document-based systems engineering processes, and a favorite for engineers entering test-based information such as a set of requirements, activities, components and other elements.

Gantt Chart

This provides a typical Gantt chart view of the elements in the diagram where resource allocation has been applied to the elements this can be visualized to give an engineering or project manager a view on progress within the model.



Diagramming Tools

There is also a wide range of other tools that can be used to display the information in the repository that will assist the engineer when working with elements in diagrams, including but not limited to:

Traceability Window

Used to view how element in the model are connected in a graph of elements and their relationships. For more information see the The Traceability Window topic.

Relationships Window

Used to view the relationship between a selected element and other elements in the model; these relationship are not visible but can be conveniently visualized in this separate window. For more information see the The Relationships Window topic.

Pan and Zoom

Used to move around a large diagram by using a small rectangle that represents the view-port, and to zoom in using a slider control. For more information see the Pan and Zoom topic.

Diagram Layout

Used to create attractive layouts of a diagram, using selected visual layout patterns such as digraphs and springs. For more information see the Layout Diagrams topic.

Diagram Filters

Used to filter elements from view in a diagram either by making hiding them, changing them to a gray scale, or fading them. You can also reverse the behavior and select element to include. For more information see the Diagram Filters topic.

Roadmaps

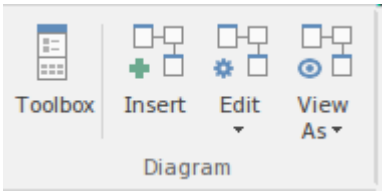
Used to create time based representations of elements, with an applied timescale that can be configured to suit the project or modeler to show a roadmap for strategic and development purposes. For more information see the Roadmap Diagram topic.

Kanban

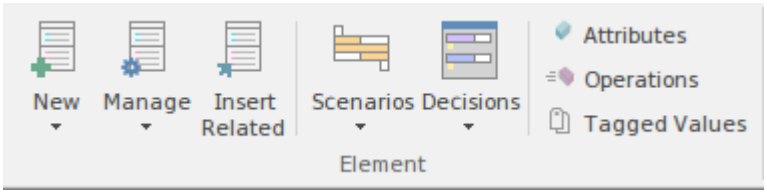
Used to manage items in a traditional Kanban diagram, where elements are moved between columns that represent their order in a staged process; resources working on the items can be visualized, providing a useful way for a team to manage its model or product development. For more information see the Kanban Boards topic.

Diagram Ribbons and Menus

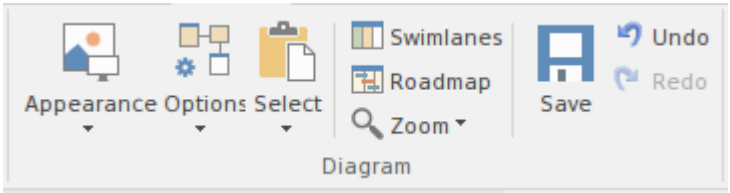
There are a number of ribbons that are useful when working with diagrams. The starting point, as described earlier, is the 'Diagram' panel of the Design ribbon, which allows you to insert new diagrams and edit and change the view of existing diagrams.



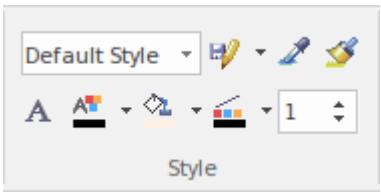
The 'Element' panel on the Design ribbon will also be useful when working with elements on the diagram (or selected in the Browser window) allowing you to insert new elements, edit existing ones and manage element properties, features and responsibilities. For more information see the Design Ribbon Help topic.

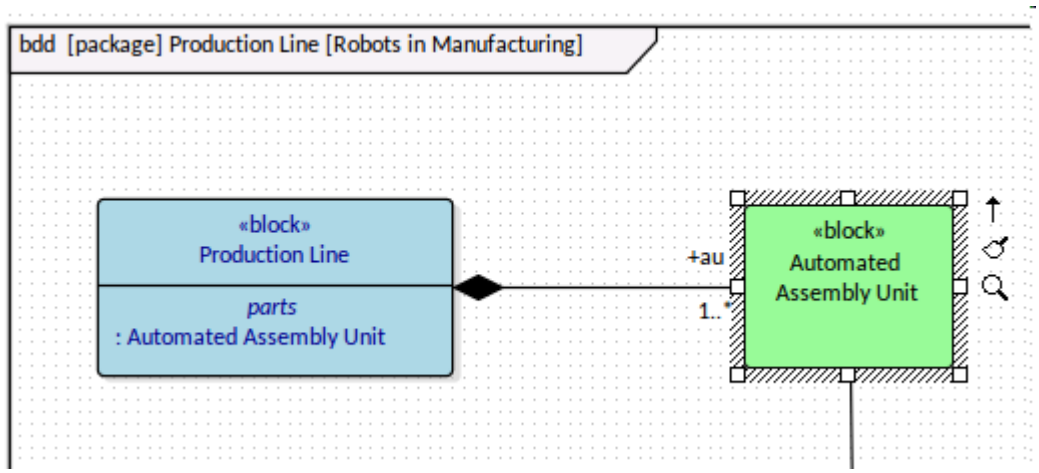


Another important ribbon is the Layout ribbon, which contains a number of panels that will be useful for working with diagrams. This includes the 'Diagram' panel, which contains options to set themes and change the diagram mode, for example to Hand-Drawn. For more information see the Layout Ribbon topic.



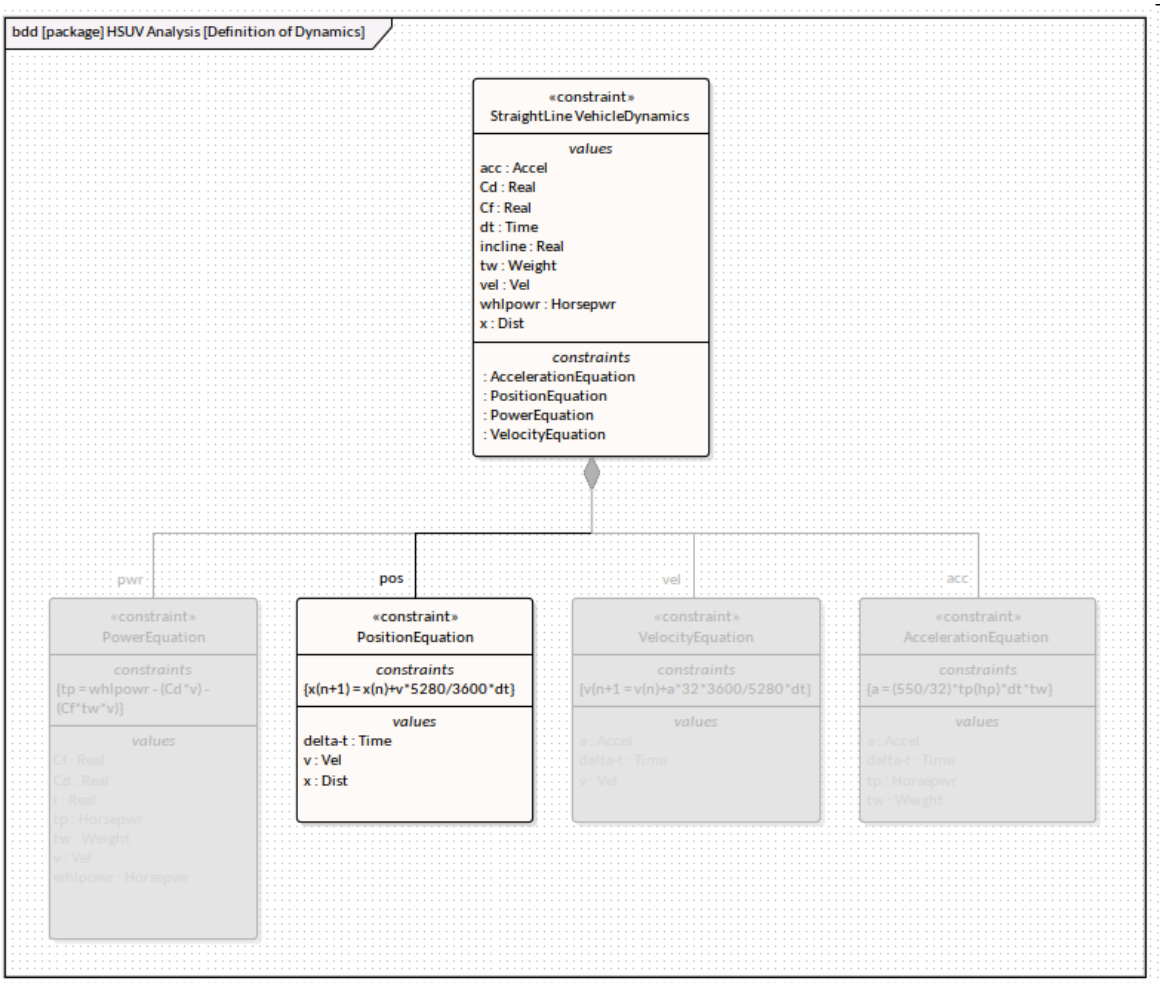
The 'Style' panel allows the visual style properties of diagram elements to be set either as a group or for individual elements.





The 'Alignments' Panel provides a rich set of tools for aligning elements in a diagram. These are very useful and provide a range of alignment options that are not typically available in most drawing Packages, allowing elements - for example - to be spaced evenly horizontally or vertically.

The 'Tools' Panel provides a series of tools for working with diagrams, allowing for filtering content in diagrams - for example, allowing a modeler to display only Constraint elements with a specified status, that were created after a specified date - Pan and Zoom and Layout options. The Filter panel provides a quick and ad-hoc way to filter content in the diagram without the need to create a diagram filter.



A series of helpers are also available for working with diagrams and their elements, controlling such things as horizontal and vertical hold to restrict element diagonal movement.

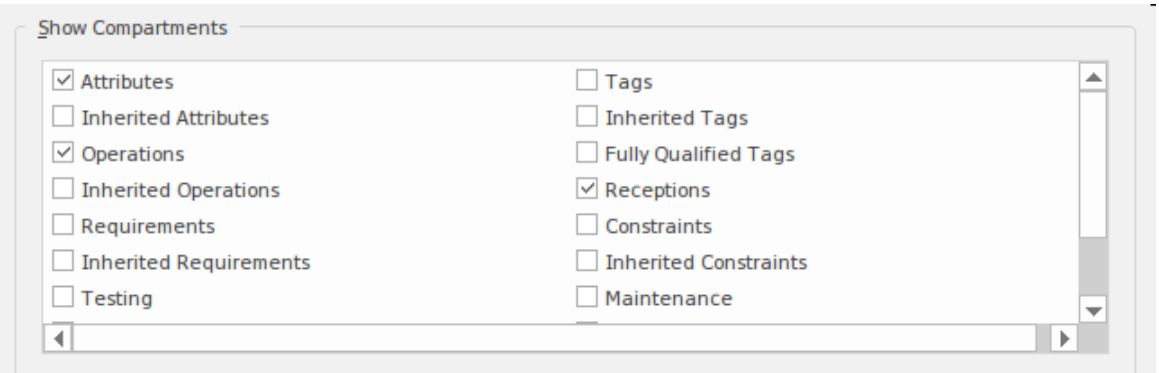
Diagram Properties

A diagram in Enterprise Architect has a rich set of properties, some being descriptive - such as the name - and others being prescriptive, specifying how the diagram should be displayed and what elements, compartments and other features should be visible, including line styles.

Many of these properties can be set both at a diagram level and at an element or connector level, allowing individual parts of the diagram to be displayed differently to others.

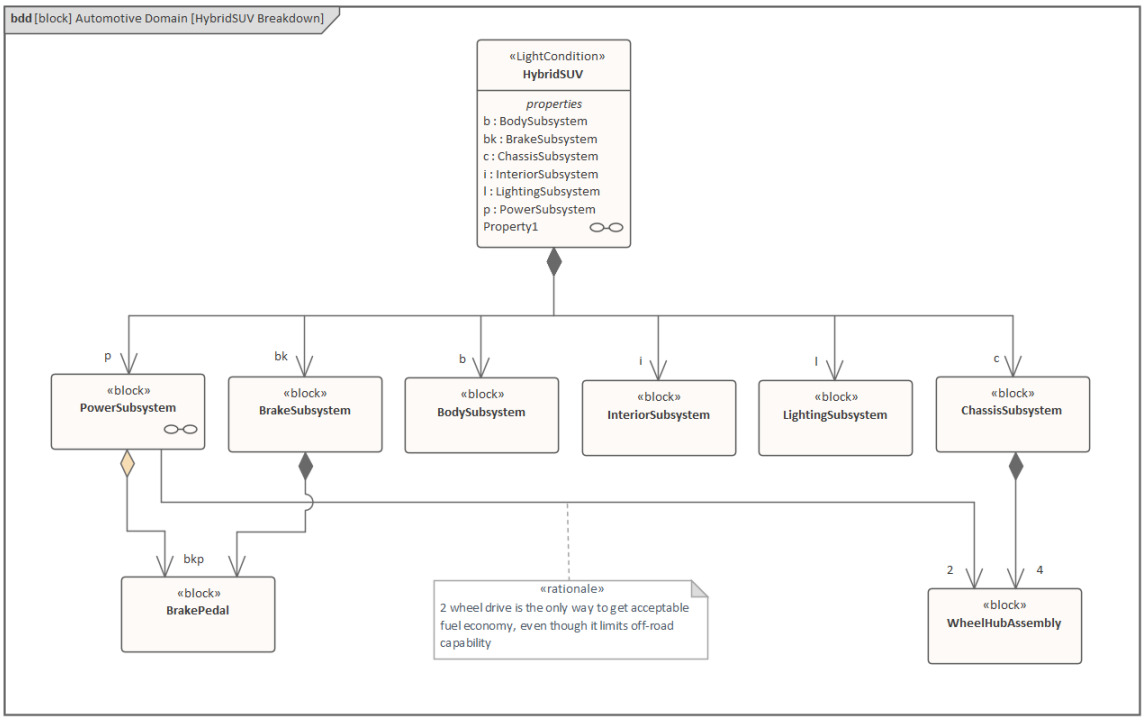
The ability to set the visibility of element compartments is particularly useful for engineering diagrams, as the SysML provides a rich set of compartments for a wide range of items. These compartments, if not managed, can clutter a diagram and attenuate a reader's ability to understand the meaning of the diagram.

The compartments, as with other properties, can be set at a diagram level or an element level, which allows a modeler to select the compartments to display for specific elements.


Show Compartments
Attributes
Inherited Attributes
Operations
Inherited Operations
Requirements
Inherited Requirements
Testing
Tags
Inherited Tags
Fully Qualified Tags
Receptions
Constraints
Inherited Constraints
Maintenance

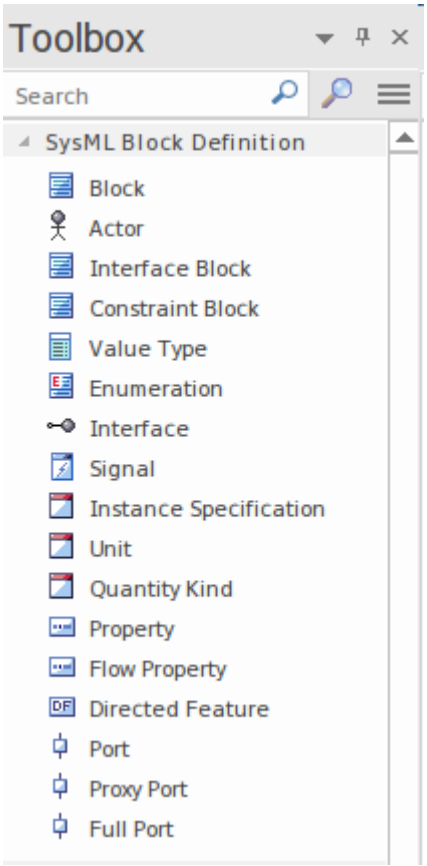
Block Definition Diagram

The Block Definition diagram is the most widely-used of the SysML diagrams; it is used to model Blocks, their relationships to other elements (including other Blocks) and their features in the form of Properties, Operations and Receptions. Blocks are modular units of system description and provide a way of modeling systems as a graph or tree of modular units. Other elements, such as ConstraintBlocks and Properties, can also appear on the diagram and help describe the system being modeled. For more information see the Block Definition Diagrams (BDDs) Help topic.

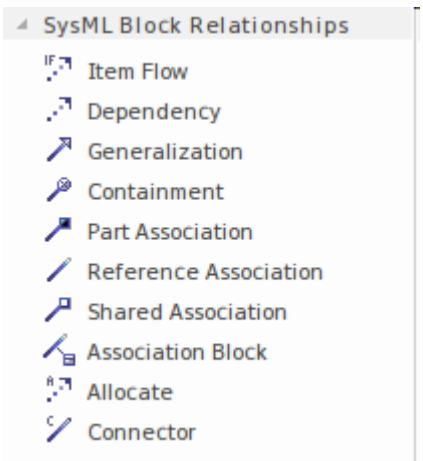


Elements

The main elements that can appear in Block Definition diagrams are provided by the Diagram Toolbox for this diagram type:



The main connectors that can appear in Block Definition diagrams are as shown:



Tools

A variety of tools can be used with structural modeling and Block Definition diagrams, including:

- Diagram Filters - which allows a user to filter elements out of the diagram to achieve a more specific focus
- Pan and Zoom - which allows a modeler or viewer to easily move around large diagrams
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model

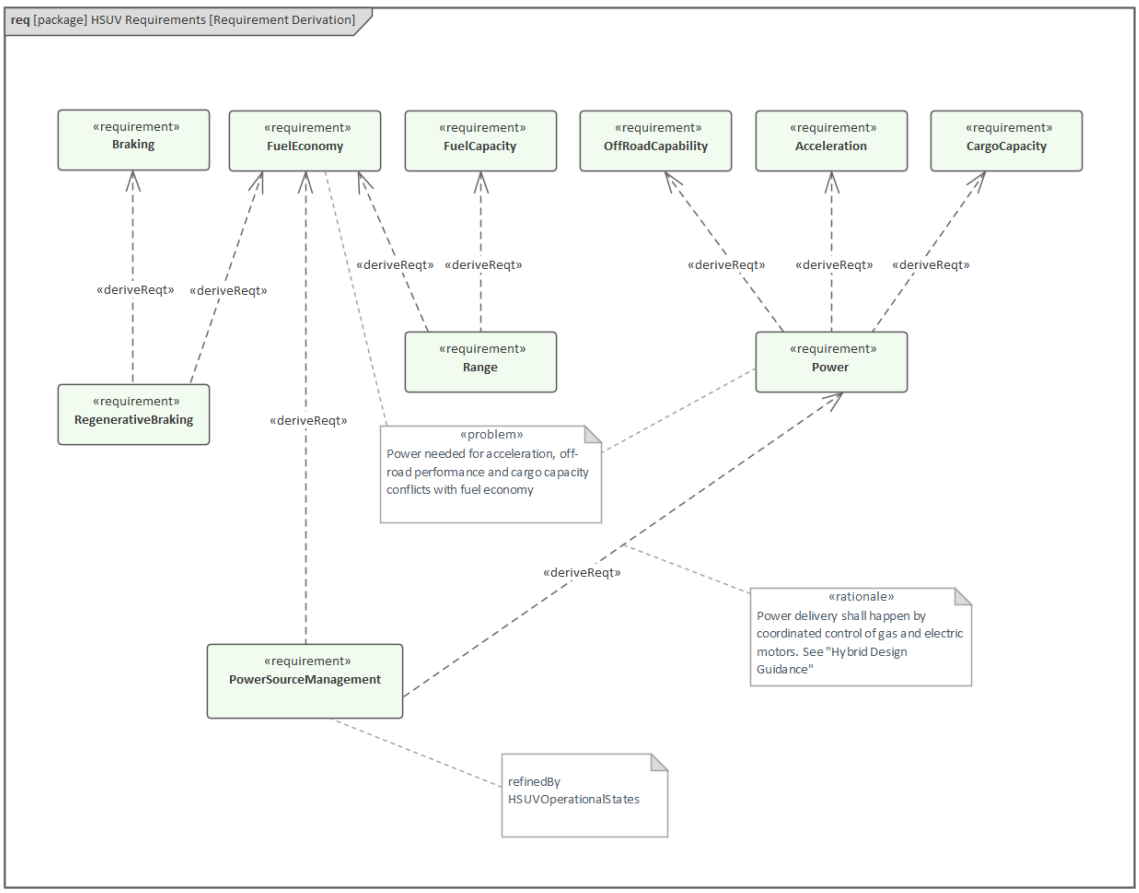
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats
- Traceability - which provides a hierarchical view of an element's relationships to other model elements
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios
- Relationship Matrix - which allows the connections between Blocks (or other elements) and other elements such as Requirements and Use Cases to be visualized in a matrix

Usage

The Block Definition Diagram is a general-purpose diagram that can be used to describe the structural aspects of a system.

Requirement Diagram

The Requirements diagram is used to create and view Requirements and their relationships to other elements, including other Requirements. Requirements can be specified at any level, from strategic enterprise or business requirements through stakeholder requirements down to low-level engineering and even software and transition requirements. For more information see the SysML Requirements Modeling Help topic.



The elements contained in this diagram can be viewed in a number of different ways, including:

- *Specification View* - allowing the elements and their notes to be displayed in word processor or spreadsheet format in a separate dockable window
- *Inline Specification View* - allowing the diagram and a list of its elements in a narrative form to be viewed side-by-side
- *List View* - allowing the diagram elements to be viewed in a list that can be sorted and the elements grouped by properties
- *Gantt View* - allowing the diagram elements to be represented on a Gantt chart showing how resources are utilized over time

Elements

The main elements that can appear in Requirements diagrams are:

- Requirement
- Test Case

The main connectors that can appear in Requirements diagrams are:

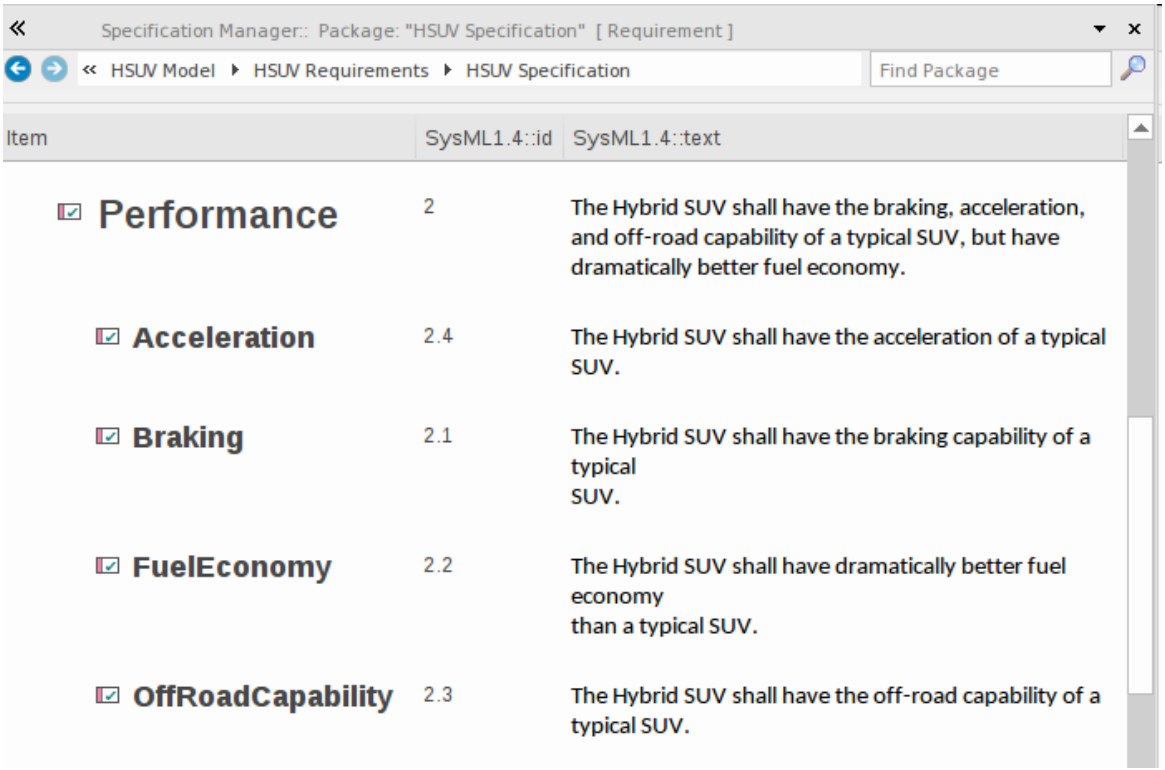
- Containment
- Trace
- Copy
- Derive
- Verify
- Refine
- Satisfy

Tools

A variety of tools can be used with requirements modeling, including:

- Specification Manager - which allows a user to work with requirements in a spreadsheet or word processor type of format
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats
- Traceability - which provides a hierarchical view of an element's relationships to other model elements
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios
- Relationship Matrix - which allows the connections between Requirement (or other elements) and other elements such as stakeholder needs to be visualized in a matrix
- Mind Mapping - which provides a way of recording the progress of a meeting, thus recording - for example - the stakeholders' needs

Probably the most widely used of these requirement tools would be the Specification Manager, which will provide a welcome and familiar way of working with textual specifications such as requirements or constraints. The Specification Manager can be used to view a list of elements contained within a Package or a diagram.

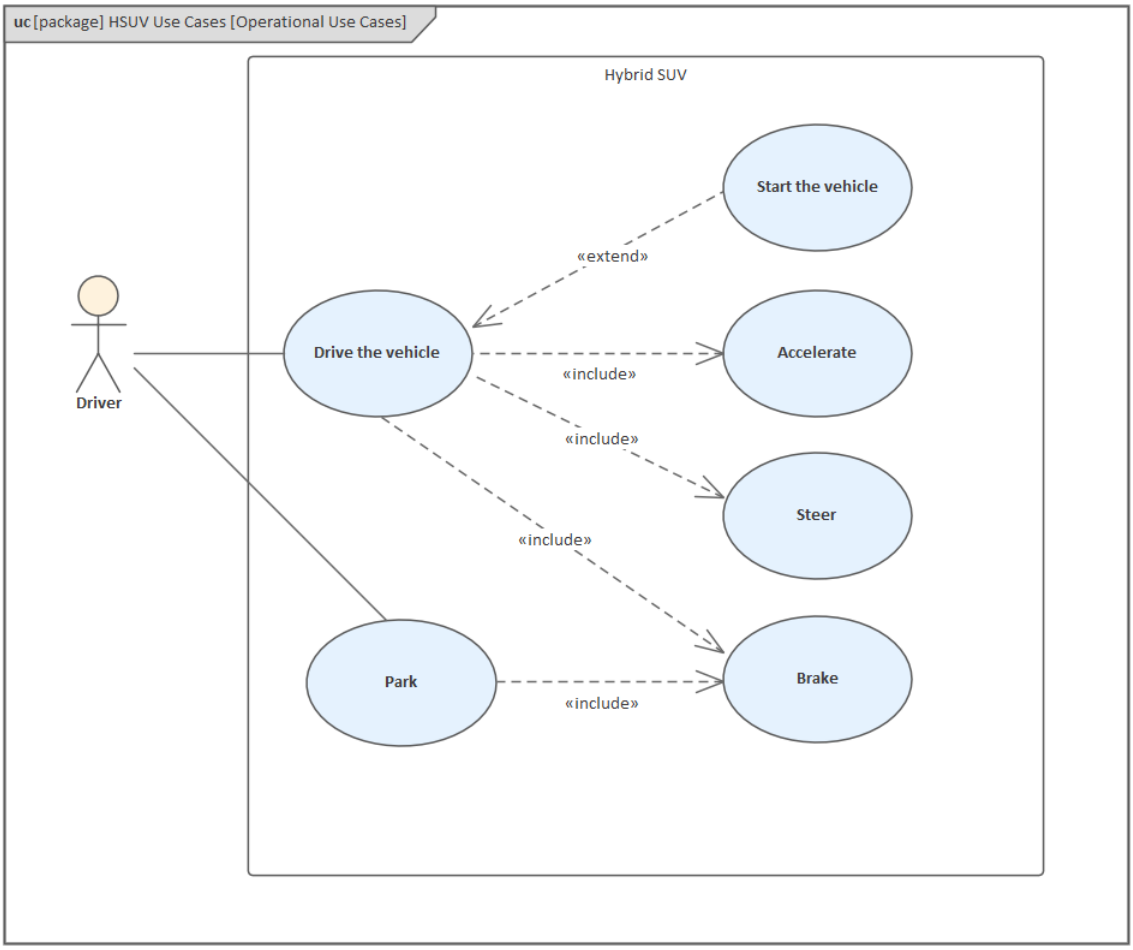


Usage

The Requirements diagram can be used to show a hierarchy of requirements using the containment relationship allowing a viewer to see how the structural relationships of the requirements. It is however most compelling when Requirements are viewed in a diagram with other elements using other relationships including other requirements. An example of this is the relationship between Requirements and Test Cases or Requirements and the Components of a solution.

Use Case Diagram

The Use Case diagram is used to define and view Use Cases and the Actors that derive value from the system. The Use Case diagram describes the relationship between the Actors and the Use Cases, enclosing the Use Case within a Boundary that defines the border of the system; the Actors, by definition, lie outside the Boundary. While the Use Case diagram can appear simplistic, it is a useful communication device that describes the value or goals that external roles obtain from interacting with the system. Each Use Case can be detailed with descriptions, constraints and any number of scenarios that contain sets of steps performed alternately by Actor and system to achieve the desired goal.



Elements

The main elements that can appear in Use Case diagrams are:

- Boundary
- Actor
- Use Case
- Scenario

The main connectors that can appear in Use Case diagrams are:

- Communication Path
- Generalization

- Includes
- Extends

Tools

There are a variety of tools available for working with Use Cases in addition to the Use Case diagram itself. These include:

- *Scenario window* - which provides a way of detailing the descriptions, constraints and the step of each scenario
- *Documentation Generator* - which allows corporate, reports or ad-hoc documentation to be created in a variety of formats, including docx, pdf and rtf
- *Traceability* - which provides a hierarchical view of an elements relationships to other model elements
- *Responsibility window* - which provides a composite view of the important responsibilities of an element including Constraints, Requirements and Scenarios
- *Relationship Matrix* - which allows the connections between Requirement (or other elements) and other elements such as stakeholder needs to be visualized in a matrix

The main tool used for working with Use Cases is the Scenario window, which is a comprehensive and purpose built facility for working with Use Cases and Scenarios. The tool can be used to define the details of a Use Case and its scenarios and constraints, which provides a productive alternative to the traditional text-document based approach to defining Use Cases. This ensures that the Use Case diagram and the textual details of the Use Cases and its Scenarios and Constraints are all contained in the same model and can be traced.

If the Use Cases are required in a document format for contractual or process reasons, a Use Case Report can be generated automatically from the models using the in-built documentation engine.

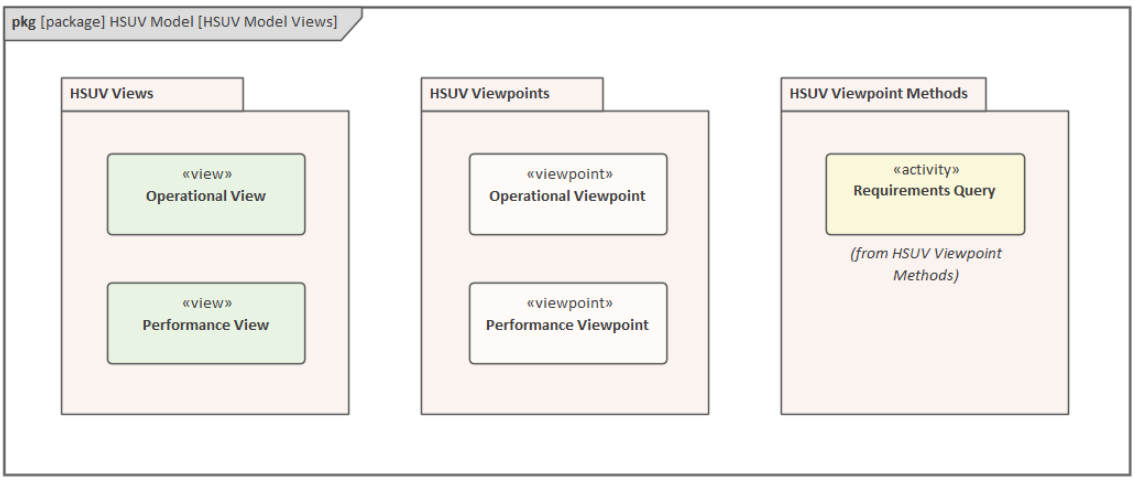
Scenario Builder

Usage

The Use Case diagram can be used to define the details of a Use Case and its Scenarios and Constraints. This is a welcomed alternative to the traditional text-document based approach commonly used to define Use Cases. This ensures that the Use Case diagram and the textual details of the Use Cases and its Scenarios and Constraints are all contained in the same model and can be traced. If the Use Cases are required to be presented in a document format for contractual or process reasons, a Use Case Report can be generated automatically from the models using the in-built documentation engine.

Package Diagram

The SysML Package diagram is used to define or view the Packages that provide the fundamental organization of the repository. These can include name-spaces and their sub-Packages and other less formally defined groups of elements. The Packages that appear in diagrams can also be viewed in the Browser window and their hierarchy can be navigated by expanding and collapsing the tree.



The main element that is represented in the Package diagram is the Package itself, with the elements it contains. There are a number of important relationships between Packages, including Dependencies that show that one Package is dependent on one more other Packages. Packages can be organized into a number of different types of hierarchy.

Elements

The main elements that can appear in Package diagrams are:

- Model
- Model Library
- Package
- View
- View Point
- Stakeholder

The main connectors that can appear in Package diagrams are:

- Conform
- Dependency
- Import
- Containment
- Realization
- Refine
- Expose

Tools

A variety of tools can be used with structural modeling and Block Definition diagrams, including:

- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats
- Traceability - which provides a hierarchical view of an element's relationships to other model elements
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios
- Relationship Matrix - which allows the connections between Packages(or other elements) and other elements such as Requirements and Use Cases to be visualized in a matrix

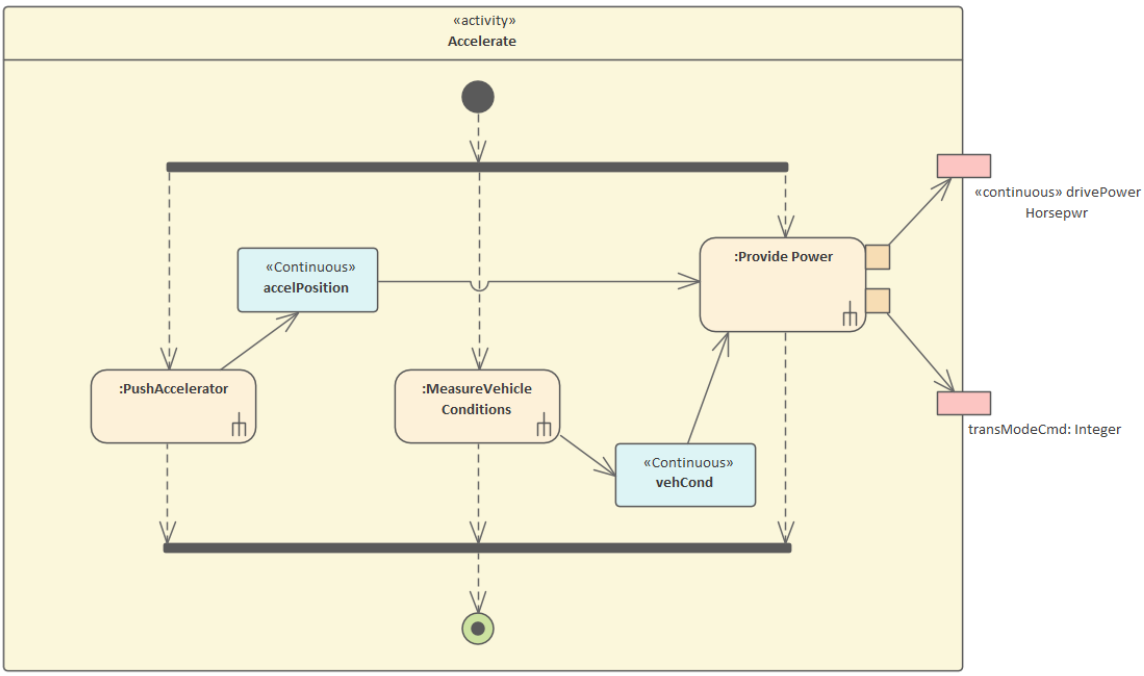
Usage

The Package diagram can be used to describe the relationship between Packages and the elements that they contain. While structural information is visible in the Browser window there is a range of relationships that can exist between Packages themselves and between Packages and elements, that cannot be visualized in the Browser window. Package diagrams can also be included in documentation and can assist in orientating an audience by giving them an overview of a section of the architecture or design in a similar way to providing a table of contents in a publication.

Activity Diagrams

The Activity diagram is the most important behavior diagram and can be used to model flow (discrete or continuous) based behavior where inputs are converted to outputs by traversing a sequence of actions that perform work on the items. They are analogous to the common flow chart diagram but have more sophisticated semantics and also allow Activities and Actions to be related to elements such as Blocks, Requirements and Use Cases.

The Actions that appear on the Activity diagrams can contain input or output pins that represent the interaction points where inputs are fed into an action and outputs are emitted.



Usage

The Activity diagram can be used to model flow based behavior and is similar to the widely-available Flow Chart or Functional Flow diagrams that had been used extensively before the SysML specification was devised. They are typically used to show the way parts of the system behave, including the input and output of items and signals.

Elements

The main elements that can appear in Activity diagrams are:

- Activity
- Action (Various kinds)
- Action Pin
- Partition
- Object Node
- Central Buffer Node

- DataStore
- Decision
- Merge
- Initial
- Final

The main connectors that can appear in Activity diagrams are:

- Control Flow
- Object Flow
- Interrupt Flow
- Dependency

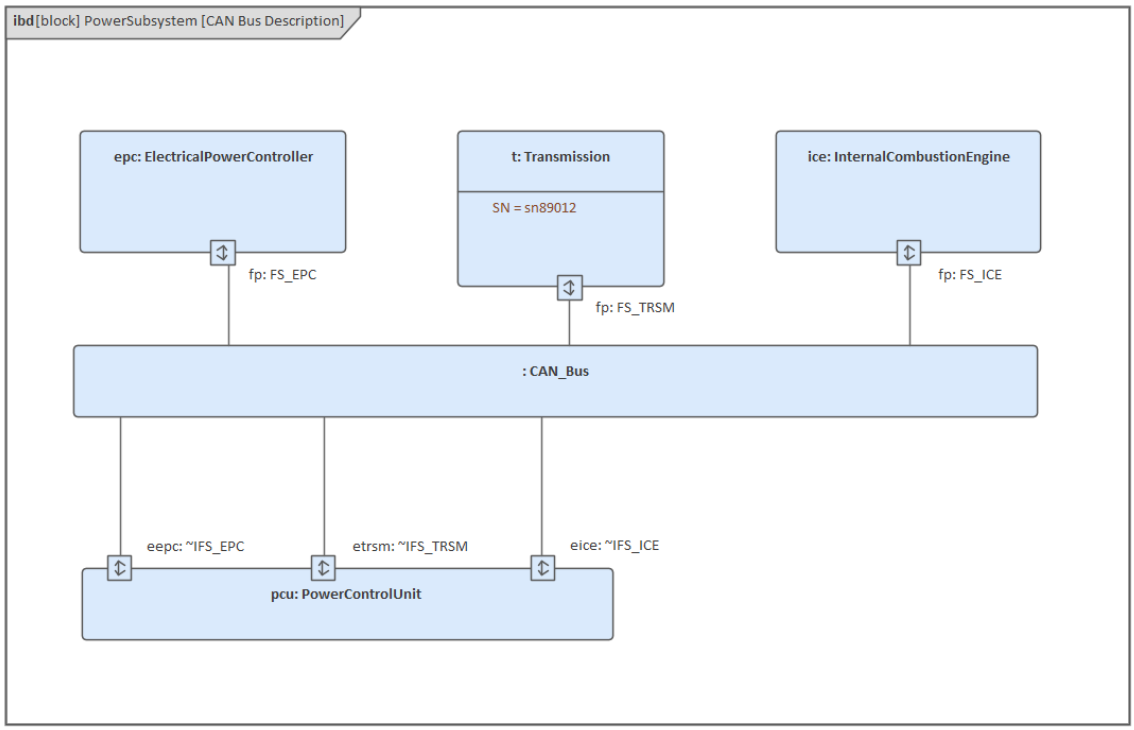
Tools

A variety of tools can be used with behavioral modeling and Activity diagrams, including:

- Diagram Filters - which allows a user to filter elements out of the diagram to achieve a more specific focus
- Pan and Zoom - which allows a modeler or viewer to easily move around large diagrams
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats
- Traceability - which provides a hierarchical view of an element's relationships to other model elements
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios
- Relationship Matrix - which allows the connections between Activities (or other elements) and other elements such as Blocks, Requirements and Use Cases to be visualized in a matrix

Internal Block Diagram

The Internal Block diagram provides a way of viewing the composition of a block using part properties connected together using ports and connectors. The diagram is useful for showing a Block's (represented by the diagram frame) composition and the flow of inputs and outputs between the various parts that make up the block, when required the direction of flow can be indicated on the connectors.



Elements

The main elements that can appear in Block Definition diagrams are:

- Property
- Connector Property
- Distributed Property
- Flow Property
- Bound Reference
- End Path Multiplicity
- Signal
- Port

The main connectors that can appear in Block Definition diagrams are:

- Item Flow
- Connector
- Binding Connector
- Dependency

Tools

A variety of tools can be used with structural modeling and Internal Block diagrams, including:

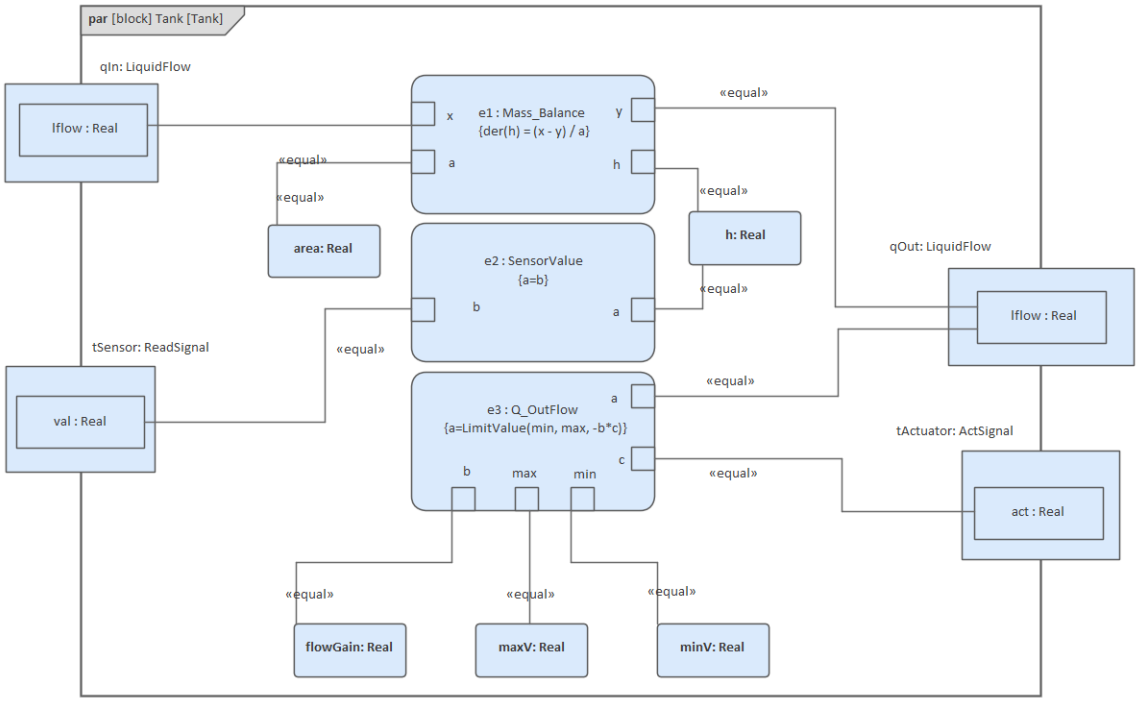
- Diagram Filters - which allows a user to filter elements out of the diagram to achieve a more specific focus
- Pan and Zoom - which allows a modeler or viewer to easily move around large diagrams
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats
- Traceability - which provides a hierarchical view of an element's relationships to other model elements
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios
- Relationship Matrix - which allows the connections between Blocks (or other elements) and other elements such as Requirements and Use Cases to be visualized in a matrix

Usage

The Internal Block diagram is used to model the internal structure of a block including its parts and the relationship between those parts.

Parametric Diagram

The SysML Parametric Diagram is a type of Internal Block Diagram (with some restrictions) that is used to model equation with parameters. They are an important tool that can be used to describe equations and their parameters and are important when performing trade off analysis and assessing design alternatives as they can be combined into systems of equations and related to Measures of Effectiveness MOEs.



Parametric diagrams describe the usage of constraint blocks and provide a mechanism for integrating engineering analysis such as performance and reliability and other factors of interest with other SysML models and diagrams.

Parametric diagrams define the way that constraint blocks are used to constrain the properties of another block. The usage of a constraint is said to bind the parameters of the constraint (e.g. F=m*a), such as F, m, and a, to specific properties of a block, such as a mass and acceleration, that provide values for the parameters.

Elements

The main elements that can appear in Parametric diagrams are:

- ConstraintProperty
- Property
- Objective Function
- Measure of Effectiveness

The main connectors that can appear in Parametric diagrams are:

- Connector
- Binding Connector
- Item Flow
- Dependency

Tools

A variety of tools can be used with structural modeling and Internal Block diagrams, including:

- Modelica Integration - which provides a mechanism for simulation,
- Diagram Filters - which allows a user to filter elements out of the diagram to achieve a more specific focus,
- Pan and Zoom - which allows a modeler or viewer to easily move around large diagrams,
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model,
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats,
- Traceability - which provides a hierarchical view of an element's relationships to other model elements,

Usage

The Parametric diagram can be used to show how the physical properties of a system are constrained by specifying a network of constraints that represent mathematical expressions such as {F=m*a} and {a=dv/dt}.

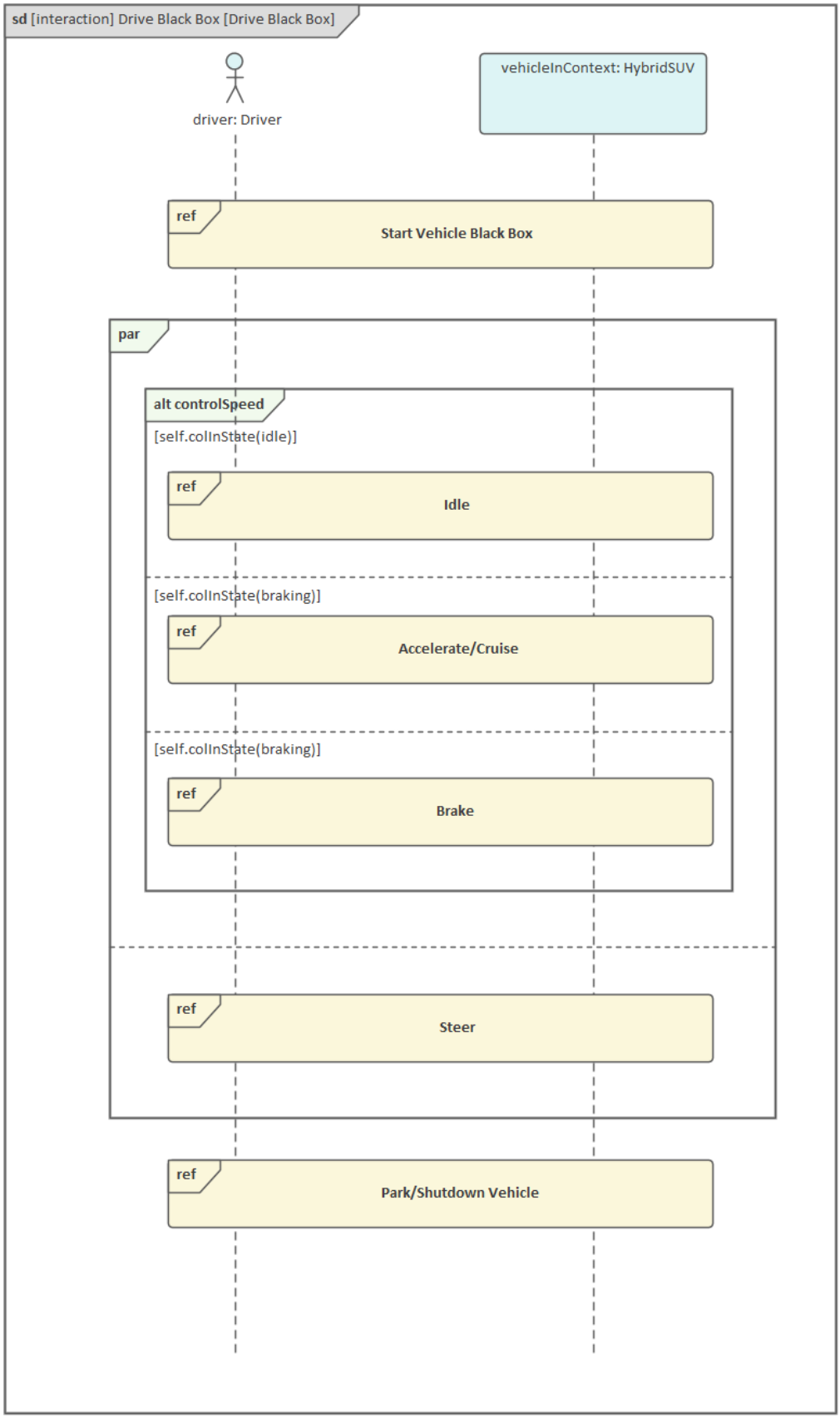
They can also be used for trade-off analysis, where a Constraint Block can define an objective function used to make a comparison between alternative solutions.

Critical performance parameters and their relationships to other parameters can be modeled, which can then be tracked throughout the system life cycle.

Sequence Diagram

A Sequence diagram is a type of Interaction diagram that shows the time ordered interaction between objects. The diagram has two axes; the vertical axis represents time and the horizontal axis represents the objects that take part in the interaction, typically ordered in a way that best illuminates the interaction. These diagrams have their origin in the modeling of software interactions, but they can be used with systems engineering to be prescriptive of how elements (such as Blocks) should interact, or descriptive in showing how they do interact, in practice.

This Sequence diagram shows the interactions and sequence of message flows between a driver and a vehicle. The diagram expresses the necessary interactions for the 'Drive the Vehicle' Use Case. The interaction is owned by the 'AutomotiveDomain' Block.


sd [interaction] Drive Black Box [Drive Black Box]
driver: Driver
vehicleInContext: HybridSUV
ref
Start Vehicle Black Box
par
alt controlSpeed
[self.colInState(idle)]
ref
Idle
[self.colInState(braking)]
ref
Accelerate/Cruise
[self.colInState(braking)]
ref
Brake
ref
Steer
ref
Park/Shutdown Vehicle

Elements

The main elements that can appear in Parametric diagrams are:

- Sequence
- Fragment
- Endpoint
- Diagram Gate
- State/Continuation

The main connectors that can appear in Parametric diagrams are:

- Message
- Self Message
- Recursion
- Dependency

Tools

A variety of tools can be used with behavioral modeling and Activity diagrams, including:

- Diagram Filters - which allows a user to filter elements out of the diagram to achieve a more specific focus,
- Pan and Zoom - which allows a modeler or viewer to easily move around large diagrams,
- Spreadsheet (CSV) Import and Export - which allows content in spreadsheets to be imported or exported from the model,
- Documentation - which allows formal or informal documentation to be generated from the model in a variety of formats,
- Traceability - which provides a hierarchical view of an element's relationships to other model elements,
- Responsibility Window - which provides a composite view of the important responsibilities of an element, including Constraints, Requirements and Scenarios

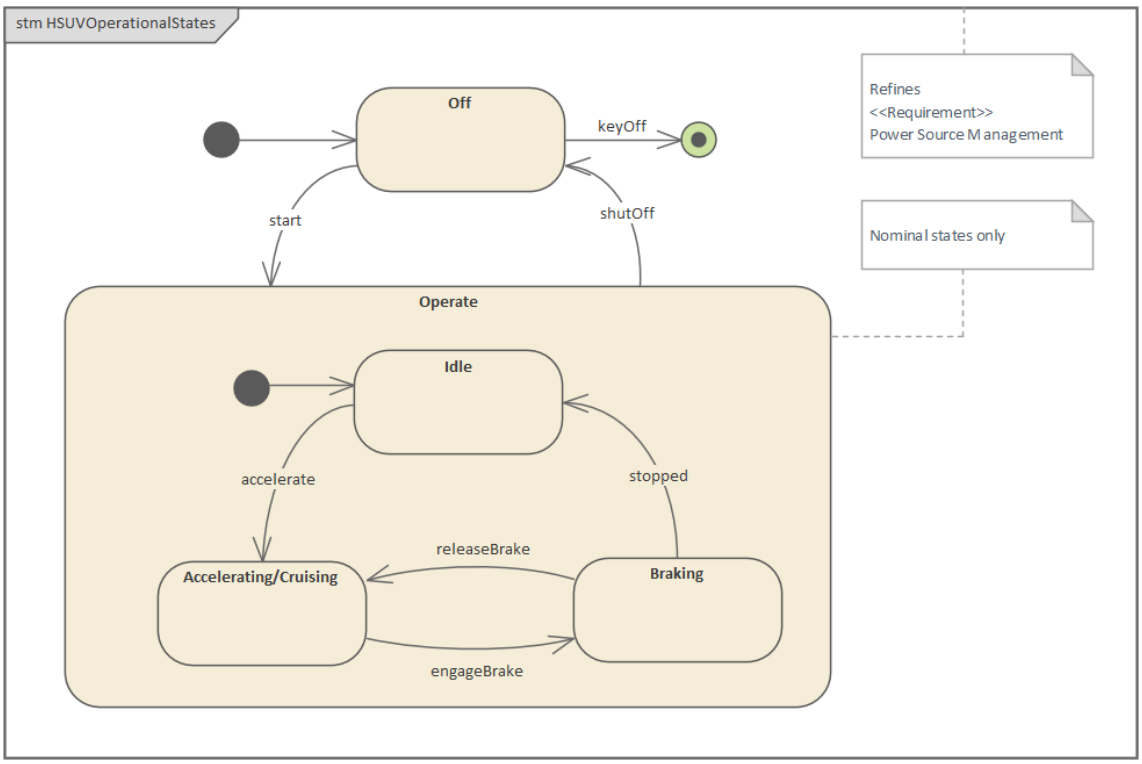
Relationship Matrix - which allows the connections between Activities (or other elements) and other elements such as Blocks, Requirements and Use Cases to be visualized in a matrix.

Usage

The very fact that we are modeling a system implies that it has a degree of complexity that can not be managed without the use of tools. Sequence diagrams can be used to represent system scenarios showing how parts of a system interact together to achieve some specified outcome. Messages are said to be exchanged between lifelines representing the lifetime of the object, the messages represent operations or signals sent and received by the objects.

StateMachine Diagram

A StateMachine diagram is an effective way of presenting information about the lifetime of a system element such as a Block. It can be used to describe the important conditions (States) that occur in an entity's lifetime or cycles. Typically only entities that have important stages in their lifetime are modeled with StateMachine diagrams. The entity is said to transition from one State to another as specified by the StateMachine. Triggers and Events can be described that allow the state transition to occur and Guards can be defined that restrict the change of state. Each State can define the behaviors that occur on entry, during and exit from the State.



Elements

The main elements that can appear in Parametric diagrams are:

- State
- StateMachine
- Initial
- Final
- Choice
- History
- Fork and Join

The main connectors that can appear in Parametric diagrams are:

- Transition
- Dependency

Tools

A wide variety of tools are available for working with StateMachine diagrams, in addition to the StateMachine diagram itself. These include:

- State Table Editor - Which allows the StateMachine diagram to be visualized in a table that, for some analysts, is easier to understand than a diagram; it contains the same information as the diagram and can be viewed in a number of different ways
- Dynamic Simulation - Allows StateMachines to be visualized, showing how an entity transitions from one state to another
- Executable StateMachines - As well as utilizing the simulation engine and allowing StateMachines to be visualized, provide a complete language-specific implementation that can form the behavioral 'engine' for multiple software products on multiple platforms

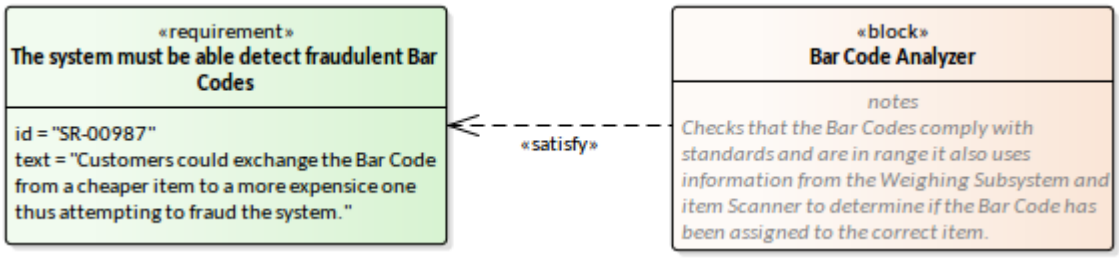
Usage

Complex systems are often composed of entities such as Blocks that have complex behavior and might have lifetimes that are difficult to understand.

StateMachines can be used to describe the important conditions (States) that occur in an entity's lifetime or cycles. Typically, only entities that have important stages in their lifetime are modeled with StateMachine diagrams. These diagrams provide insight into the way an entity transitions from state to state, ignoring conditions that are not important to the analysis.

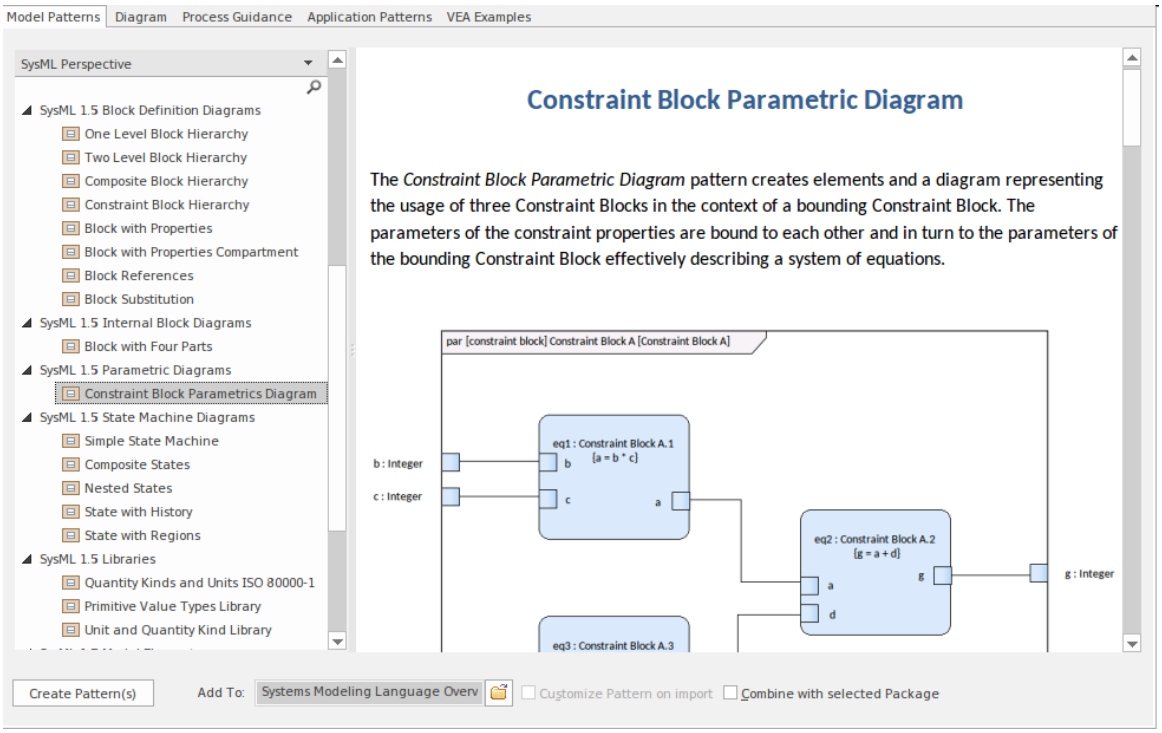
Systems Modeling Language Overview

Model Based System Engineering heralds a new era of communication and real time model collaboration. It brings with it the concept of the model as a machine rather than a cabinet full of documents. A machine that can do work such as validate requirements, generate parametric simulations of complex mathematics and physics equations, bring to life executable StateMachines and simulate business and decision logic, evolve in response to reviews, and create documentation, to list a few. These benefits are realized by the power of Enterprise Architect and because a standard and shared language is used to create the models - the Systems Modeling Language, commonly abbreviated to just *SysML*.

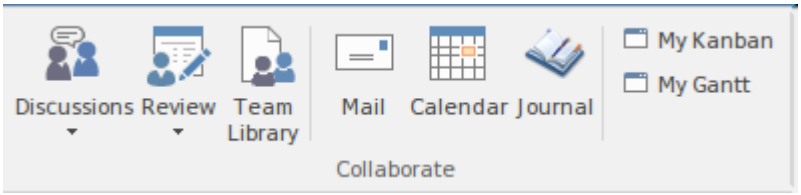


SysML allows both humans and machines to understand the models - the humans adding ingenuity, engineering and design and the machines performing the tedious and error-prone tasks such as validation, doing the heavy lifting such as generating parametric simulations and performing *what-If analysis,* and carrying out the more mundane tasks such as searching and report generation.

The acquisition of a language is not something that happens for free, but it is something that can happen without pain or frustration and without what some skeptics call a 'flair for languages'. Enterprise Architect will also be a friend that will assist you in learning the language by providing many in-tool devices to assist with the learning, and a rich and replete library of model patterns that will help you get started, ensuring you are creating industry best-practice models.



When you begin your journey with Enterprise Architect you immediately and effortlessly become part of an extensive international community of users and practitioners, who work with the tool day-in day-out to specify, design, implement and support system engineering models that are used to solve real world problems. Many of these problems and opportunities are complex and often seemingly intractable, but can be worked through by the collaboration of modelers applying the SysML to express and solve problems.

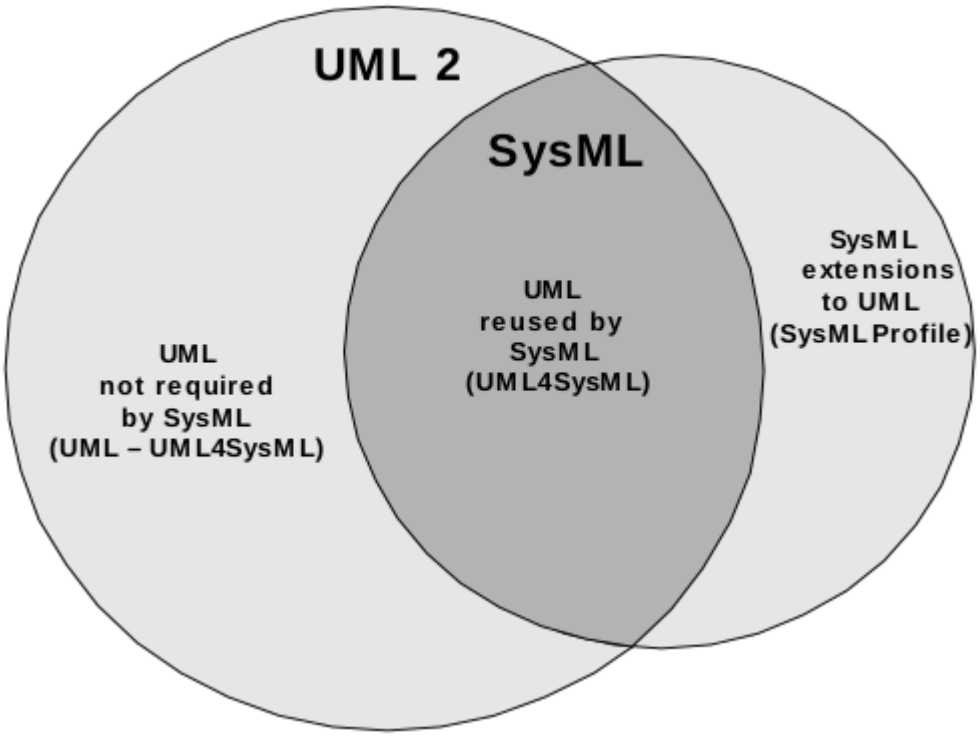


Enterprise Architect seamlessly facilitates this collaboration with its rich desktop and Cloud platforms that help to ensure the models are both robust and expressive, and the result of not just one but any number of engineers and other stakeholders, working together no matter what natural language they speak, what device they are using or where they are located in the world.

Language Architecture

The Systems Modeling Language (SysML) is fundamentally a set of conventional symbols that allow humans and tools to communicate about systems engineering. It is an international standard that defines and describes a general-purpose modeling language for systems engineering. Enterprise Architect is one of the world's leading tools that implements this standard and allows Systems Engineers to apply the approach that is known as Model Based Systems Engineering. In addition Enterprise Architect offers tool features that support a wide range of ancillary aspects of engineering practice and management. We will explore these useful and productive tool features throughout this guidebook.

The SysML is based on another standard, the Unified Modeling Language (UML), that has been adopted and used by Software Engineers since the late nineties. This is important, as many Systems Engineering projects involve both system and software aspects and so both system and software engineers are able to understand each others models, leading to greater transparency, less chance of errors and mutually intelligible language constructs, resulting in a system that is less likely to fail or exhibit faults. This Venn diagram shows the relationship between the two standards. SysML reuses the Use Case, Activity and Sequence Diagrams.



Requirements Driven

The creation of the Systems Modeling Language (SysML) was driven by user requirements; the design of the SysML responded to the needs set out in the *Request for Proposal for the Unified Modeling Language for Systems Engineering*. This document specifies a customization of UML for Systems Engineering (SE) and mandates that this customization should support modeling of a broad range of systems, which could include hardware, software, data, personnel, procedures, and facilities. The document states:

'The customization of UML for Systems Engineering should support the analysis, specification, design, and verification of complex systems by:

- Capturing the systems information in a precise and efficient manner that enables it to be integrated and reused in a wider context
- Analyzing and evaluating the system being specified, to identify and resolve system requirements and design issues, and to support trade-offs

- Communicating systems information correctly and consistently among various stakeholders and participants'

The designers of Enterprise Architect have read these documents and the resulting SysML specification in detail and created a sophisticated and highly usable tool that implements all these requirements and adds a rich set of additional features to ensure an organization's engineering and business success.

-

OMG Systems Modeling Language™

Version 1.5

For a language to be useful and relevant it must evolve in response to the needs of its communities of users. In so responding, the SysML specification is updated regularly and the teams at Sparx Systems also update and extend Enterprise Architect to ensure it complies with the evolving standard and, more fundamentally, meets the diverse needs of its community of users.

Unified Modeling Language Reuse and Extensions

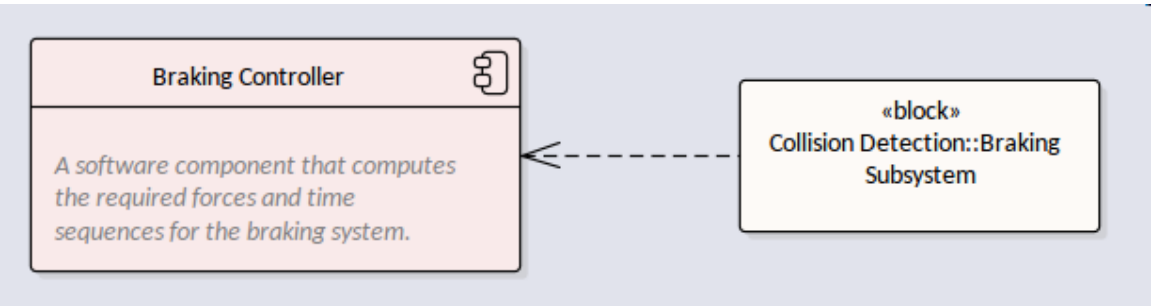
The Systems Modeling Language (SysML) is built on top of the Unified Modeling Language (UML). The UML had been ratified and adopted by the Object Management Group (OMG), who continue as the custodians of the specification. In 2005 UML was also published by the International Organization for Standardization (ISO) as an approved ISO standard. The language provided a specification for modeling software centric systems. The SysML language dates back to 2001 and had its origins in an open source specification, but when the International Council on Systems Engineering (INCOSE) began working with the OMG a final version of the SysML was adopted by the OMG in 2006.

In many ways SysML is theoretically the more primitive language as it is a general purpose modeling language, and UML is more specialized, being designed for modeling software centric systems. However, history and the languages' genesis has inverted this position. In practice, SysML has been created using the UML profiling system and is an extended subset of the UML. What this means is that SysML did not take all of UML and it also defined some additional language constructs. The Venn diagram we saw in an earlier section describes mathematically the two intersecting sets of language constructs.

The Enterprise Architect implementation of the SysML specification is highly compliant, with the developers working closely with the specification and in constant communication with industry experts, thought leaders and the systems engineering communities in a wide range of industries. This has resulted in a world class tool that not only implements the specification but also provides a wide range of additional tools such as Executable StateMachines, Parametric Simulations, Gantt Charts, Kanban Boards, Mind Mapping, Strategic models and literally hundreds of other features.

In addition there continues to be an increase in the interaction between system and software engineering problems and solutions in a wide range of disciplines, from rail systems to aeronautical systems, energy systems and many more.

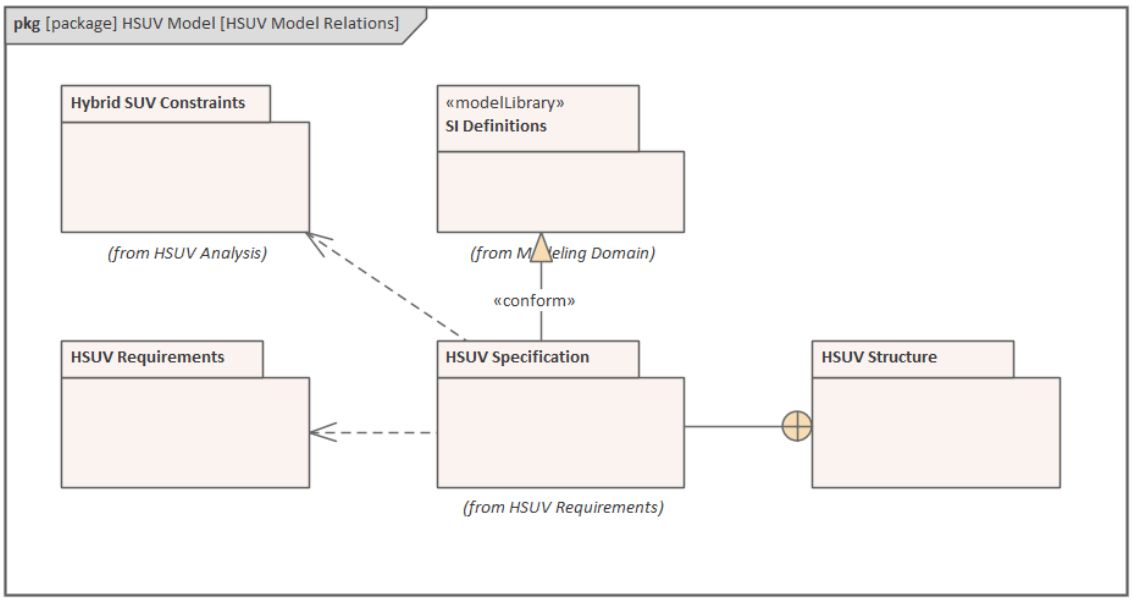
Enterprise Architect is uniquely positioned because of its formidable features supporting both these disciplines and also its strengths as an architectural tool.



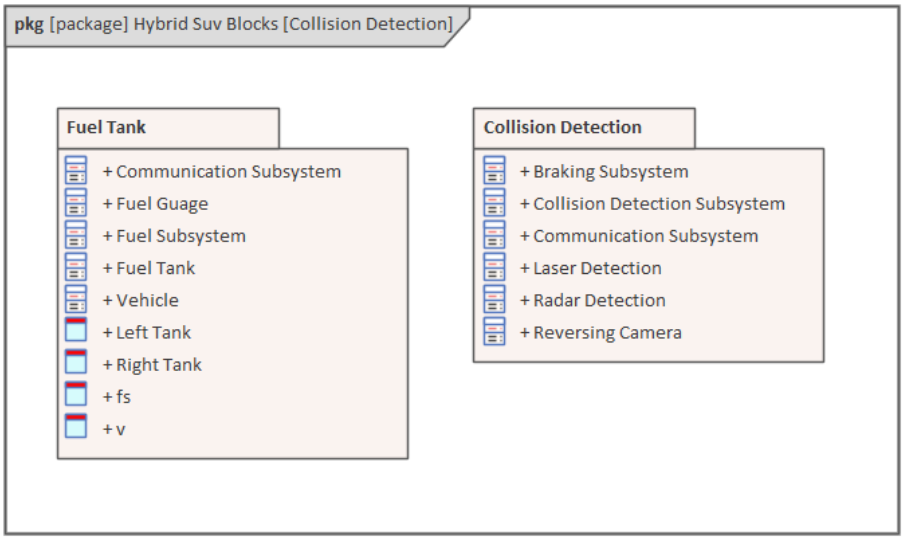
Partitioning with Packages

Packages are the fundamental unit of partitioning in the language and are designed to prevent circular dependencies. The language is formally partitioned into sets of model elements that group the elements logically and allow a language user to understand the elements as a collection of linguistic units.

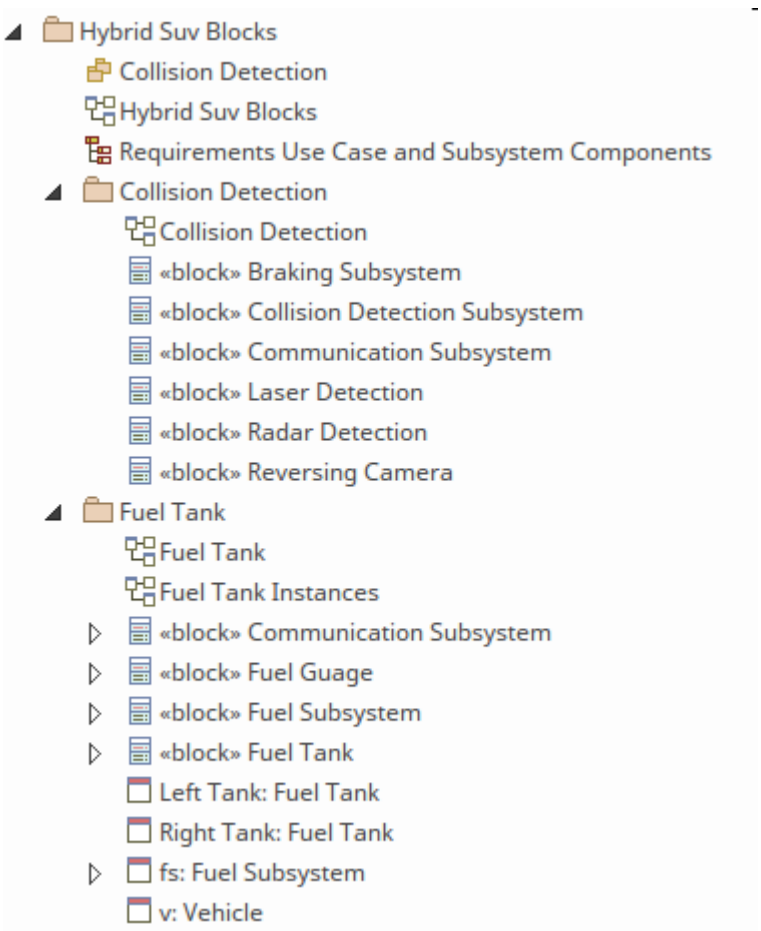
They are also the fundamental structural unit in user defined models and act as a general purpose mechanism used to group elements based on user defined factors. Formally they can be used to specify a namespace, which is important in some modeling constructs such as the definition of XML schemas or code generation. Packages can be created and viewed either in the Browser window or in diagrams, and both locations provide different ways to work with the Packages. Diagrams are useful for displaying the contents of Packages visually or to describe the relationships that exist between Packages.



Architect provides numerous ways to display Packages in diagrams that will assist users in understanding the structural relationships between Packages and the elements and diagrams they contain. When a Package is included in a diagram, the tool allows the user to choose from a number of display options and the compartment visibility can be changed to show the Package content. In this diagram the author wants to show the contents of two Packages that have significance in the unlikely event of a collision. The 'Show Package Contents' option has been selected in the element compartment visibility making it clear what elements are contained in each Package.

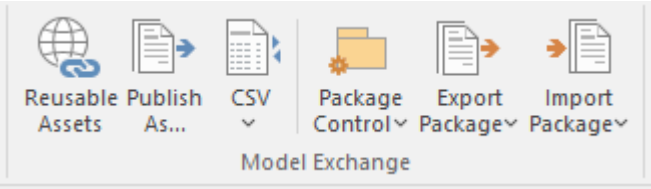


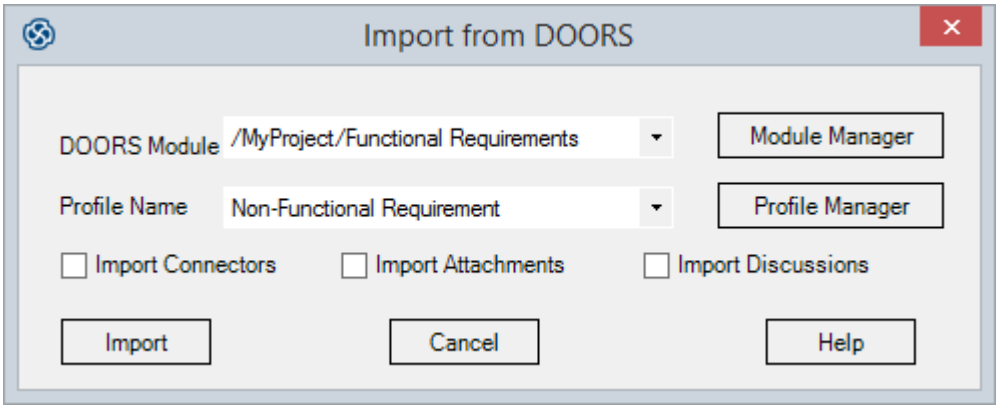
The same Packages and their contents can be viewed in the Browser window, it is important to remember that while it is possible to include the diagrams in publications such as reports, the contents of the Browser window would not be visible in these documents.



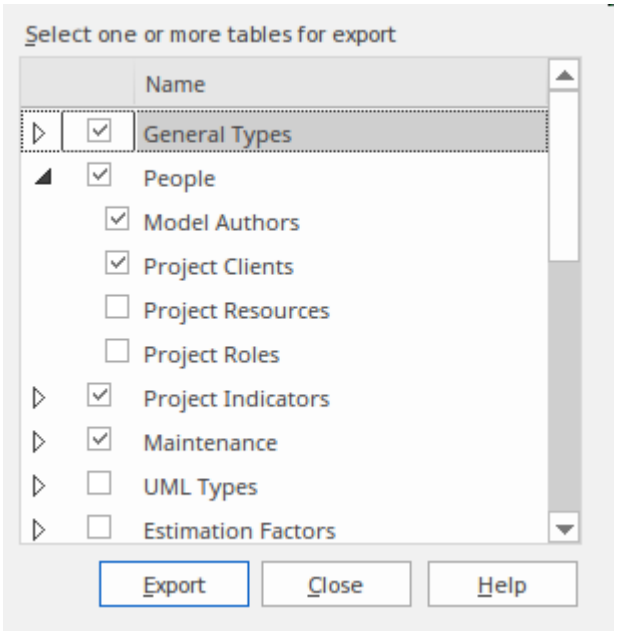
Interoperability and Model Exchange

Enterprise Architect is one of the leading SysML tools with a requisite set of features, but the designers are aware that organizations will have the need to use a variety of tools to accomplish the complex business and engineering tasks that confront every organization in the Twenty-First Century. To ensure that the important engineering and business information is available to be exchanged with other tools and platforms, there is rich support for model exchange in compliance with the ISO 10303-233 data interchange standard to support interoperability among other engineering tools. This is implemented based on the UML XMI interchange capability, which is supported in the tool at the Package level, allowing any Package and its contained hierarchy to be exchanged with other compliant tools.

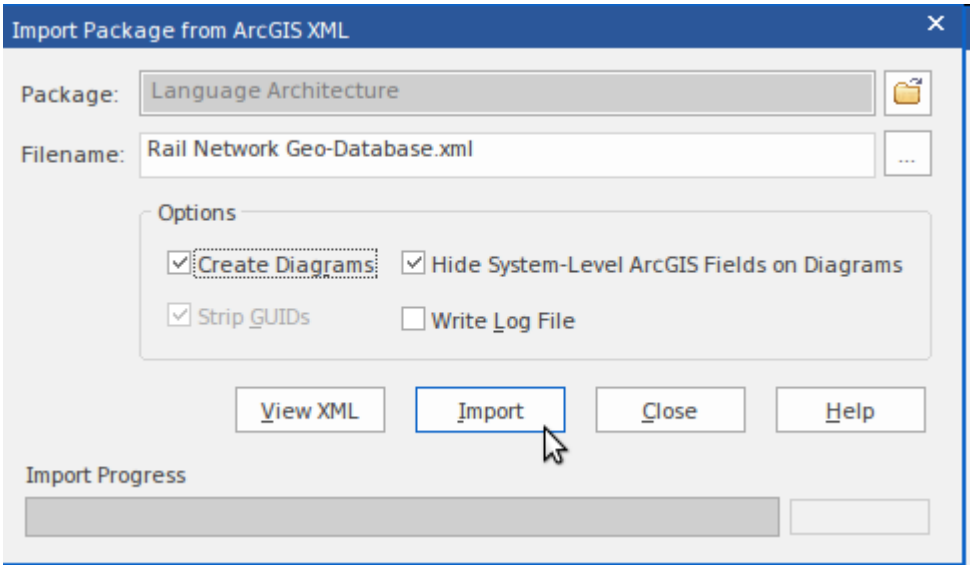




Enterprise Architect goes further than this and provides exchange mechanisms with a wide variety of business, project management, analytical and project delivery tools. This is achieved at the modeling tool level with the provision to exchange data contained in spreadsheets using the CSV file format, and text in word processors. Reference data such as lists of Priorities, Statuses, Complexities, Constraints, and other data such as Glossaries, Roles and Authors, Calendars and more can all be imported and exported from the repository.

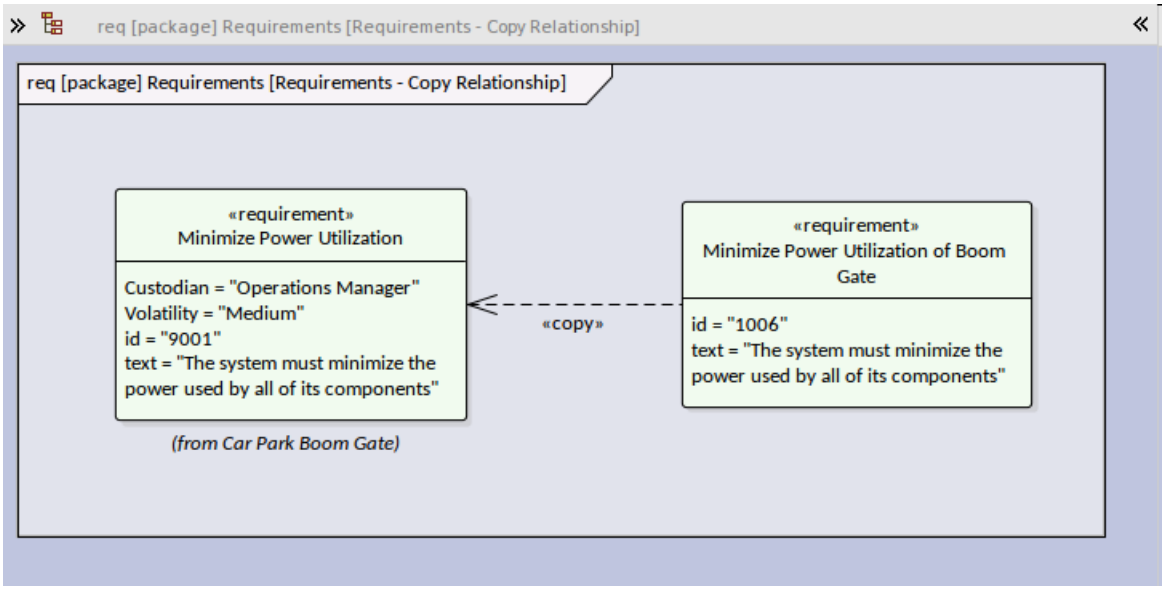


Geospatial information forms a critical data set in a world where geo-location is an important aspect of almost every project and initiative. Enterprise Architect provides a data exchange with the leading geospatial modeling tools, allowing two previously disparate and heterogeneous data sets to be viewed and managed together.

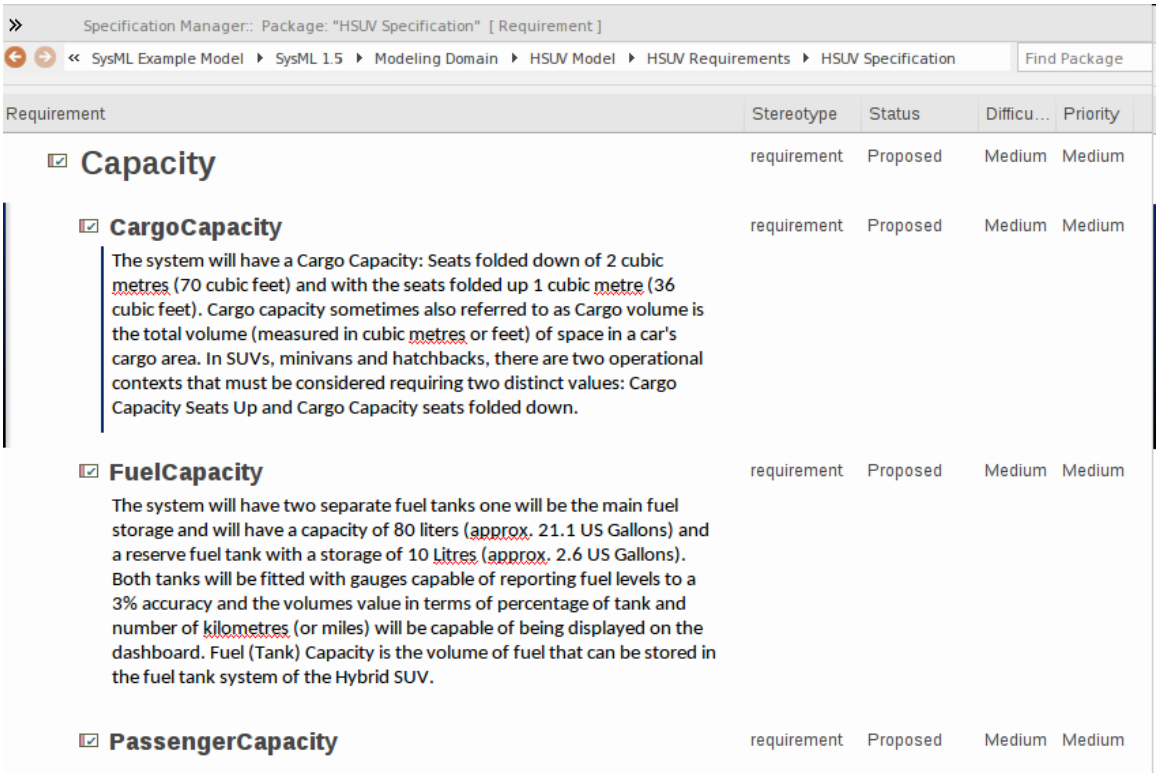

Import Package from ArcGIS XML
Package:
Language Architecture
Filename:
Rail Network Geo-Database.xml
Options
Create Diagrams
Hide System-Level ArcGIS Fields on Diagrams
Strip GUIDs
Write Log File
View XML
Import
Close
Help
Import Progress

Key Grammatical Concepts

The SysML, like its parent the UML, is a visual language where diagrams are central to the communication strategy of the language. Even though the emphasis of the language is on this visual codification and transmission of ideas, the language also has facilities to express ideas textually, which is an important complement to the visual mechanisms. A number of elements, such as the Requirement element, have a visual form, but the details of the Requirement are written in a property called Requirement 'text' as shown in this diagram.



Enterprise Architect has also been carefully designed to respect the way in which different users work with information. The design team works closely with its community of users and is aware that some users work better with diagrammatic visualization and others with text. Many of the tools available in Enterprise Architect have been designed with these different types of user in mind. For example, in the Requirements Management discipline, users are often more familiar with working in documents and spreadsheets. To cater for this Enterprise Architect has a number of views that users can switch to, that allow them to enter, edit and manage Requirements through these types of interface. One of these tools is the Specification Manager, which provides a flexible and familiar way of working with Requirements from both a document-based view and a spreadsheet view, allowing Requirements to be viewed, created and managed with ease.


Specification Manager:: Package: "HSUV Specification" [Requirement]
« SysML Example Model ▸ SysML 1.5 ▸ Modeling Domain ▸ HSUV Model ▸ HSUV Requirements ▸ HSUV Specification
Find Package
Requirement
Stereotype
Status
Difficu...
Priority
Capacity
requirement Proposed Medium Medium
CargoCapacity
requirement Proposed Medium Medium
The system will have a Cargo Capacity: Seats folded down of 2 cubic metres (70 cubic feet) and with the seats folded up 1 cubic metre (36 cubic feet). Cargo capacity sometimes also referred to as Cargo volume is the total volume (measured in cubic metres or feet) of space in a car's cargo area. In SUVs, minivans and hatchbacks, there are two operational contexts that must be considered requiring two distinct values: Cargo Capacity Seats Up and Cargo Capacity seats folded down.
FuelCapacity
requirement Proposed Medium Medium
The system will have two separate fuel tanks one will be the main fuel storage and will have a capacity of 80 liters (approx. 21.1 US Gallons) and a reserve fuel tank with a storage of 10 Litres (approx. 2.6 US Gallons). Both tanks will be fitted with gauges capable of reporting fuel levels to a 3% accuracy and the volumes value in terms of percentage of tank and number of kilometres (or miles) will be capable of being displayed on the dashboard. Fuel (Tank) Capacity is the volume of fuel that can be stored in the fuel tank system of the Hybrid SUV.
PassengerCapacity
requirement Proposed Medium Medium

Models, Diagrams, Elements and Views

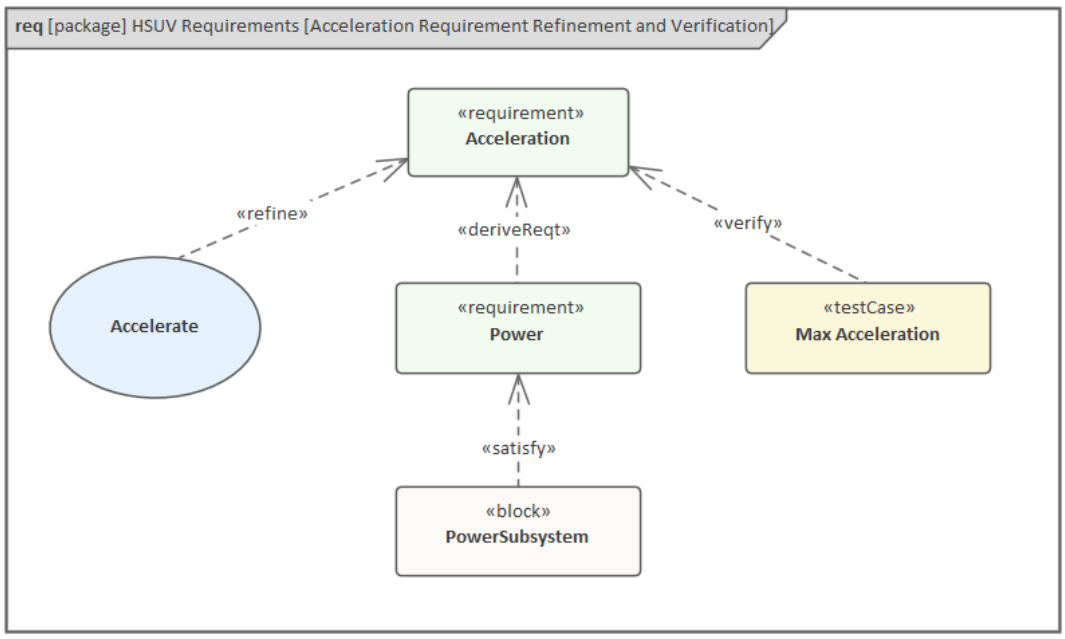
Models

The word model is quite overused and over-loaded with meaning. Some people use model to signify an entire repository whereas others use it to refer to a section of a complete repository. Models are the structural divisions in the repository.

Diagrams

The SysML specification defines nine diagram types. This is the canonical list and the SysML in addition defines the elements that are typically used on each diagram. Many newcomers and even some experienced users are not aware that even though these lists of elements describe the commonly used elements for a particular diagram type, that does not preclude a modeler from using other elements on these diagrams. In fact, using a number of element types on the same diagram results in an expressive model and allows stakeholders and engineers from different disciplines to understand the inter-diagram connections between the models.

In this section we will also learn that there are a number of 'universal' model elements that the specification suggests can be incorporated as part of any diagram, including Comments, Constraints and Rationales. This diagram shows a range of element types including a Block, a Use Case, a Requirement and a Test Case, all expressed on a Block Definition Diagram (BDD).

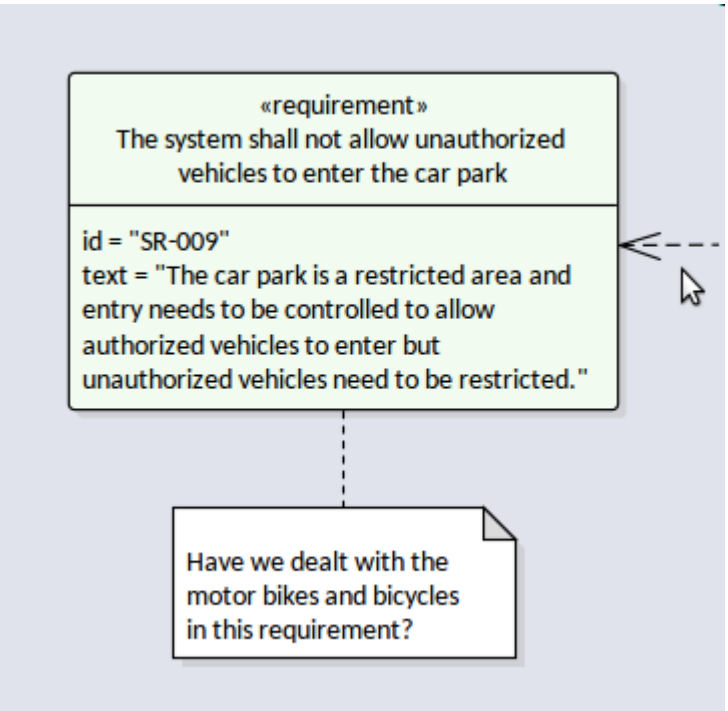


As stated earlier, the Systems Modeling Language specifies nine different types of diagram.

Elements

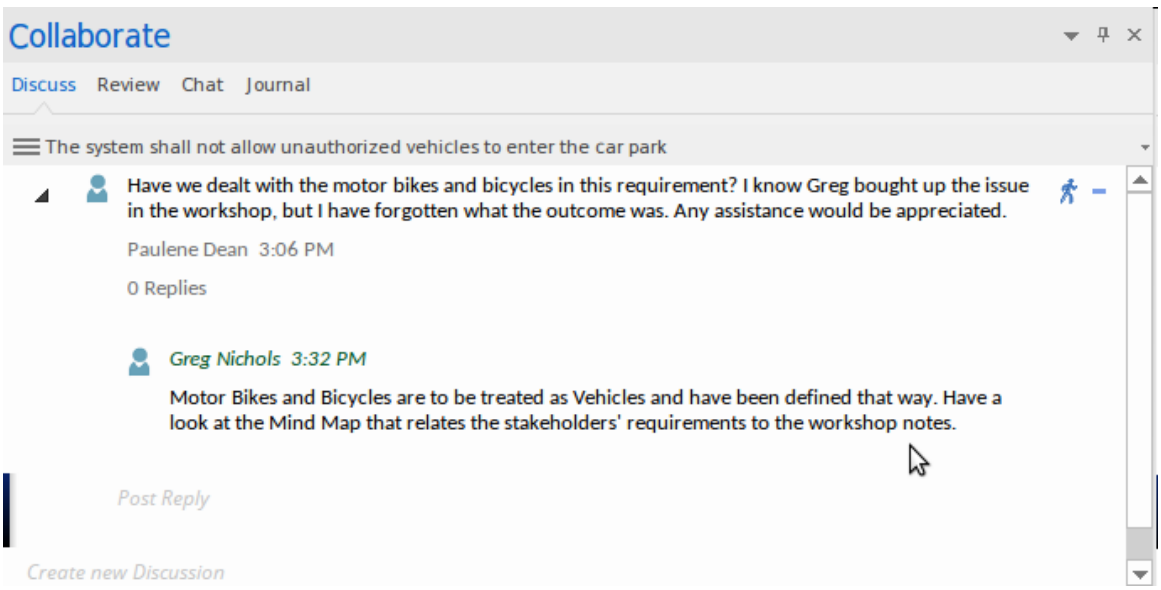
The nine diagrams described earlier are individually specified to convey a particular aspect of an engineering opportunity or solution; for example, the Parametric diagram is intended to show how equations are constructed. There are, however, a number of types of element that are universal to the model endeavor and can appear on any type of diagram. Many of these elements are used on diagrams to convey important annotations to a model or to help explain a particular aspect of

the model. They include elements such as Notes, Constraints, Rationales and Views. In this diagram a stakeholder who has been viewing the model has added a comment to question a part of the model.



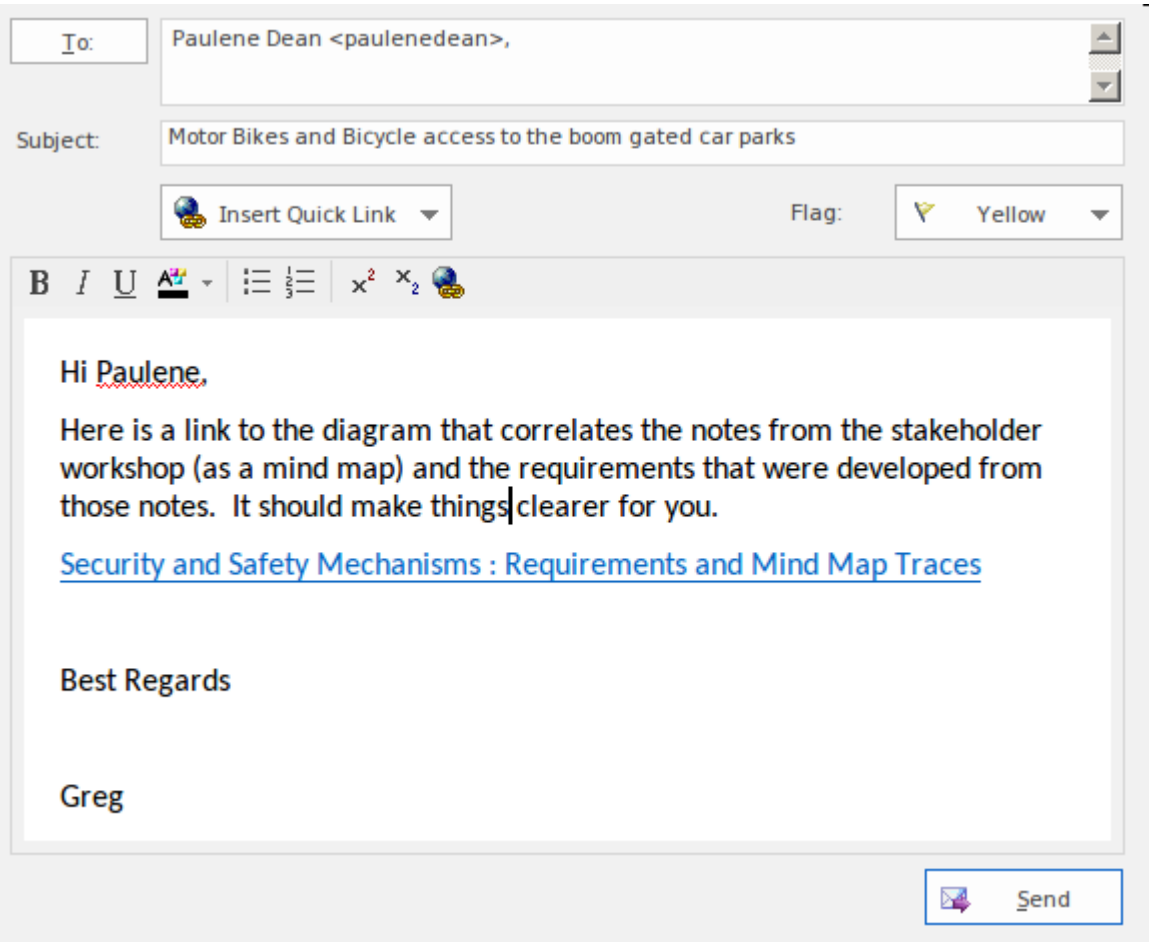
For more information, see the Common Pages Help topic.

While Enterprise Architect is highly compliant with the SysML specification it has a number of collaborative features that allow such comments to be managed, such as through its Discussion feature. This allows discussions to be kept separate from the elements proper that comprise the model. This screen image shows the same comment added using the Discussion feature, which allows replies and a range of other collaborative devices.



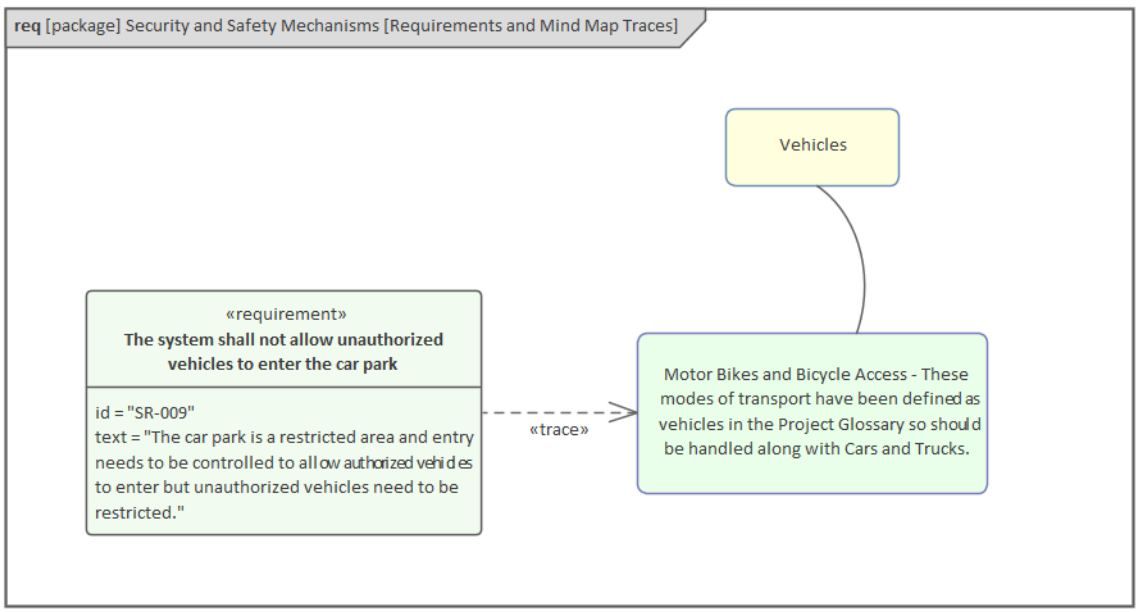
For more information, see the Discussions Help topic.

The responder could also have sent a Model Mail message, which might include a link to an Enterprise Architect element or diagram as shown.



For more information, see the Model Mail Help topic.

When the user opens the mail message, they are able to follow the link and open the diagram referenced in the link. This is a useful mechanism allowing dynamic and real-time views of the modeling information to be referenced and accessed rather than an image sent in a static document. This image shows the diagram that would be displayed in the tool when the link is opened.



The ability to link elements from different modeling domains is one of Enterprise Architect's great strengths as a unifying platform for teams and disciplines, and nowhere is this more true than in the relationship between business strategy and

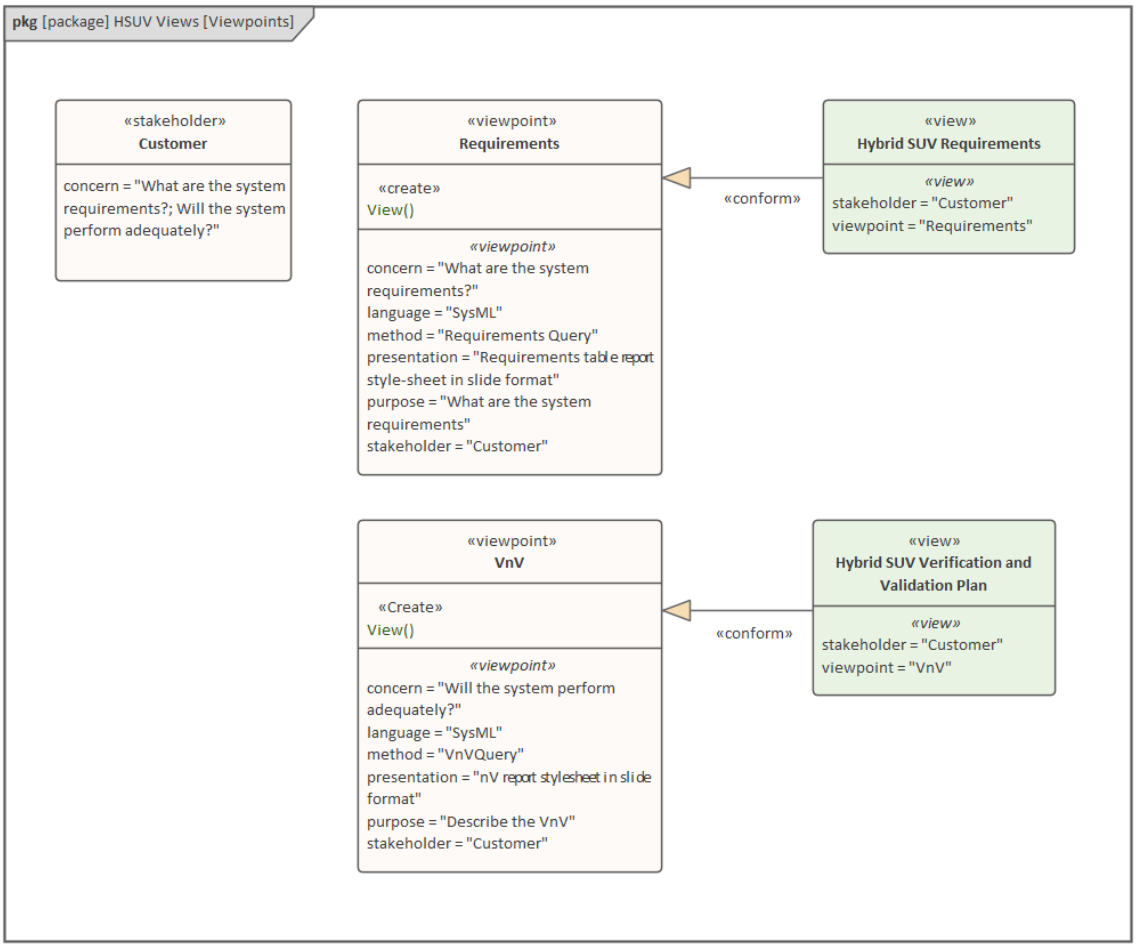
engineering, and between software development and engineering. The result is a consistent and harmonized model where the possibility of faults resulting from seams between different teams is significantly reduced. For more information see the Traceability Window Help topic.

Views

Fundamentally a system is conceived of, analyzed, designed and built for its stakeholders. Systems Engineers gather concerns and interests from stakeholders and apply analysis to create requirements and constraints. These are used as input for analysis and design and before the system is delivered for validation and verification. Stakeholders need to be able to visualize how their interests are being addressed at various stages in the engineering process and this visualization can be provided by views and viewpoints. The concepts of viewpoint and view are articulated in ISO-42010 (formerly IEEE-1471) and the SysML specification was written to be consistent with the ISO-42010 standard. There are a number of commonly used viewpoints including:

- Operational
- Performance
- Manufacturing
- Security

A viewpoint is a prescription for constructing a view that will address the needs, interests and concerns of a given stakeholder. A view is what the stakeholder sees from a given viewpoint and should enable them to visualize the parts of the system that are of concern to them while leaving out, or obscuring, the aspects of the system that are not of interest.



In addition to the formal mechanism described by the SysML specification in the form of View and Viewpoint elements, Enterprise Architect has a wide range of tools to assist with the creation and management of viewpoints, views and representations. There are several tools that can be used to create different views of the elements in the repository; these

include the Working Sets and the Model Views. Working Sets allow a collection of diagrams, Matrices, Model Libraries and other items to be saved and reopened as a set, which is useful when working with different groups of stakeholders. For more information see the Working Sets Help topic

Model Views can be used to create views of elements grouped together irrespective of their location in the Browser window. There are also several tools that can hide or obscure parts of a diagram to make it more appealing to a particular audience. The appearance of diagrams can be altered by changing the appearance of elements, including using an image, and Diagram Filters can obscure or hide elements from view. For more information see the Visual Filters Help topic. The Documentation engine can create high quality publications directly from the model into pdf or docx formats. For more information see the Documentation Help topic.

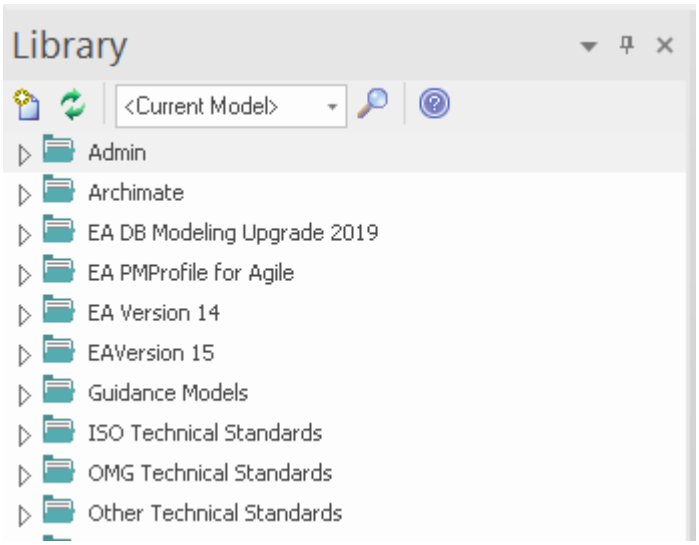
Collaborating as an Engineering Team

An engineering team is multidisciplinary and consists of strategists, managers, system engineers, software engineers, testers and others. The commercial pressures to release a product or provide a solution means that teams have to work more cleverly and cohesively to ensure engineering outcomes. Enterprise Architect has been built from the ground up as a collaborative platform, not just for engineers but for all disciplines. It facilitates individuals and teams working together and sharing information, models, designs and solutions with a full range of tools from discussions, reviews, a team library and chat to Version Control and Baselines.

Central Shared Repository

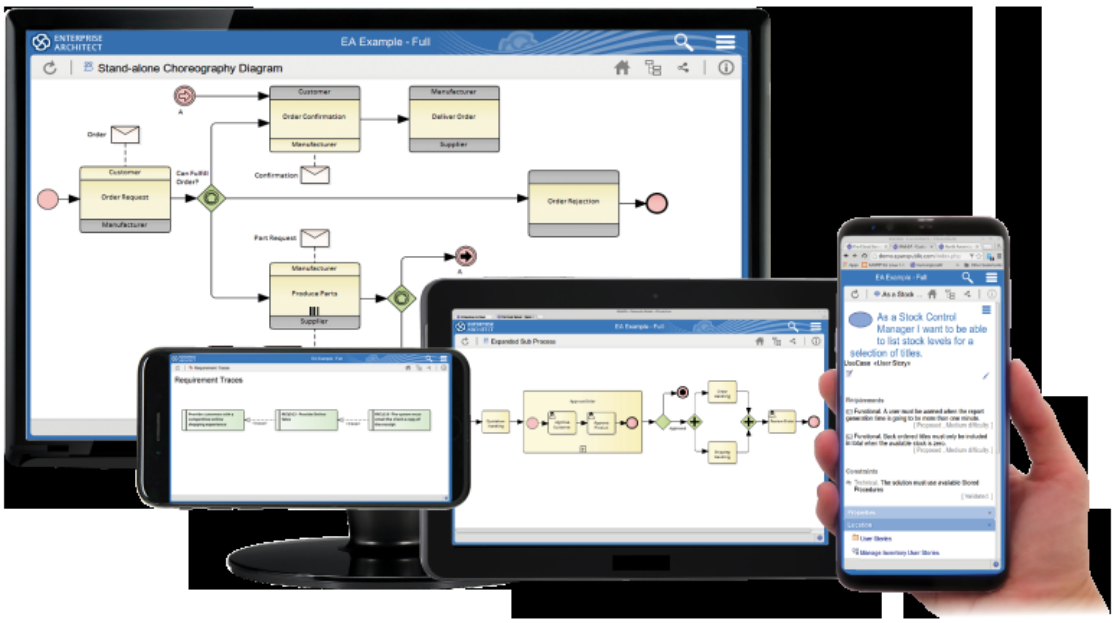
Enterprise Architect is not just a sophisticated drawing tool but a modeling platform that allows models to be validated, simulated, transformed and manipulated programmatically. This can be achieved because the tool does not store the diagrams in raster or vector format but rather the images are coded in a relational database along with all the reference data and other repository metadata. The platform uses this relational database to store all modeling information and metadata and this provides the back-end storage that is accessed by client and web based tools. For more information see the The Model Repository Help topic.

This product architecture allows users to share models, diagrams and other repository information such as reference data, images in the Image Library and documents in the Model Library and many other tool features that facilitate collaboration and working together in a co-located or distributed team. For more information see the Help topic The Modeling Team.



Cloud Computing

The central repository described in the previous topic can be accessed directly from the Enterprise Architect desktop client, using appropriate database connectivity layers (ODBC) and the accompanying drivers. There is, however, another easier way to access a repository hosted in a Cloud, and that is via Cloud services using the http or https protocols. The Cloud can be on-premise or off-premise and there are many ways the Cloud could be configured. There is also a Server comprising a number of modules that can be purchased, called the Pro Cloud Server, which provides a variety of tools, facilities and ways to view the models via a browser on a computer or hand held device, such as a tablet or smart phone. For more information see the Pro Cloud Server Repositories topic.

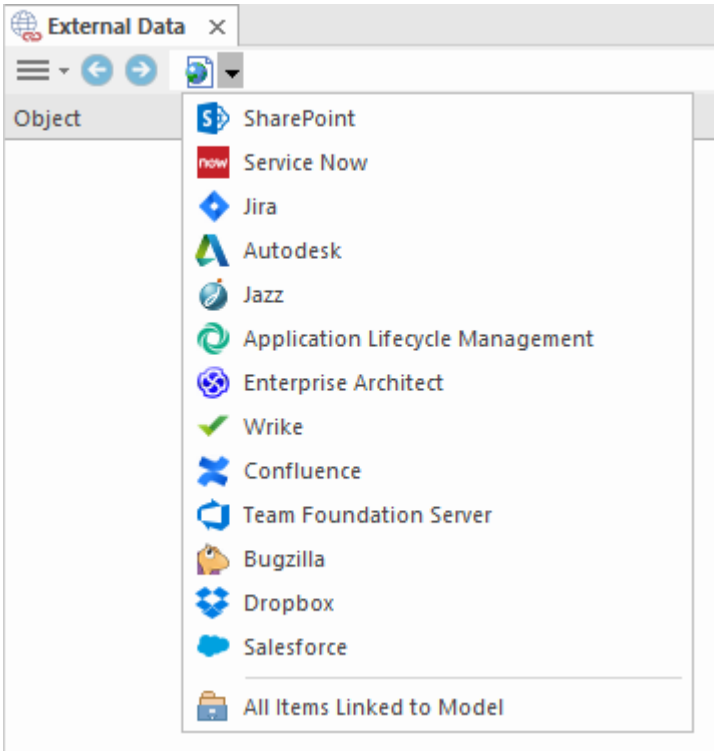


This provides a platform for working with a variety of stakeholders, from Engineering Managers to Consulting Engineers, locally located or distributed across the globe. The power of this way of working can be realized with minimal set up and the benefit of having all team members and stakeholders viewing and contributing to the same models cannot be underestimated.

Productivity gains can be achieved by being able to get valuable and timely feedback, or to discuss a diagram with an engineering consultant while they are traveling on a train to a trade show, or from the engineering lead while on a break from a symposium - all in real time and within the model from a smart phone or tablet.

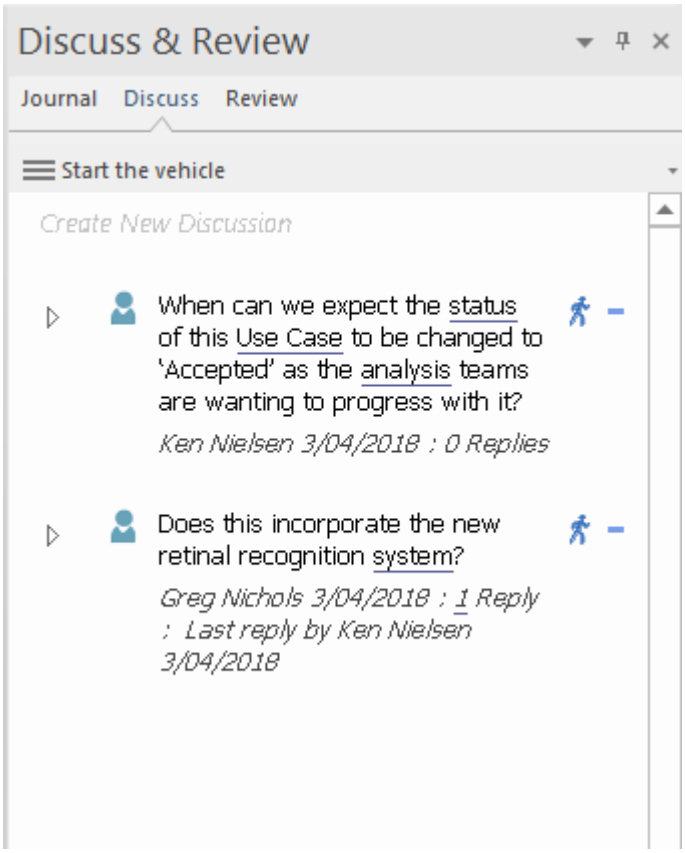
The Pro Cloud Server also provides connectivity to a wide range of other tools and platforms, spanning Requirements Management systems, Configuration Management Tools, Issue Tracking Systems, Project Management Systems and more.

This screenshot shows a list of some of the available integrations, although this list is being added to regularly so it is worthwhile checking whether additional integrations have been included. For more information see the Integrate Data from External Providers topic.


External Data
Object
SharePoint
Service Now
Jira
Autodesk
Jazz
Application Lifecycle Management
Enterprise Architect
Wrike
Confluence
Team Foundation Server
Bugzilla
Dropbox
Salesforce
All Items Linked to Model

Discussions and Chat

Central to the notion of collaboration is a modeler's ability to discuss and chat with colleagues or industry and standards specialists about a problem or solution. Enterprise Architect allows engineers, managers and others to enter into discussions about elements, diagrams and connectors. A post can be created that starts a thread or conversation which other modelers can then enter into by replying. The discussions are kept separately from element and diagram meta information allowing rich and constructive comments to be made without affecting documentation or reports generated from the models. The discussions and chat are two of the options available, discussions from the Discuss & Review window and chats from the Chat & Mail window.

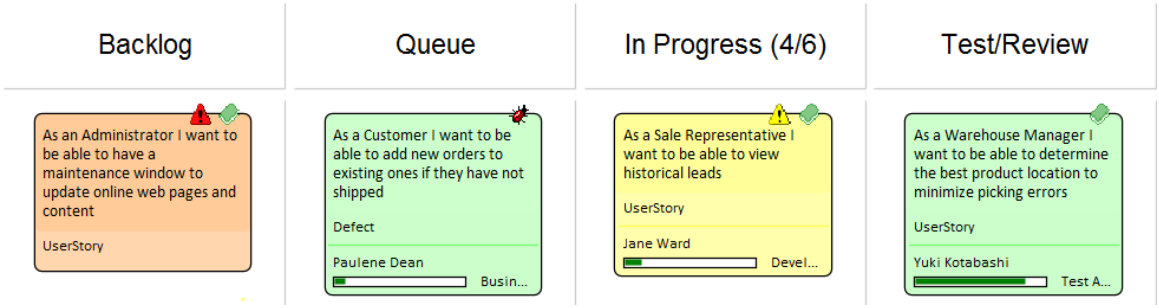


Chat is useful for quick and responsive communication with colleagues or experts that have been defined as part of a security defined group of users. Chats are not related to model elements in the way that discussions are but rather are global and when the Chat & Mail window is opened and a group is selected, the items are listed in date-time order. For more information see the Teams & Collaboration Help topic.

Kanban Resources and Calendars

The Kanban technique has been implemented in Enterprise Architect in a way that will greatly enhance the productivity of your team and the project management of software and system engineering projects or sets of tasks. Within Enterprise Architect it is a simple to use feature, enabling you to manage items in a backlog and move them into any number of lanes, or even to other boards, representing stages in a process. The facility can be incorporated into existing or new engineering or software development processes, resulting in unprecedented efficiencies.

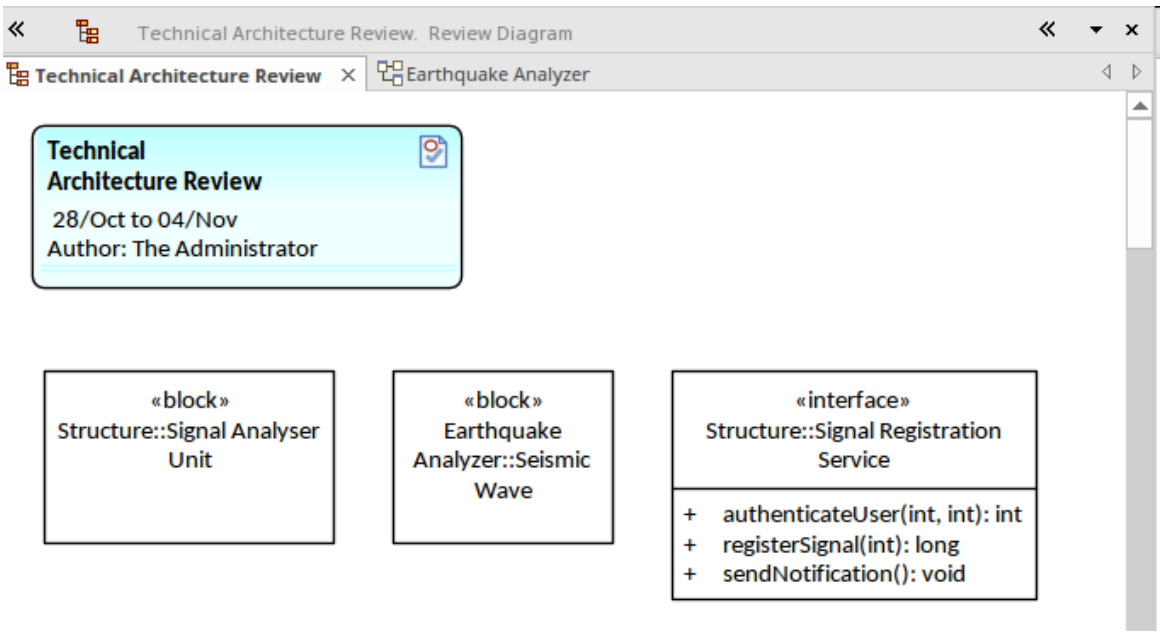
One of the great advantages of using this feature is that elements that participate in the Kanban diagrams can be linked to other elements in the repository, allowing full traceability from, for example, a requirement up to a strategic intent or to a component of a design and down to an element of a released product. Kanban can be used to visualize the resources, effort and state of completion of items as they move through any number of linked Kanban boards. This is an example from a software development process related to a warehouse systems engineering project.



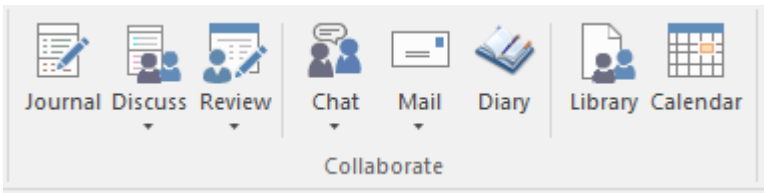
For more information see the Kanban Boards and Resources & Work Items Help topics.

Model Reviews

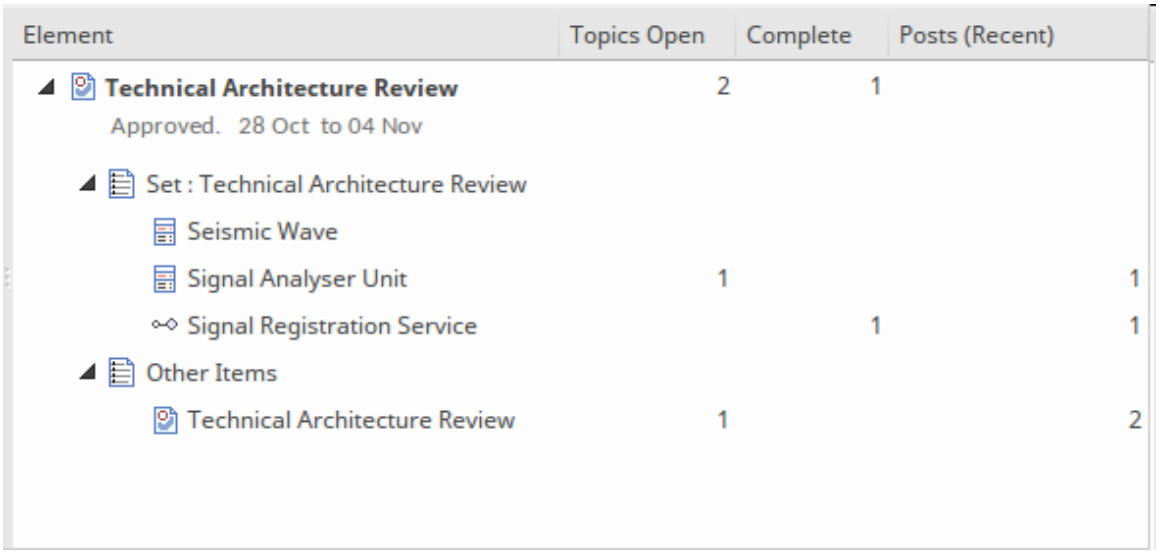
Model Reviews provide an ability for project stakeholders to collaborate formally in the assessment (review) of model content, including elements and diagrams. This handy collaboration tool utilizes a number of built in features - such as the Review view - to manage the process of the review and to visualize discussions and contributions to the review. A review diagram provides a mechanism for participants to add elements and diagrams to the review. This diagram contains a number of elements related to the review topic.



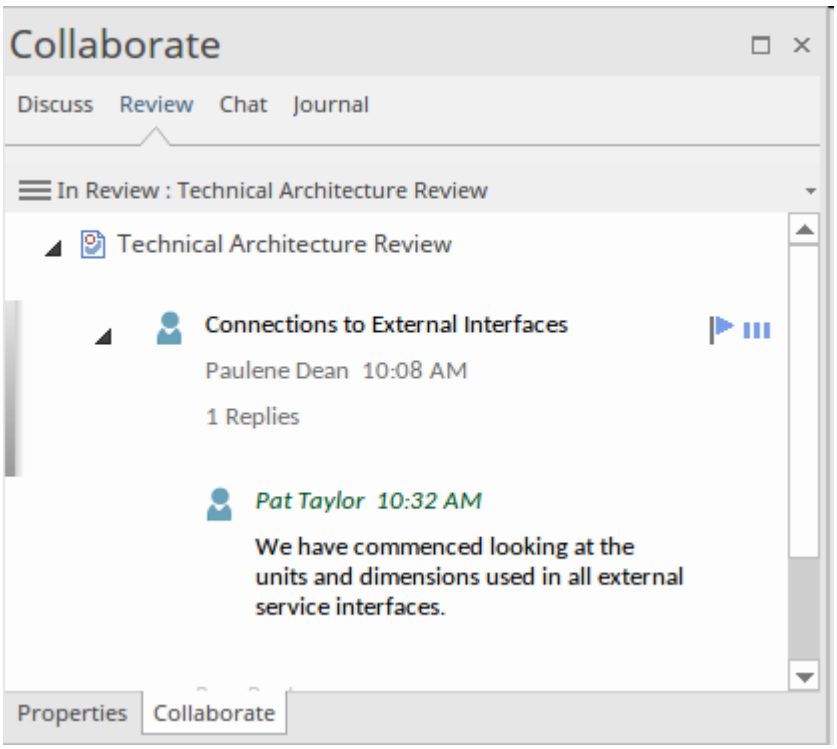
Any number of reviews can be created and modelers can join and participate in the reviews. The launching pad for the review facility is the 'Collaborate' panel of the Start ribbon.



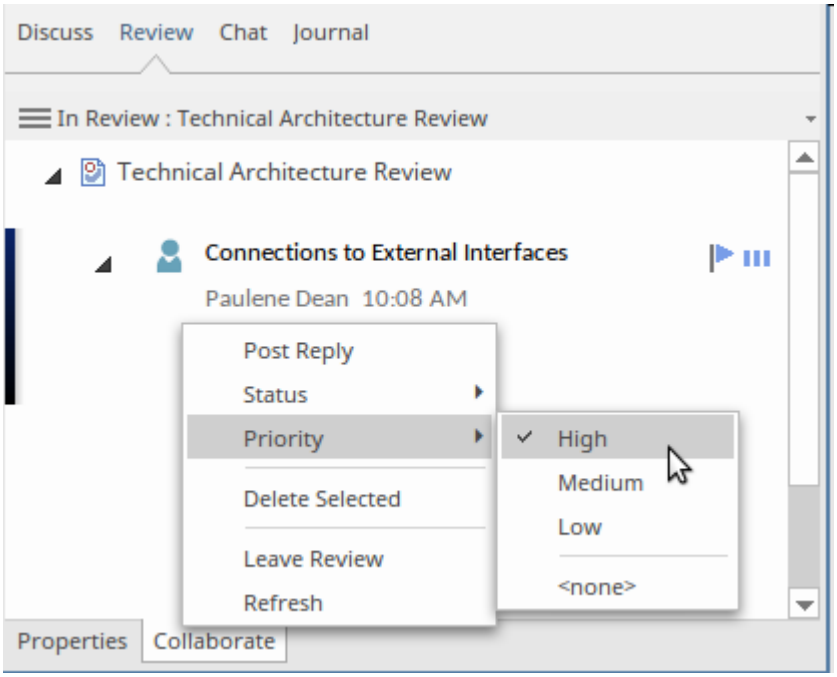
Selecting the 'Review > Manage Reviews' option will open the Reviews view, where all existing reviews will be listed and - with a right-mouse-click - new reviews can be created. The view shows the Review metadata in the left hand panel (including start and end dates) and the review details in the right hand panel, including the element and diagrams that form part of the review and the posts and replies for each item.



The review comments can be entered and viewed in the 'Review' tab of the Discuss & Review window. These will keep a running tally of all the posts and replies annotating the post, and replies with the author and the date. In this way highly collaborative outcomes can be achieved and - through the WebEA product available on smart phones and tablets - stakeholders such as external consultants and industry experts could contribute to the review without having to use the Enterprise Architect desktop application.



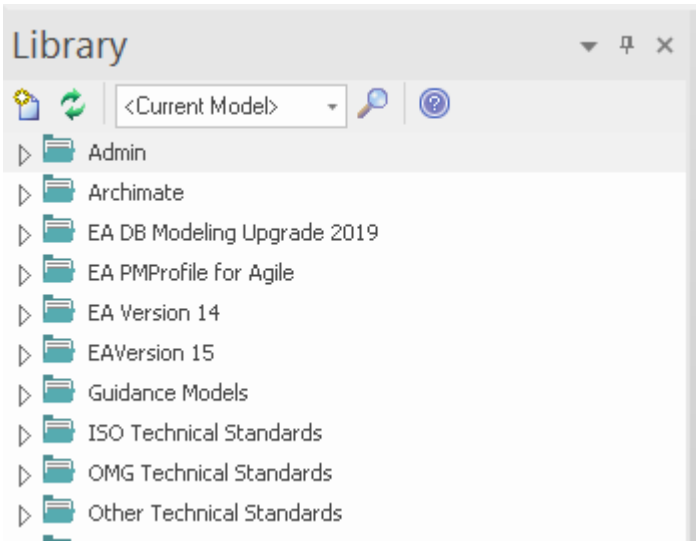
Each of the items in the Review view can have a number of properties set, including the Status and Priority of the review item; these can be seen as two small icons (a flag and a quantity icon) to the right of the item.


Discuss
Review
Chat
Journal
In Review : Technical Architecture Review
Technical Architecture Review
Connections to External Interfaces
Paulene Dean 10:08 AM
Post Reply
Status
Priority
Delete Selected
Leave Review
Refresh
High
Medium
Low
<none>
Properties
Collaborate

Sharing Resources in the Model Library

Teams that are using processes centered around Model Based Systems Engineering will invariably rely on a vast array of documents in the form of policies, methods, instructions, process descriptions, guidance documents, standards and other types of engineering or project documentation. Some of these will be document based and others will be resource based and available on an internal network, shared system, an Intranet or Extranet or more broadly the public Internet.

Regardless of where the documents or pages are located they can be either imported into Enterprise Architect or referenced as external resources via a URL. They can be included on a diagram as an Internal or External Artifact but more conveniently they can also be imported or referenced from the Model Library.



Consideration could also be given to importing some, or all, of the items in these documents as first class model elements. For example, a policy could be created as a metamodel element and the list of policies could be imported, allowing individual policies to be traced to particular system components. For more information see the The Model Library Help topic.

Viewing Models on Mobile Devices

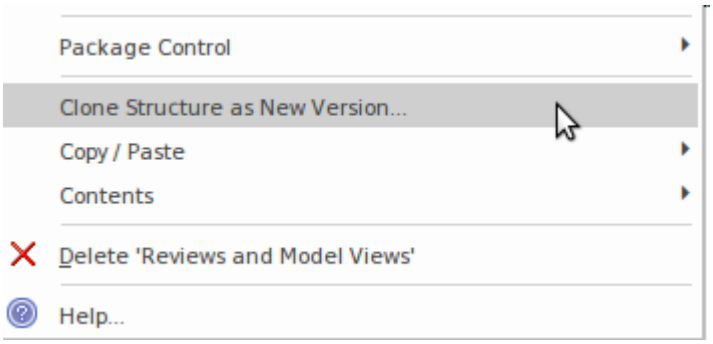
The modern workplace has changed significantly in recent years, with organizations encouraging flexibility in the form of hot desks and working from a home office, leading to more and more people working on portable devices. Also, the pace of change in our modern world increases every year, being driven by innovation and disruption - for example, a pandemic affecting every country in which suppliers, customers or colleagues are based, requiring people to be absent from offices everywhere, and unable to travel to consult or deliver expertise directly. . Strategists, Technologists and Engineers need to collaborate to achieve engineering outcomes and in a dispersed workforce this typically means they need to contribute to models from mobile devices, both while on the move and under restricted isolation.

Enterprise Architect repository content can be viewed in real-time through a browser on a mobile device such as a tablet or smart phone. This allows engineers, managers and others to collaborate while they are between meetings, at offsite inspections, on public transport or anywhere they happen to be. Never before has this been possible, and now the velocity of a project does not need to be slowed while waiting for reviews or for people to return to the office; the models can be updated at Internet speed from anywhere. For more information see the Pro Cloud Server Repositories Help topic.

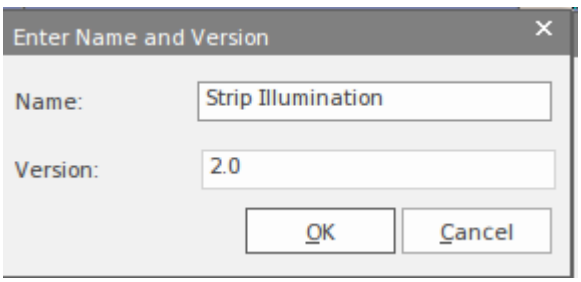
Modeling the Future

The world is being driven by an almost insatiable appetite for change and innovation and this has resulted in Systems Engineers needing to work smarter and faster and come up with clever ways of solving problems. *Time Aware Modeling* is a unique modeling facility that allows engineers and other stakeholders to model any number of future states. This introduction of state into the models provides a mechanism to visualize what a proposed solution might look like, and allows engineers to compare a number of proposed solutions. A given solution can be analyzed and reasoned about and potential pitfalls and problems can be identified in the models. Reviews and walk-throughs can be carried out by any number of stakeholders and these can be used to determine which of a number of solutions is suitable.

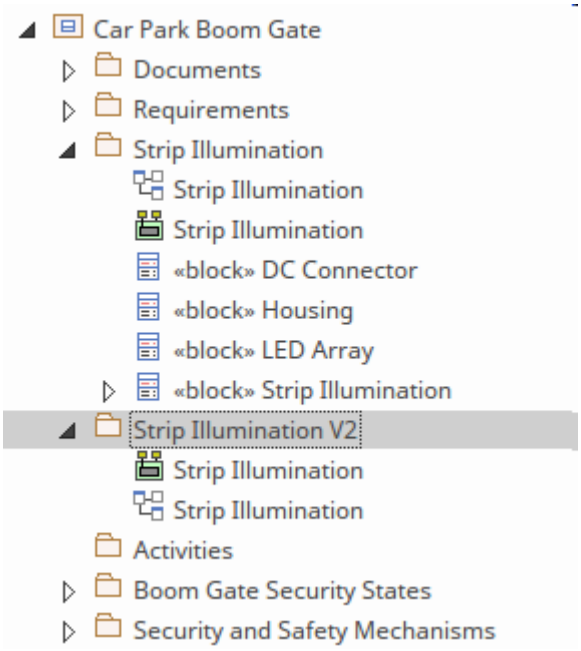
The process can be initiated by making a clone of a Package for which you need a future state model. This can be done using the ribbon options or from the Browser window context menu.



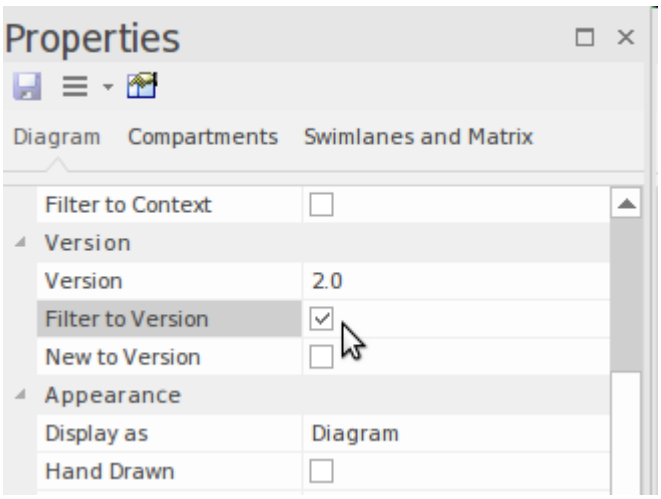
The tool will display a prompt allowing the engineer to specify a *name* and a *version number* for the cloned structure. The *version number* is critical to the operation of the feature and is used by Enterprise Architect as a way of tracking elements and diagrams that form part of this future version with the specified version number. The tool will also prompt for a location within the repository for the cloned Package structure. Typically it can be given a name that includes its version and the contained-in parent of the Package being cloned; it might also, however, be kept in a separate part of the repository designated for future state versions.



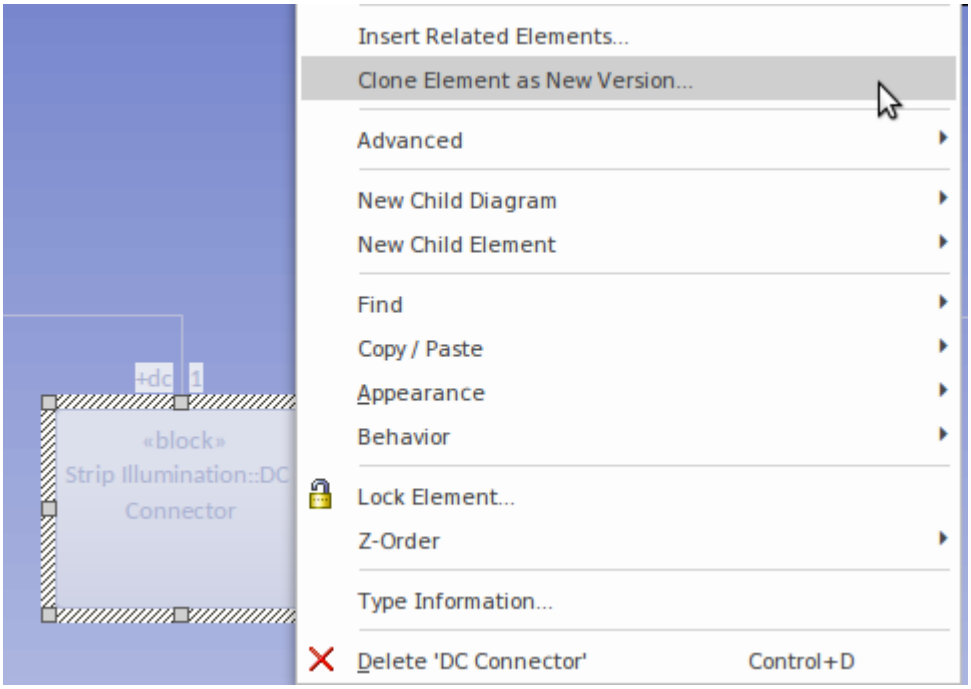
The step just performed simply sets up the structure for the cloning and does not itself create future states of element; it does, however, make a copy of any diagrams contained in the Package. This illustration shows a Package that has been cloned, containing two diagrams that are copies of the diagrams in the original Package.



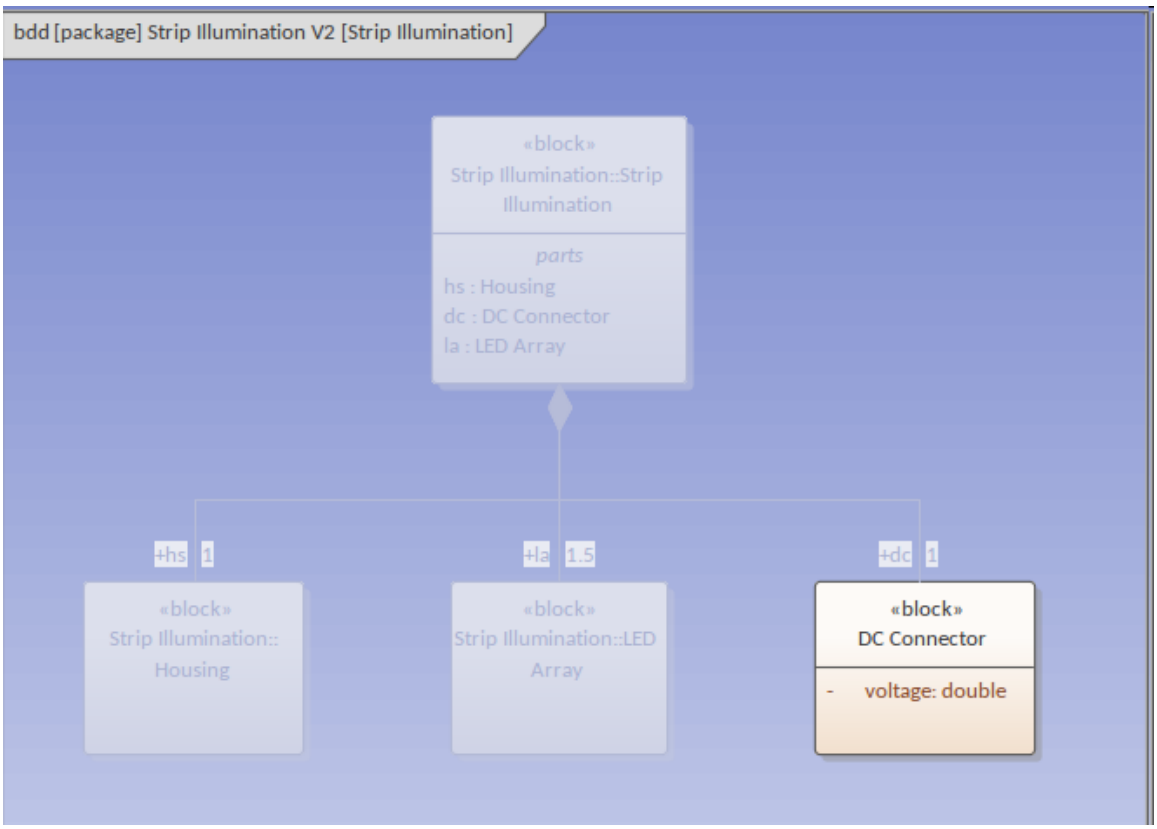
The diagram is assigned the version number specified in the version dialog, set by the user in the previous step. The cloning of individual elements is by selecting an element on a diagram, but until this is done all elements on the diagram will be the previous version. Enterprise Architect has a facility available from the diagram property sheet that allows the modeler to filter the elements on the diagram to the version of the diagram.



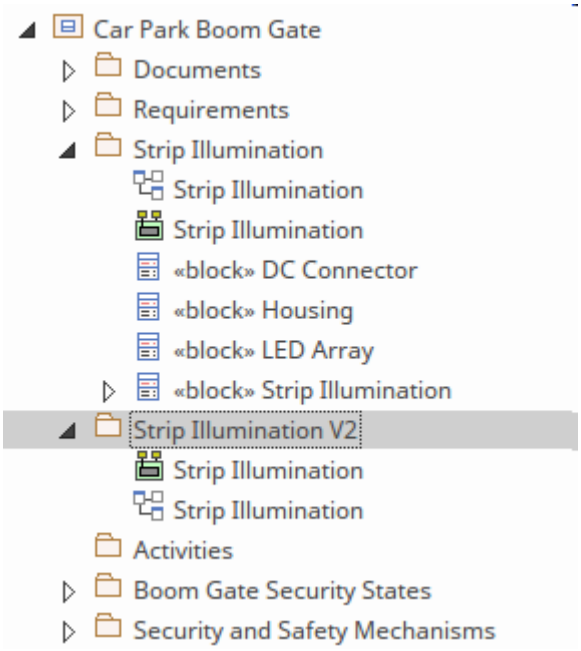
Setting this property on a newly cloned diagram will display all the elements in a gray-scale as they are all from the previous version. From this point, individual elements can be cloned and Enterprise Architect will make a copy of the selected element, creating copies of all its connectors. This is important as it will allow the element to be promoted as the updated current version once the change has been implemented. Individual diagram elements can be cloned by selecting the element in the diagram and displaying the context menu, as shown in this illustration.



Once again you will be prompted for the version number, conveniently the tool will default to the one chosen for the cloned Package. Once this has been accepted Enterprise Architect will create a copy of the selected element and because the diagram is still *filtered to version* this element will appear normally in the diagram, with the other elements still displayed in gray-scale.



The Browser will show the newly created (version 2) element that will be collated with the diagram. In this way the cloned Package will only contain elements and diagrams with the new version number. For more information see the Time Aware Modeling Help topic.


Car Park Boom Gate
Documents
Requirements
Strip Illumination
Strip Illumination
Strip Illumination
«block» DC Connector
«block» Housing
«block» LED Array
«block» Strip Illumination
Strip Illumination V2
Strip Illumination
Strip Illumination
Activities
Boom Gate Security States
Security and Safety Mechanisms

Version Control and Baselines

We live and work in a world which is moving at Internet speed and consequently engineering problems and opportunities change at the same speed. It is the engineers challenge to record, analyze, conjure and implement engineering solutions in this timescale. This means that almost before an engineer has described a problem fully it will have changed or the business or engineering context of the problem will have changed. Most times the new direction or the changes will be described and adopted, but other times an engineer will be required to back-track and return to a previous version of the problem, opportunity or solution. Enterprise Architect has sophisticated tools for performing this back-tracking.

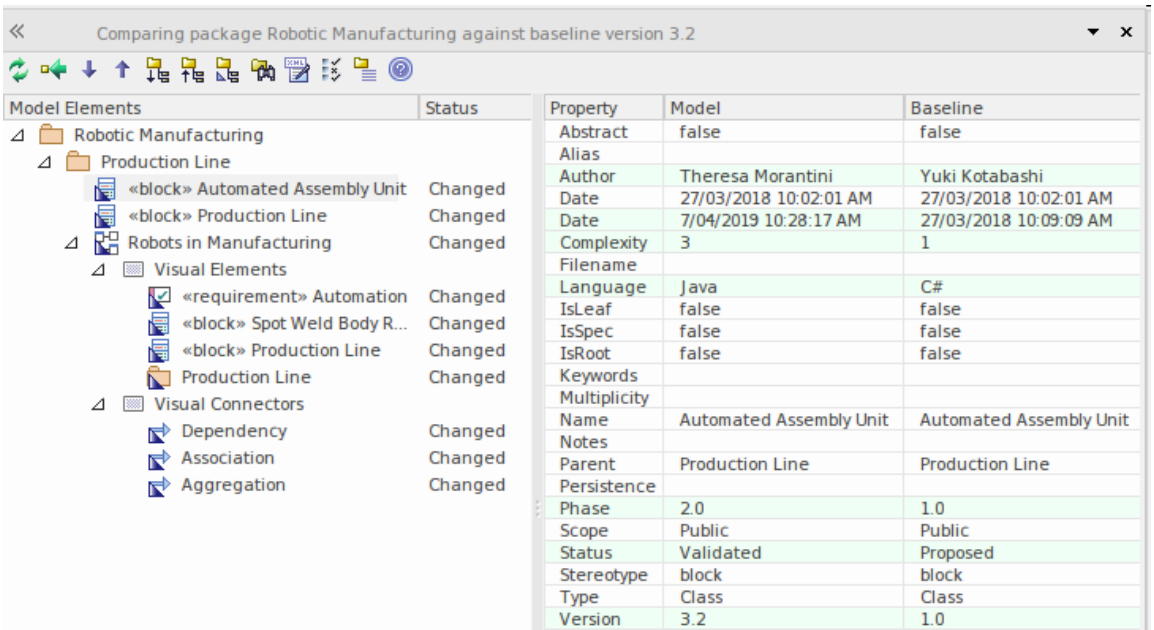
Enterprise Architect has two fundamental tools for working with prior versions of modeling content.

- *Version Control* - Once configured, any change point can be returned to; users check-out model fragments, make changes, and check-in the fragments, thus creating versions - for more information see the Version Control Help topic
- *Baselines* - Created at milestone or significant points in a model's development; a user creates a Baseline, and then at a future point the evolved model is compared to the Baseline at any level of granularity, and prior content can be restored - for more information, see the Baselines Help topic.

There is an important difference between the two methods, and that is: Baselines must be created intentionally. For example, if a team of engineers creates a model and it is signed off by a product owner, and then work continues onto phase two. If for some reason the team wants to back-track and return to the phase one model, without a Baseline in place this would not be possible. In contrast, once Version Control is configured, the same team could easily return to the phase one milestone so long as they knew the date it was completed.

Baselines

Baselines are an effective way to ensure that a team can back-track a model to a milestone or significant point in the model's evolution. They provide a user-driven way of managing change and give the modeling teams a sense of comfort that if they go off-track for some reason or some dimension of the problem, opportunity or solution changes and they need to return to a previous point it can easily be done. It is important to remember that a baseline is simply a snapshot of a Package (potentially including sub-Packages) in the repository but it must be created intentionally and needs to be created at the point in time that it represents. For more information see the Baseline Tool Help topic.



Version Control

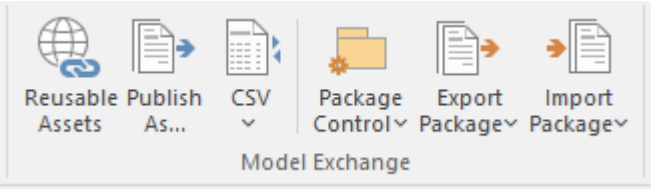
Enterprise Architect allows an engineering team to manage changes and revisions to projects by placing individual model Packages, view nodes or root nodes under Version Control. Version Control is configured within Enterprise Architect through any number of third-party source-code control applications that manage access to and store revisions of the controlled Packages. Once the Version Control software has been installed and configured a team can save a history of changes to Packages, view and retrieve prior revisions of work, check out and check in content as it is being worked on and more. This facility allows a team to work collaboratively while providing an isolated way for engineers to work on particular parts of the model. For more information see the Version Control Help topic.

Reusable Asset Server

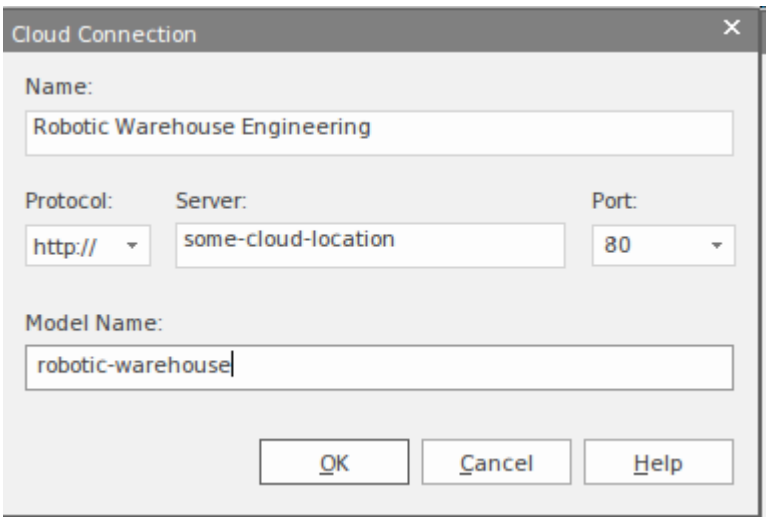
The Reusable Asset Server (RAS) is a team productivity feature that allows teams to store modeling content in a location that can be accessed by distributed groups of modelers for reuse. Any team or organization can set up a RAS, store content and - through security settings - make it available. The atomic unit of storage is an asset that can be both modeling- and file-based information:

- Packages contained in a repository and viewable in the Browser
- Files in a range of text, code and graphic formats, including .qea files

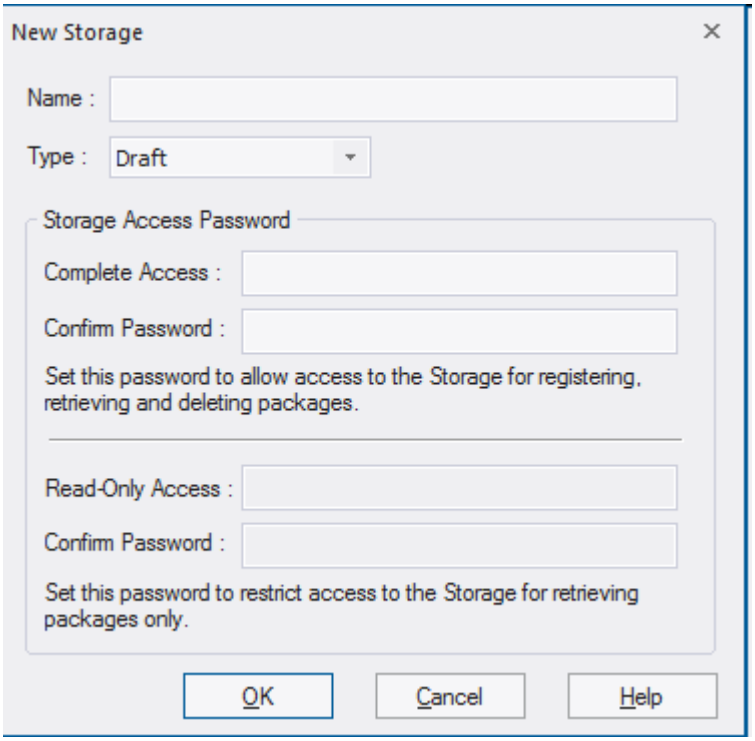
The Reusable Asset Server is available from the 'Publish' ribbon using the 'Model Exchange' panel. This puts this service at your fingertips.



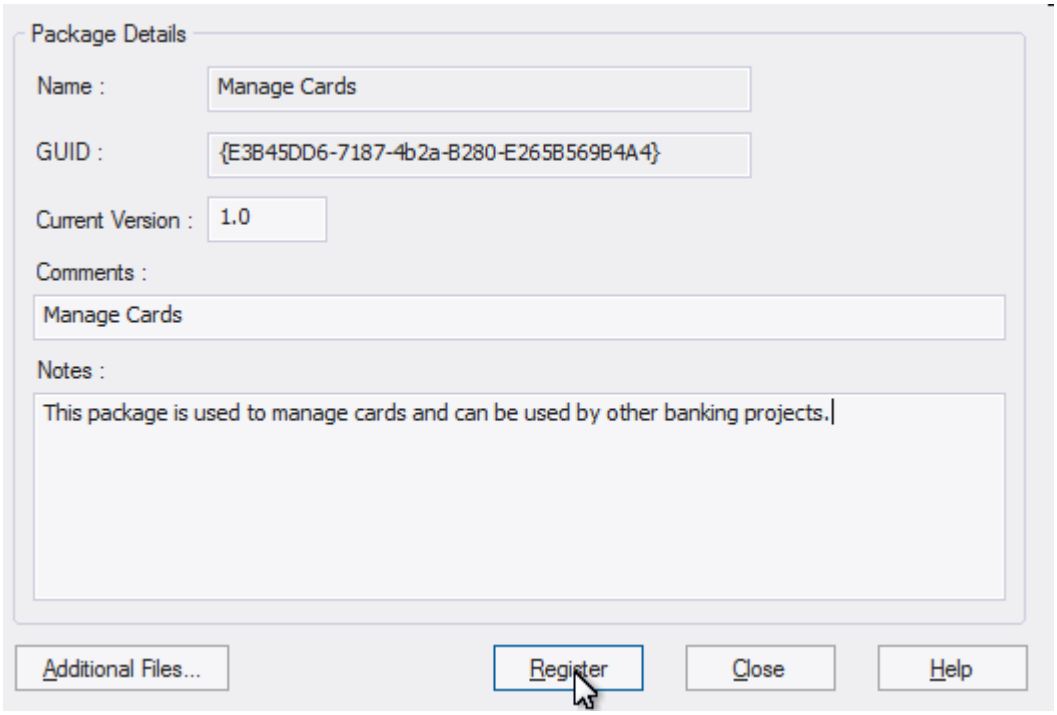
The assets are stored in the Cloud and require a connection to be specified to a Pro Cloud Service model that has been set up for this purpose. Typically, this task is performed by the infrastructure section of an IT department, and the details of how to connect would be simply provided to the engineering team. This screen capture shows the details that are required to make the Cloud-based connection.



Once a server has been set up it is possible to add any asset to it. Formally, the server is a registry and content is set to be registered on the server.

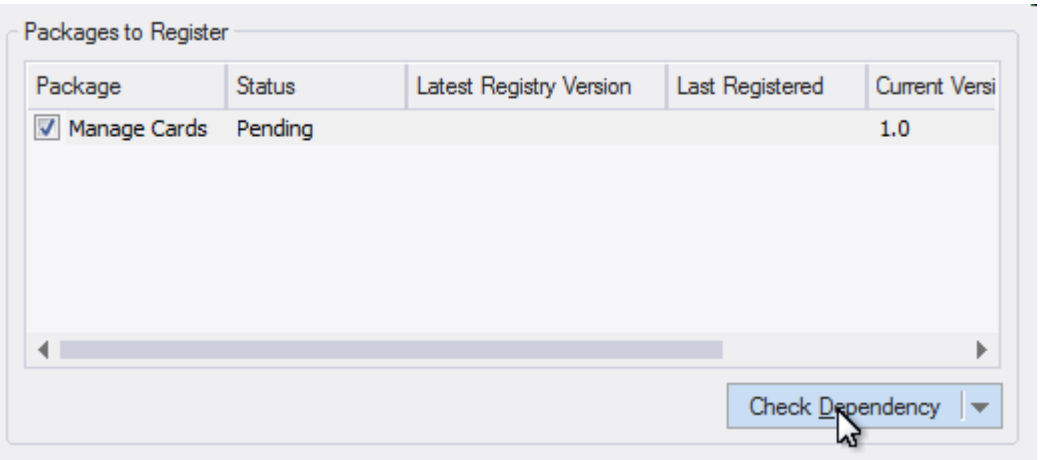


The Reusable Asset Server can be used to store information and modeling assets between projects, and is particularly useful for storing information for reuse between projects or programs. When a project has delivered its value to the business it is quite common for the project artifacts to be archived and effectively made inaccessible to other teams. The Reusable Asset Server is a convenient place to store these artifacts so they can be reused by other teams. For example, a project that has developed models for a new or upgraded hospital could store these valuable modeling artifacts in the RAS and then any time another hospital is being worked on they could be reused, saving potentially thousands of hours of work. This image shows the details of a Package registered within the RAS, including the version number, the Global User Identifier and comments that will help an engineer browsing for reusable content find the assets they are looking for.



One of the key advantages of using this feature is that the RAS holds assets in a dependency tree, allowing a potential user of the asset to understand the Packages that it depends on. This is an analogous mechanism used by software installation programs that determine if a software item selected for installation depends on other items that are not

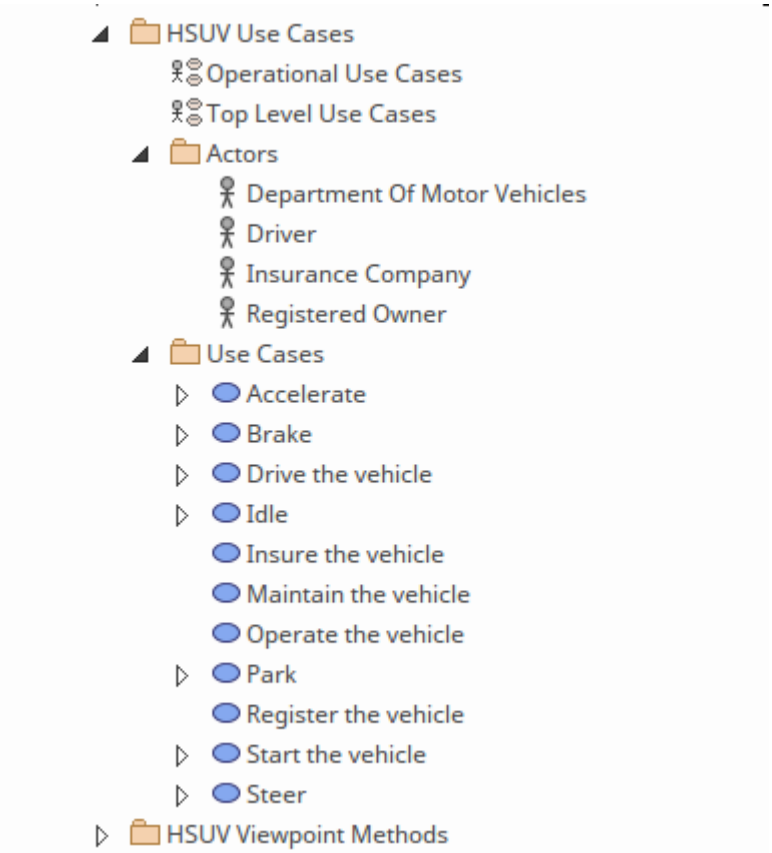
present on the target machine, and if these items in turn have other dependencies. The Reusable Asset Service does this work for the engineer and performs a traversal of the dependency graphs, allowing the user to understand what the required asset depends on. This screen capture shows how dependencies can be managed in the RAS.



For more information see the Reuseable Asset Service (RAS) Help topic.

Using Packages to Structure the Repository

The Package is one of the most fundamental and important elements in the SysML. It functions as a container and, viewed simply, it resembles a folder in your favorite file explorer software for your computer. So, in this way it is firstly a container that groups together other elements, including other Packages, but we will learn in this topic that it also has other important functions in Enterprise Architect.

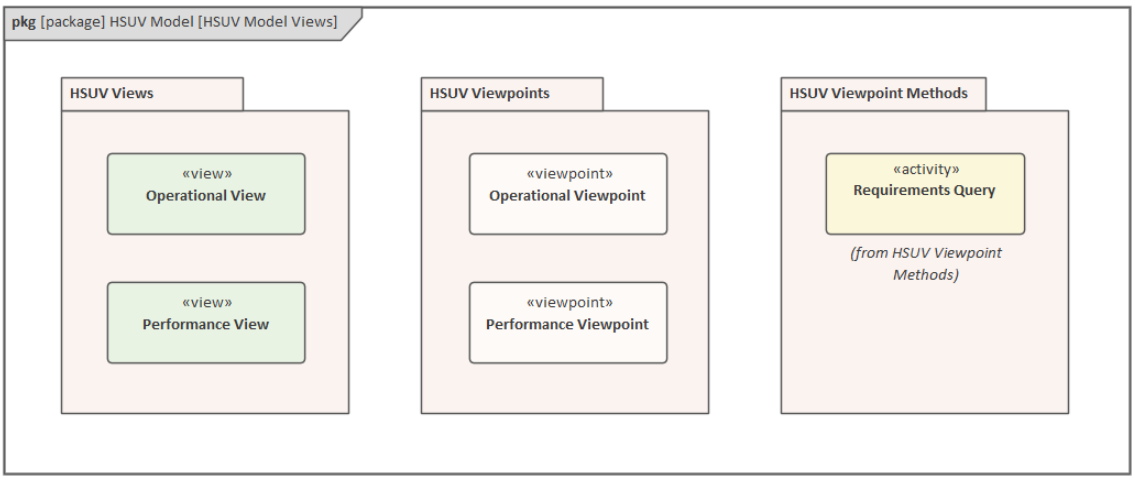


In a deeper sense a Package is a namespace that provides a way of uniquely identifying any element in a repository, similar to the way URL works. The path shown here has been extracted automatically from Enterprise Architect; it allows you to visualize the namespace.

SysML Example.HSUV Model.HSUV Use Cases.Use Cases.Drive the vehicle

Setting up the Package structure is an important and often dreaded task, but fortunately Enterprise Architect takes away a lot of the anguish that newcomers (and experienced modelers) feel when approaching this task.

The next few topics will introduce you to best practice in setting up a Package structure, and to some of the additional tools and facilities that will make working with Packages a lot easier.


pkg [package] HSUV Model [HSUV Model Views]
HSUV Views
«view»
Operational View
«view»
Performance View
HSUV Viewpoints
«viewpoint»
Operational Viewpoint
«viewpoint»
Performance Viewpoint
HSUV Viewpoint Methods
«activity»
Requirements Query
(from HSUV Viewpoint Methods)

The Function of Packages

In addition to the previous discussion Packages are an important element in the use of Enterprise Architect as they are used as the basis for a number of facilities in the tool, including:

- Container for elements,
- Namespace Definition,
- Security,
- Version Control,
- Baselines,
- Importing and Exporting,
- Documentation,
- Auditing,
- Time Aware Modeling, and much more.

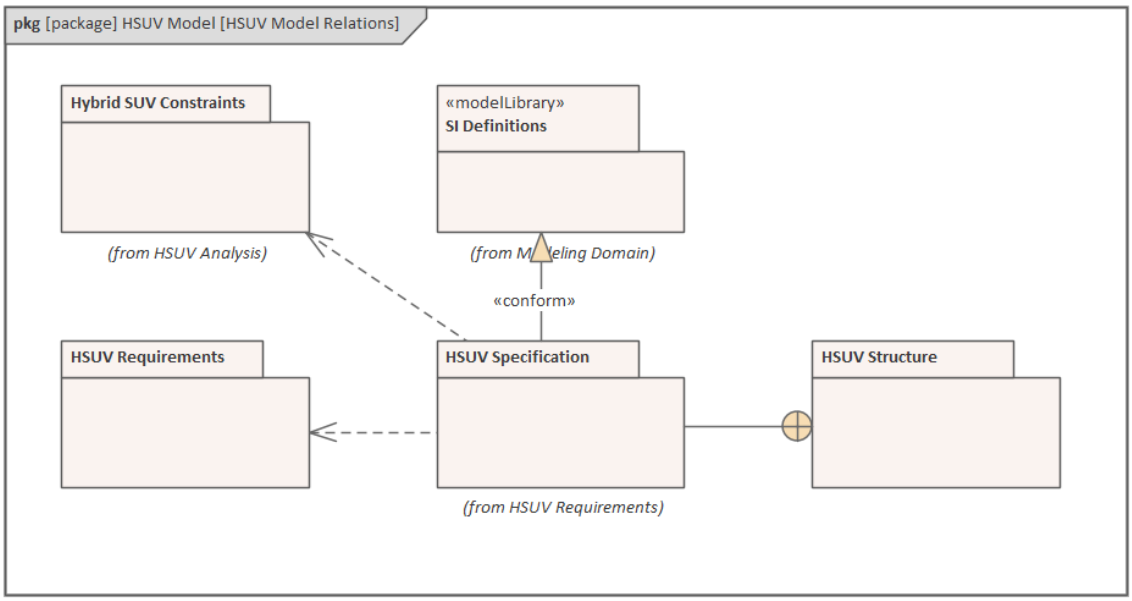
These things all have to be considered when deciding upon the structure of the Packages. Containership and namespace are the most critical, but all the other functions must be kept in mind when deciding on an initial model structure or when the model structure is being re-factored. It is often the case that some of the functions are not initially used and only brought into play when the repository has gained a degree of maturity. This is often a trigger for the repository to be reorganized, but fortunately - because of the ease of drag-and-drop - this can be done easily and effectively and is not a time consuming exercise.

Introducing Package Diagrams

The Package diagram is a simple diagram that visually describes the structure of the repository including relationships between Packages and other Packages and elements. Package diagrams appear quite simplistic with a small number of elements:

- Model
- Model Library
- Package
- View
- View Point
- Stakeholder

These are connected by a series of relationships.

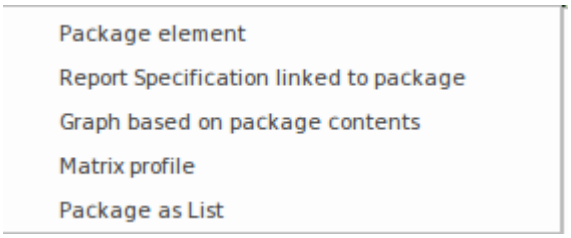


Again, the number of relationships is quite limited, but each has specific meaning in the diagram.

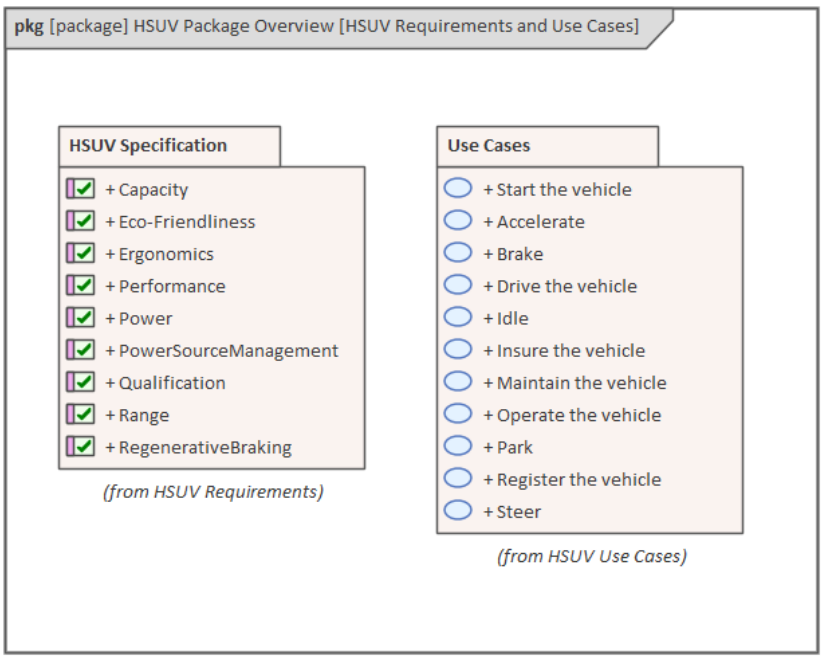
- Conform
- Dependency
- Import
- Containment
- Realization
- Refine
- Expose

As with all SysML elements, there is both a graphical and textual aspect to the elements, notes can be added to each of the Packages and the relationships to clarify the purpose of the element or the connector. The Package diagram can contain any type of model element but typically it contains Packages. Enterprise Architect extends the SysML specification by providing a number of different and innovative ways to visualize Packages and their content on a Package diagram. These options can be seen in the menu that is displayed when a Package is dragged from the Browser

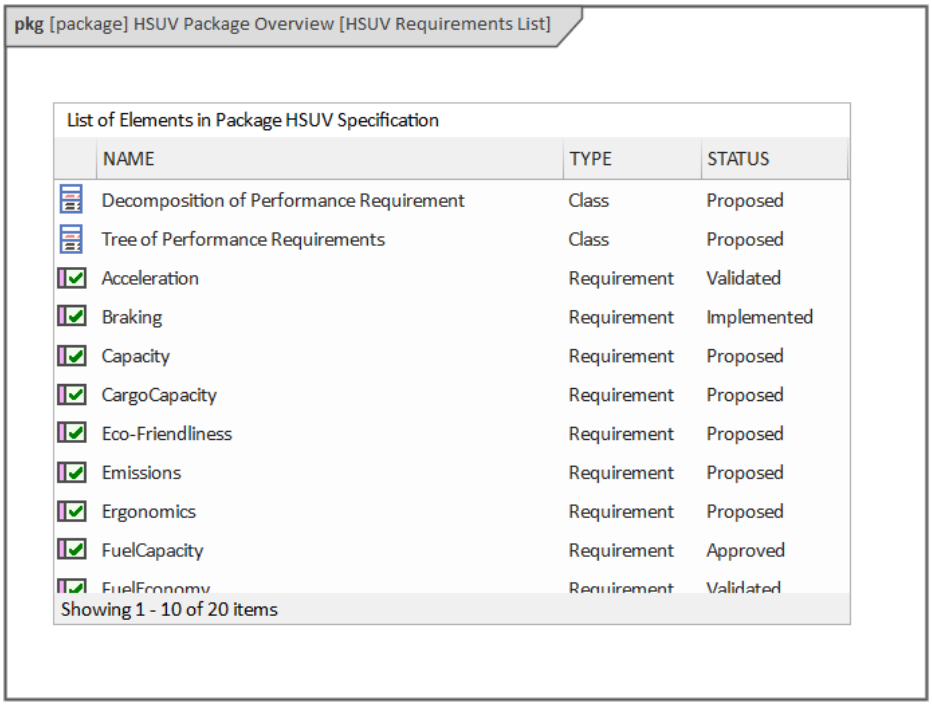
window onto a diagram.



In this diagram we can see that the modeler has chosen the 'Package Element' option and has set the Compartment Visibility of the Package diagram object to display the Package contents. The Compartment Visibility options are available from a diagram object's context menu for any element, and Enterprise Architect dynamically changes the options depending on the element type and the available compartments.



In the next diagram the engineer has chosen the 'Package as List' option, which relies upon auto-generated but user-configurable SQL code to create a dynamic list of elements based on any of the Package element's metadata. Here we see the same list of requirements but this time a number of properties are also displayed.



Any number of Package diagrams can be created to define or help visualize the structure of the repository. For more information see the Package Diagram Help topic.

Creating Package Diagrams

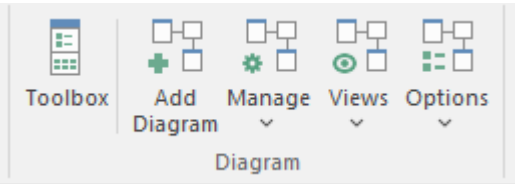
A Package diagram can be created from a number of places in the User Interface, by selecting:

- Design ribbon - *Add* icon on the *Diagram* panel
- Browser window toolbar - *New Diagram* icon
- Browser window *context menu* - *New Diagram*

The access options will all display the same dialog, they are simply different entry points to the same tool features. We will use the Design ribbon to create a Package diagram.

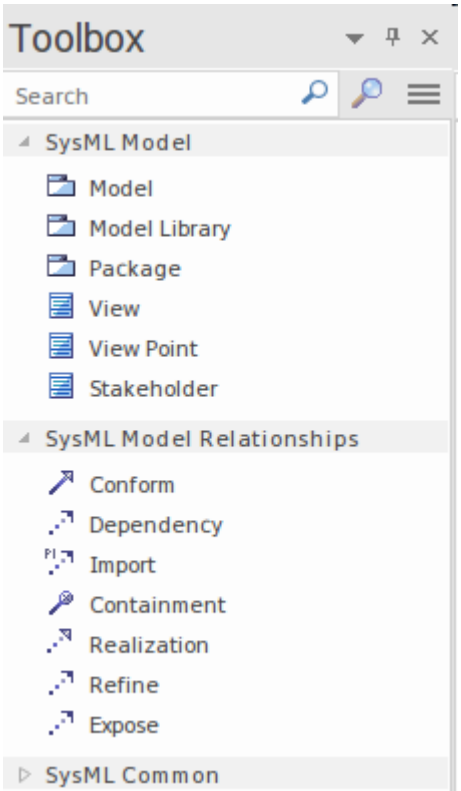
Firstly select the location in the Browser window where you want the Package diagram to be located. This can be either a Package or an element, but it is common to insert Package diagrams into a Package. Once the Package location has been selected in the Browser window, select the ribbon option:

Design > Diagram > Add Diagram



Selecting this option will open the *Diagram Builder* tab page in the *Model Builder* dialog, where you select the diagram type and specify a name for the diagram; the name initially defaults to the name of the Package or element that contains the diagram. With the SysML perspective chosen and the version of SysML selected, a list of diagram types will be displayed from which you select the Package diagram. Click on the *Create Diagram* button to create a new Package diagram in the location selected in the Browser window. The Diagram View will be opened, allowing you to start adding elements and connectors that describe the structure of the system and its division into these structural groups. Enterprise Architect will also display the 'Package' Toolbox pages that contain the elements and relationships as defined by the SysML specification to be applicable for constructing Package diagrams. Any number of other Toolbox pages can be opened, if required, in addition to the 'Common' elements and 'Common Relationships' pages that will always be

available.



The most import elements and connectors that are used with the Package diagram are:

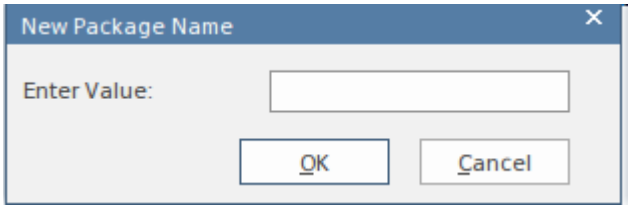
Elements

- Model - used to define a high level part of the system
- Model Library - used to define a reusable set of elements
- Package - used to create a basic structural unit
- View - used to define what a stakeholder will see when viewing
- Viewpoint - used to define a reference point for a view
- Stakeholder - used to describe people or parties with material interest

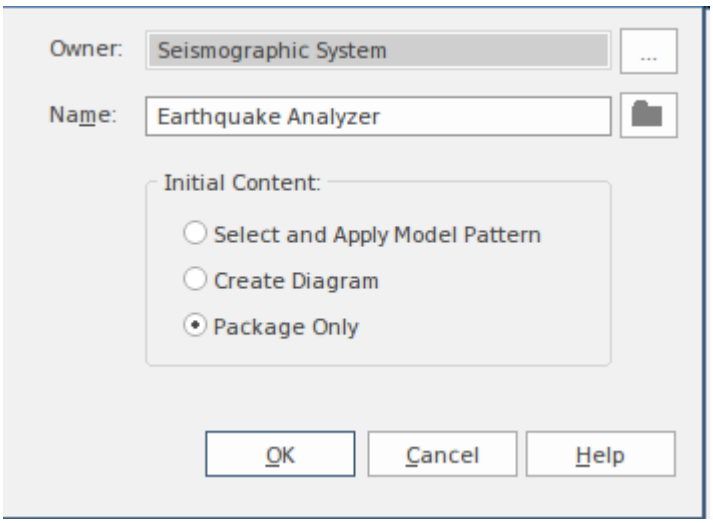
Connectors

- Conform
- Dependency
- Import
- Containment
- Realization
- Refine
- Expose

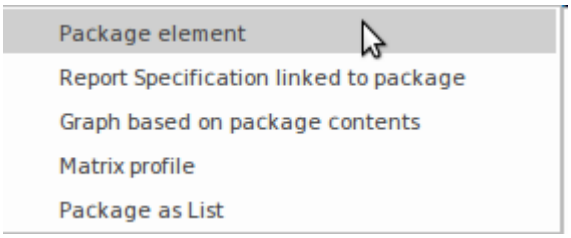
Elements can be added to the diagram by dragging-and-dropping them from the Toolbox onto the Diagram View. It is considered good practice to start by defining Model and Model Libraries. When a Model or Model Library icon is dragged from the Toolbox to a diagram, the modeler will be prompted to enter a new Package name using this dialog:



Once the Package name has been entered a new Package element will be created on the diagram. When creating a Package from the Toolbox or inserting one into the Browser window, a modeler is given a number of options as indicated on the dialog in this screen capture.



As described earlier, when an existing Package is dragged from the Browser window (or copied from an existing diagram) Enterprise Architect allows you to create it in a number of different ways, as indicated in this screen capture:



Package Organization Regimes

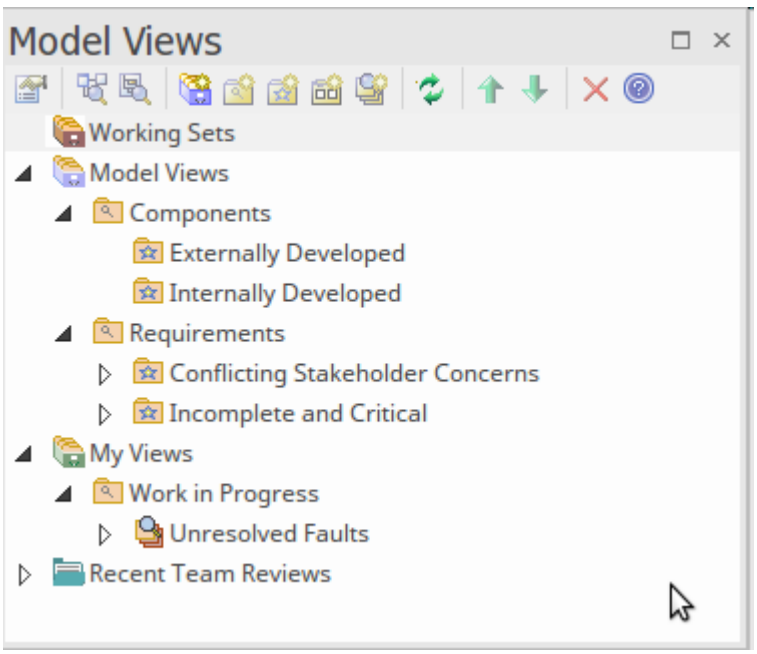
As suggested earlier, the librarians, administrators or engineers involved with the set up of the repository can find themselves conflicted about which direction to take because there is a wide range of organizing principles that can be used to structure the contents of the repository. Some of these are:

- A Breakdown structure - Systems | Subsystems | Components | Parts
- Engineering Teams, working on different aspects of a system - Team One | Team Two
- Programs of work and Projects - Program One | Project One, Project Two
- Divisions within a method - Architecture | Requirements | Design
- Security and Access Control
- Ease of Navigation

Any one or any combination of these principles can be used to structure the repository, and they can be changed over time to suit the evolution of the engineering practice and the model usage and experience of the users. Possibly the most difficult of these principles is the need to make the repository friendly to its inhabitants, to ensure ease of navigation so that they can easily find what they are looking for. Enterprise Architect has some useful facilities to reduce this tension, allowing other mechanisms to be used for navigation and freeing the repository design to develop based on a number of the other more important principles. Some of these tool features are listed here.

Model Views

Provide a flexible and effective mechanism allowing an engineer or team to create any view of the model that they find useful. Using this facility removes the need for modelers and engineers to access the Browser window, as they can locate the elements of interest through the Model Views window.

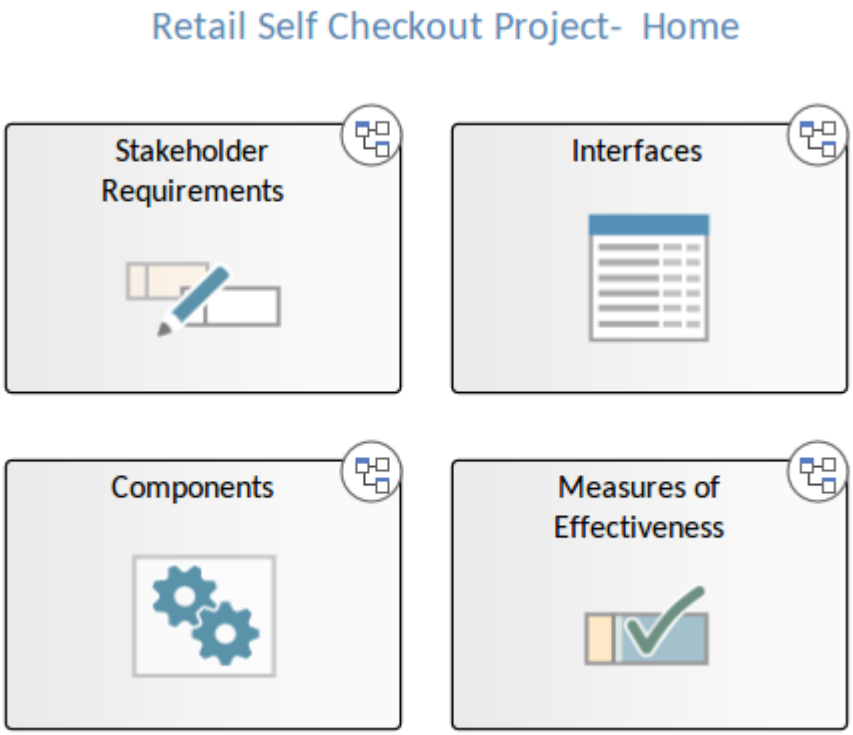


For example, views can be created based on a search that returns elements from any part of the repository; an engineer could define a view that returned all requirements that were high priority, with the status 'Approved' and flagged as 'Difficult', regardless of which project they were part of or where in the Package hierarchy they were located. Alternatively, a modeler might just cherry-pick particular elements and diagrams of importance to them and include them in a Favorites View, or create a view based on newly created Components. This facility provides a highly flexible mechanism for accessing the important parts of the repository, and views can be created at modeler or team level. We

will return to the Model Views facility in a later topic, as it is an extremely useful part of the tool.

Diagram Navigation Cells

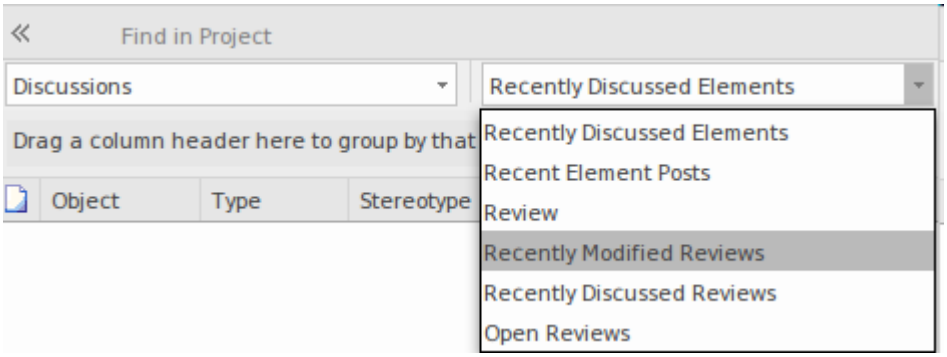
Enterprise Architect has made it easy for users to navigate through a repository by providing a diagram mechanism to hyperlink to any diagram in the repository.



This allows librarians and even modelers themselves to create any number of diagrams that act as launch pads that will take a viewer to diagrams of interest, effectively shielding them from needing to know how the repository is structured. These diagrams are viewable through the Internet browser and Cloud products, and provide a compelling experience for casual users and non-modelers.

Search Facility

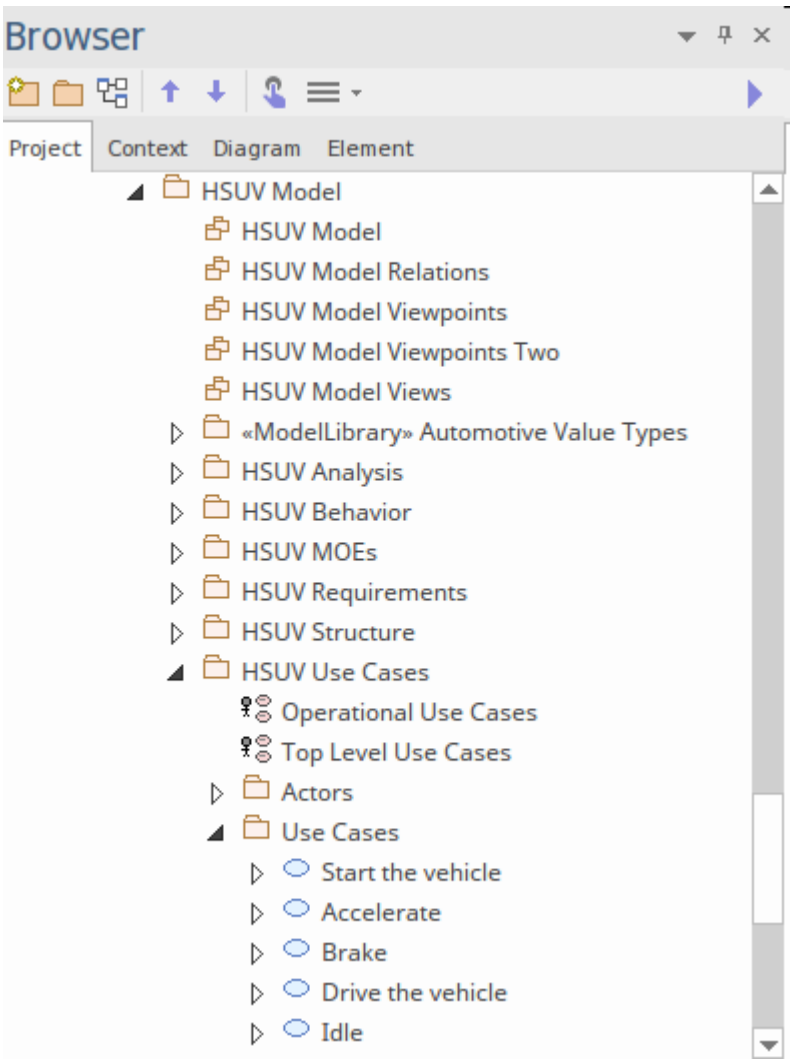
This is a power feature that provides built-in and user defined searches to retrieve a list of elements or diagrams that meet a specified set of criteria. The amount of information contained in a repository can grow exponentially as more people contribute to the models and information is imported from external sources such as Risks, Policies, Rules, Principles and more. There is a rich and useful set of searches that are defined as part of the product and in many circumstances one of these built-in searches will suffice for a modeler or engineer to locate the elements or diagrams they are looking for. These searches can be parameter driven providing a mechanism to reuse a search to find a variety of elements. For example a search could be written that has a user input parameter of Status allowing users to input a status, for example 'Proposed' at the time the search is run.



Searches can be created by non-technical staff using the intuitive Query Builder but there are also a number of other ways that searches can be created including SQL based queries that do require knowledge of the database tables and Add-in queries that require a technical person to create a program that defines the search. These searches can be used by a number of other facilities including, as discussed earlier, Model Views.

The Browser Window

The Browser window is the primary tool for structuring and navigating the repository, using expanding and collapsing tree nodes. The key structural element is the Package, which is a folder-like element that can contain other elements and diagrams, including other Packages. The elements in turn can contain other elements, features and diagrams, but not Packages.



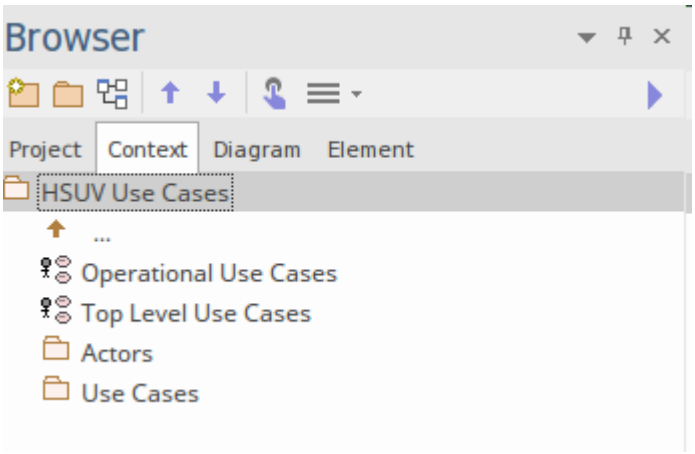
Root nodes are the highest nodes in the tree; these root Packages can contain Views that in turn can contain any level of Packages and elements. Tree nodes including Packages, elements, Features and diagrams can be copied and pasted between locations, or dragged and dropped to new locations. Many important tools, functions and windows are applied at the level of the Package, such as import or export of model content, documentation and Package Control, including Baselines. For more information see the The Browser Window topic.

Context Diagram and Element Browsers

Enterprise Architect provides a number of additional browsers that help an engineer or modeler to focus on a subset of the repository content. These browsers can be selected as tabs from the main Browser window.

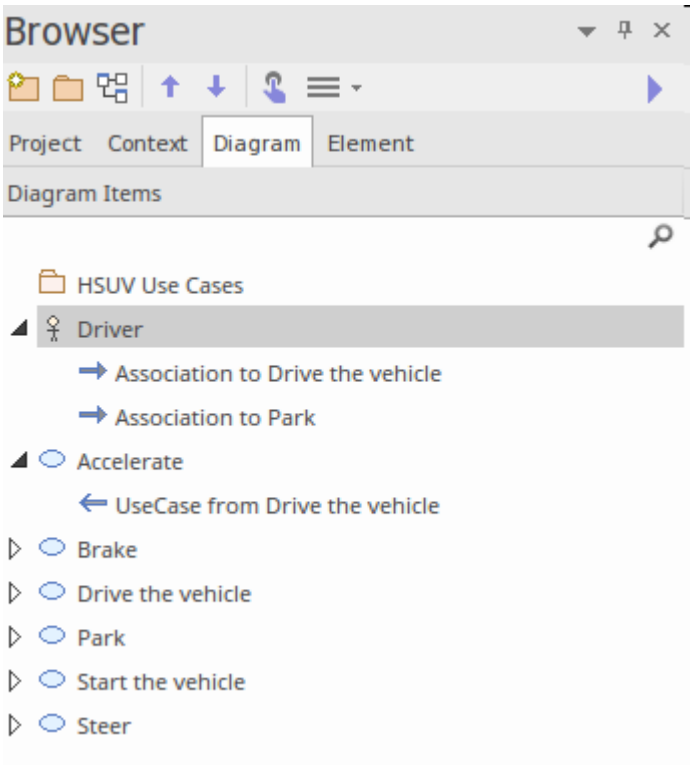
The *Context Browser* provides a filtered view just of a particular branch of the model, to work on just the section of the repository that is relevant at a particular time. This spotlight view takes away the noise and complexity of the Project

view and shows just the part of the model of interest, allowing the engineer to view it in isolation.

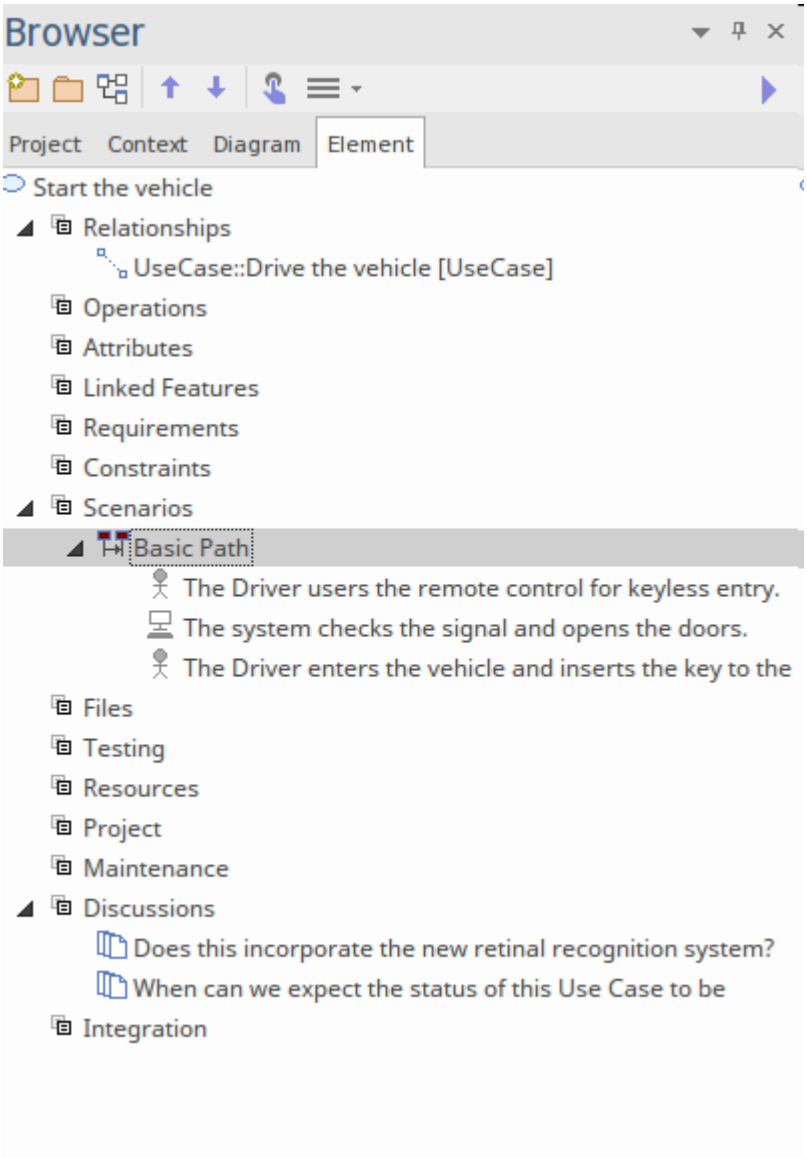


In our example, the modeler wants to focus his attention on the Use Case Package. Using the ... button you can move back up the tree, or by clicking on a Package or element that contains other elements these can be displayed, but the view always remains at a single level of hierarchy.

The *Diagram Browser* lists the objects present on the active diagram. Each object's connectors can also be displayed, making this a valuable view of the diagram.

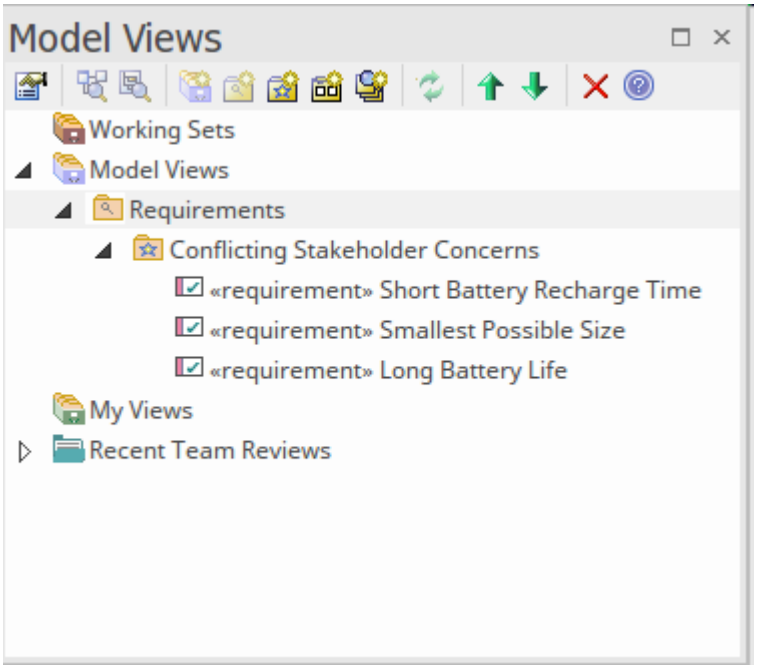


The *Element Browser* displays information about the currently selected element. This provides a way of visualizing all of the elements relationships, scenarios, requirement, features and much more.


Browser
Project
Context
Diagram
Element
Start the vehicle
Relationships
UseCase::Drive the vehicle [UseCase]
Operations
Attributes
Linked Features
Requirements
Constraints
Scenarios
Basic Path
The Driver users the remote control for keyless entry.
The system checks the signal and opens the doors.
The Driver enters the vehicle and inserts the key to the
Files
Testing
Resources
Project
Maintenance
Discussions
Does this incorporate the new retinal recognition system?
When can we expect the status of this Use Case to be
Integration

Accessing the Repository using Model Views

Model Views provide alternative views of the elements in the Repository. Whereas the Browser window is designed to organize the Packages and elements structurally, the Model Views facility allows the modeler to create a number of views that can group elements and diagrams differently. This is a handy facility that can be used by individuals and engineering teams to see the repository contents in any number of proprietary views designed to present only the information that is important or relevant. It effectively allows you to create windows through which to view the repository in unique and compelling ways that will provide insight and clarity, allowing the modeler to see things that might not have been possible using the Browser window.



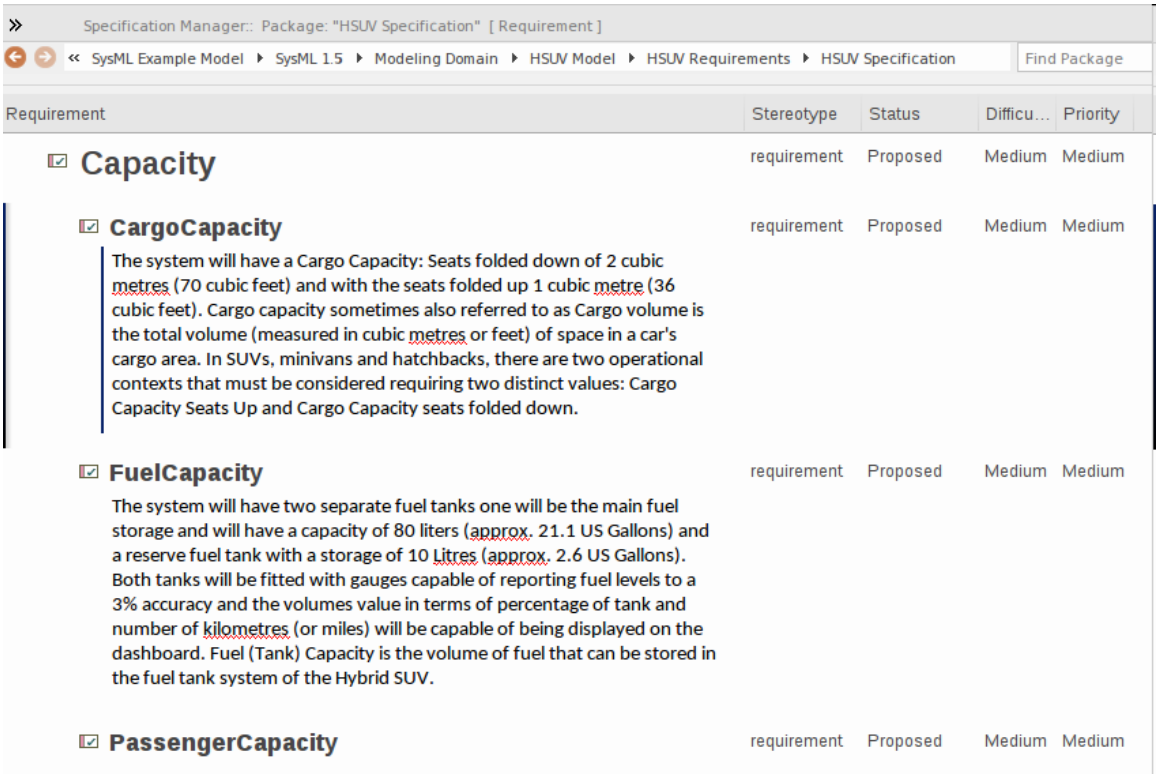
The views can be based on a wide range of criteria, including Favorites folders containing hand picked items of interest, and folders based on a search such as 'all elements created last week that have a status of Proposed' or 'all Components provided by a particular engineering supplier'. For more information see the Model Views Help topic.

Requirement Definition and Management

The field of Requirement Engineering is one of the most critical disciplines in the solution development lifecycle, and it has a documented impact on the success of projects. In the words of the renowned Twentieth Century physicist, Albert Einstein:

'If you define a problem carefully enough the solution will jump out at you.'

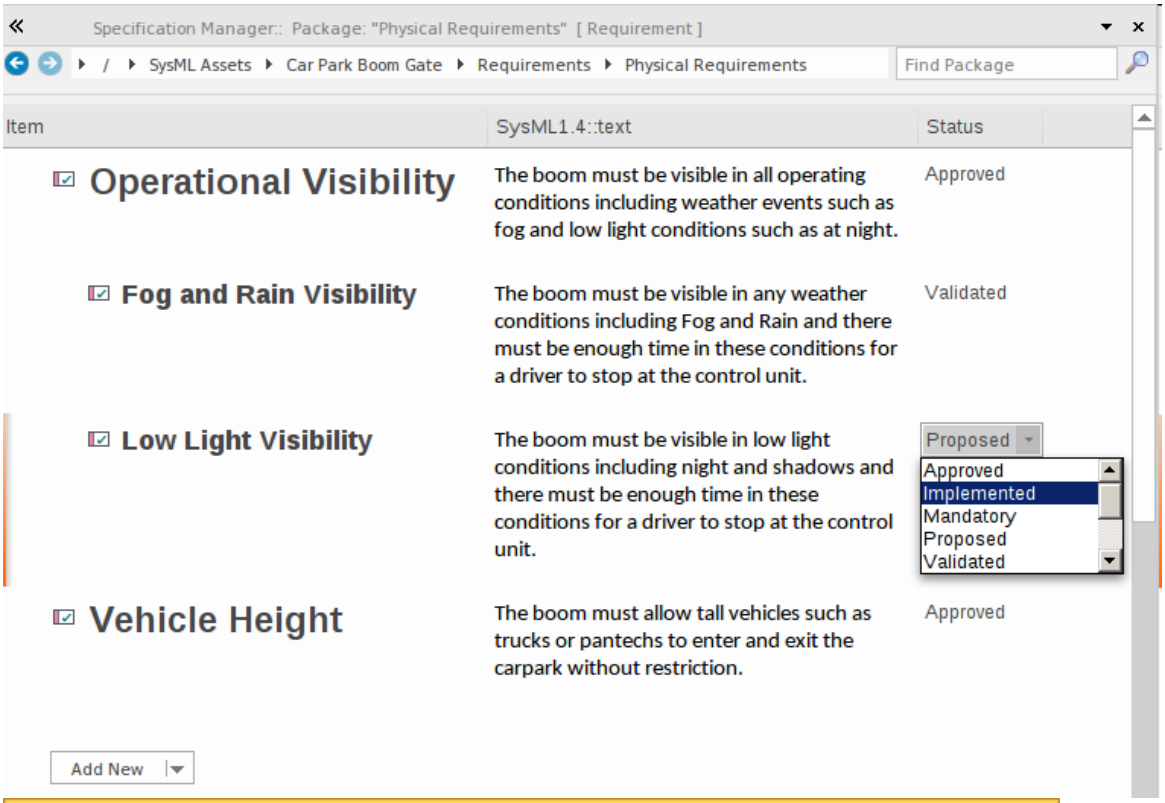
Enterprise Architect has an unparalleled range of tools for developing, managing, visualizing and documenting requirements, including tools to import or integrate and synchronize with external requirement management systems.

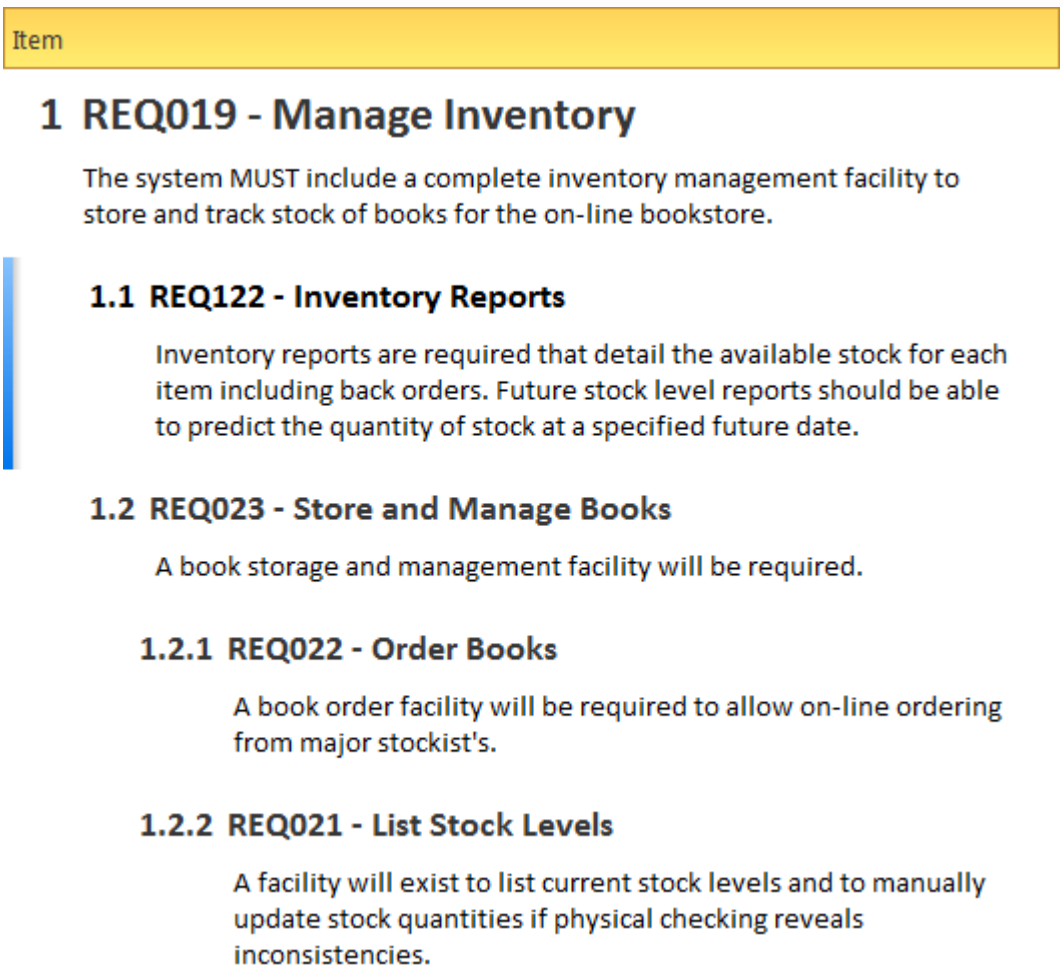


These tools implement all aspects of requirements that are defined in the SysML specification, but the tool features go far beyond this to create a sophisticated requirements platform replete with tools for all disciplines associated with the management and definition of requirements. Not only are the tools useful for those engineers or managers working directly with requirements, but there is a range of facilities such as the Traceability window that will assist any discipline, and that can be used by the Architecture and Design Teams who are responsible for ensuring the requirements are built into the designs, and consequently implemented into the delivered product or service. For more information see the The Requirements Model Help topic.

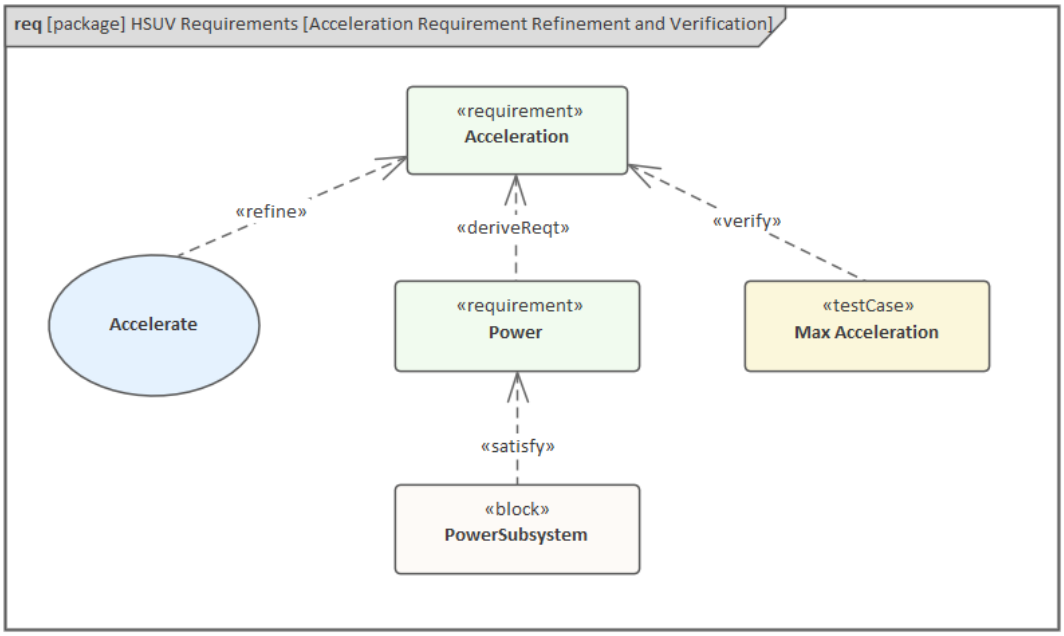
Developing Requirements

Requirement Development consists of all the activities and tasks associated with discovering, evaluating, recording, documenting and validating the requirements for a particular project. Requirements are discovered, analyzed, specified and verified. Enterprise Architect has a wide range of tools and features to assist the Systems Engineer as they develop requirements. The centerpiece for Requirement Development is the Specification Manager, through which the Engineer can enter, view and manage requirements in textual form as if in a spreadsheet or document. Requirement properties such as Status, Priority and Author can be edited in-line, and filters can be applied to restrict the display to particular requirements.



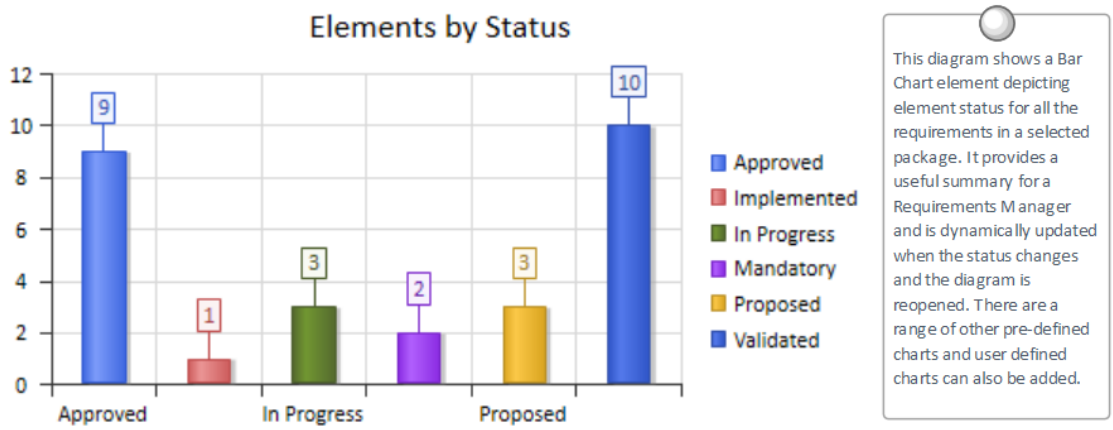


The Specification Manager can be used in conjunction with a platform of other tools such as diagrams, the Traceability window and the Discussions facility.



Managing Requirements

This consists of the activities to maintain a set of requirements that represent an accord or agreement between the project team and the customer. It also has a focus on ensuring that the requirements are acceptable to the Design and Development teams, and that they are sufficiently specific to be implemented into working business, software or hardware systems. Enterprise Architect is a sophisticated platform for managing requirements, and regardless of the domain, the size of the project or the method being followed, Enterprise Architect provides tools that make it easy to manage the largest of requirement repositories in complex projects.



Requirement Relationships

There is a rich set of requirement relationships that allow the Requirement elements to be connected to other modeling elements including other Requirements. The relationships include:

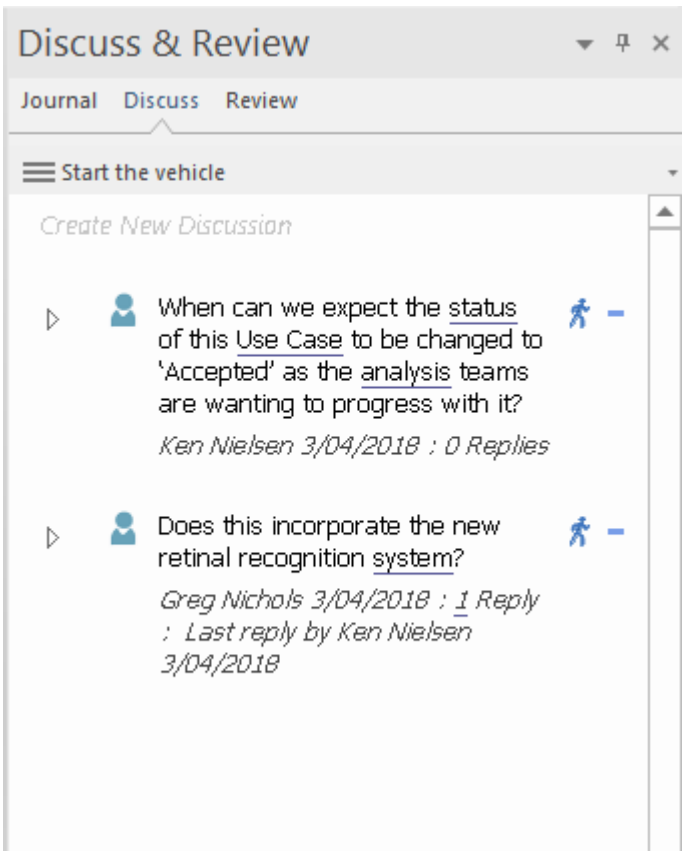
- Containment

- Trace
- Copy
- Derive
- Verify
- Refine
- Satisfy

We will explore these relationships fully in the body of this topic.

Visualizing Requirements

The visualization of requirements is an important aspect of the requirements process as it is critical that the catalogue of requirements can be viewed by all stakeholders as they are specified, analyzed, developed and managed. The requirements represent an engineer's interpretation of the discussions, observations and articulations made by stakeholders concerning the problem or opportunity at hand. Enterprise Architect has a wide range of mechanisms not only to present these requirements to the stakeholder community but also to allow the requirements to be discussed, reviewed and curated.



Documenting Requirements

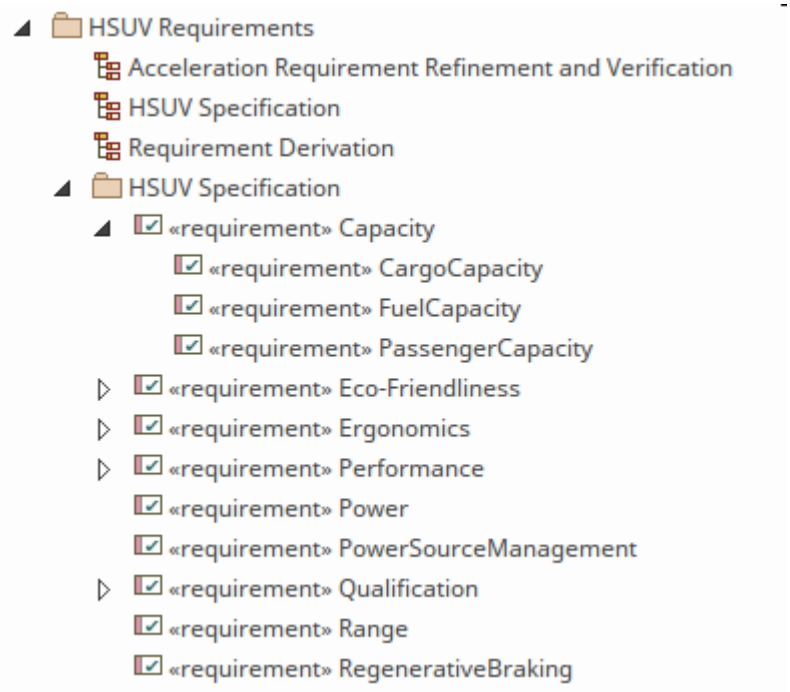
There are a number of documents that are commonly produced as part of the requirements engineering discipline such as the System Requirements Specification and Use Case Reports and these can be generated automatically from a requirements model using built-in templates or user defined templates. In addition a wide range of other documents can

be produced using built-in or customized templates.

There is also the possibility of viewing the models in a web browser on a portable device such as a phone or tablet or a PC. This facility is available as part of the Pro Cloud Server product and provides an alternative to producing static documentation and allows an engineering team to communicate and collaborate with an extended audience outside the modeling environment without the need for any software installation or configuration.

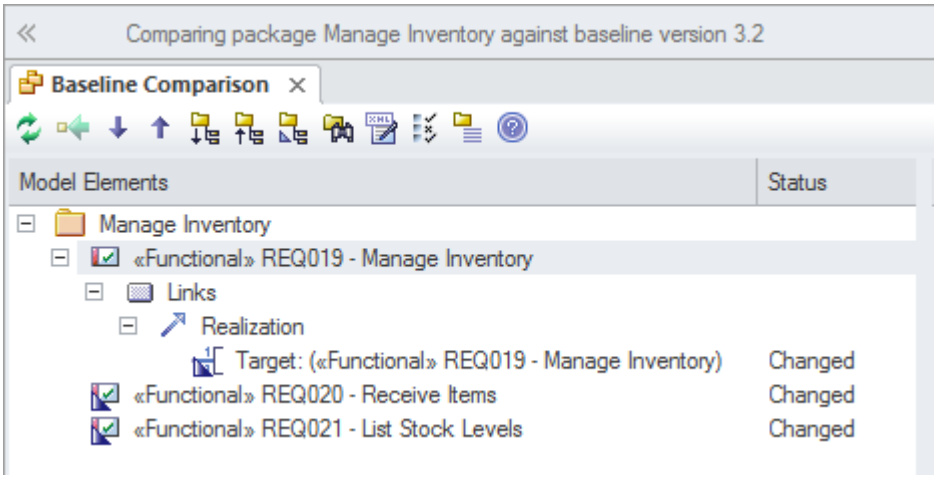
Requirements as First Class Citizens

Enterprise Architect provides a wide range of facilities that can be used for the development, visualization, management and documentation of Requirements as first class elements. People reading general SysML textbooks will often come away with the idea that Requirements are expressed on diagrams, but Enterprise Architect provides a wide range of other ways to visualize Requirements that will assist the engineer when working with them as text based elements, including being able to visualize them in a hierarchy in the Browser window.



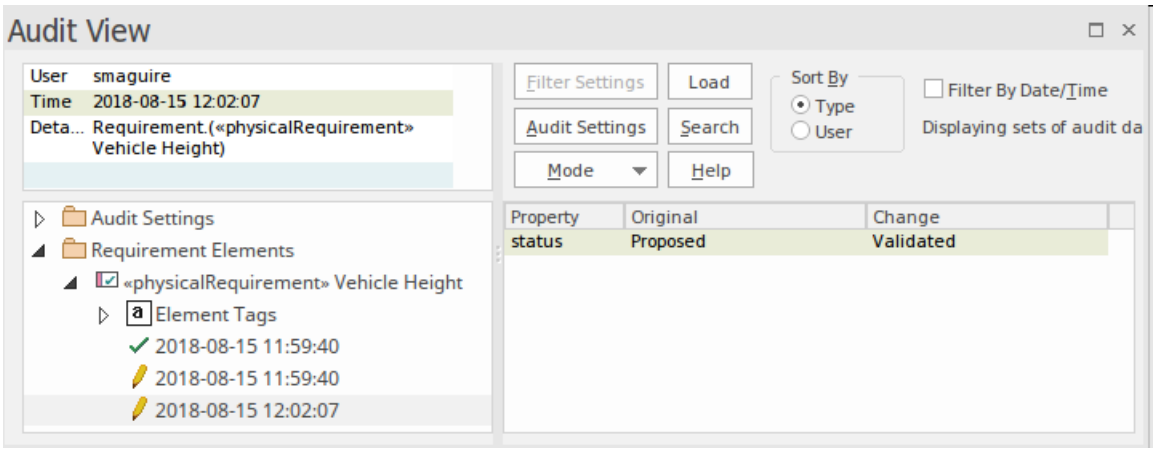
Requirements can be created as part of a specification or tender, or form part of a contractual document, in which case they can be easily imported into Enterprise Architect. However, it is more common for them to be developed as part of an elicitation effort typically conducted with workshops and reviews. Enterprise Architect has a number of features that can be used to record the proceeds of these meetings, such as Mind Map diagrams. Once the workshops have been completed, the ideas recorded in these meetings can be converted to Requirements or mapped to the meeting elements in a way that allows them to be developed collaboratively.

The Requirements often form part of a contractual relationship between organizations, or an agreement between different sections of the same organization, and as such need to be maintained and managed with rigor. Enterprise Architect provides a wide range of facilities to assist with this rigor, including Baselines, Audit tools, Version Control and more.



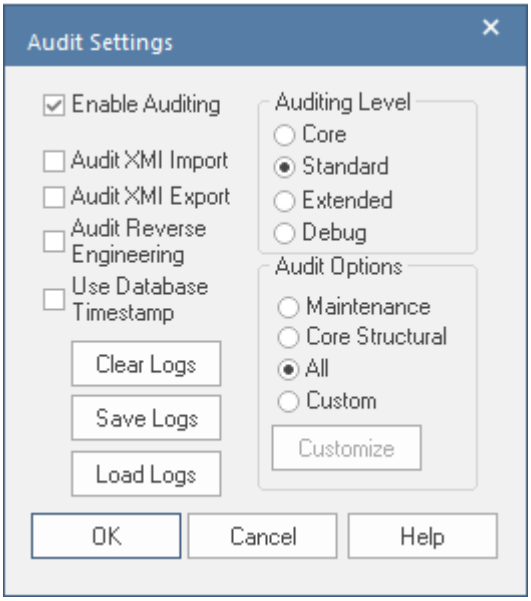
Requirements Auditing

Auditing can be turned on in a model and can track the details of a requirement change, including when it was changed, who changed it and the delta - before and after the change. Auditing can be used to track what was changed in a model, who changed it and when. There are a number of modes and a repository administrator can use the settings to specify what is recorded in the audit. While a Baseline can be used to show the difference between a model and a snapshot at a point in time, the Auditing tool records each individual change; it cannot, however, be used to revert to a previous state (the Baseline tool would be used if that was required).



This is a particularly useful feature in Systems Engineering, where there are regulatory or compliance aspects to a process or when faults have to be traced back to their design or requirement specification. Auditing would typically be set up and administered by a librarian or administration function within the team. Auditing can be enabled, set up and viewed using the ribbon option 'Settings > Model > Auditing'.

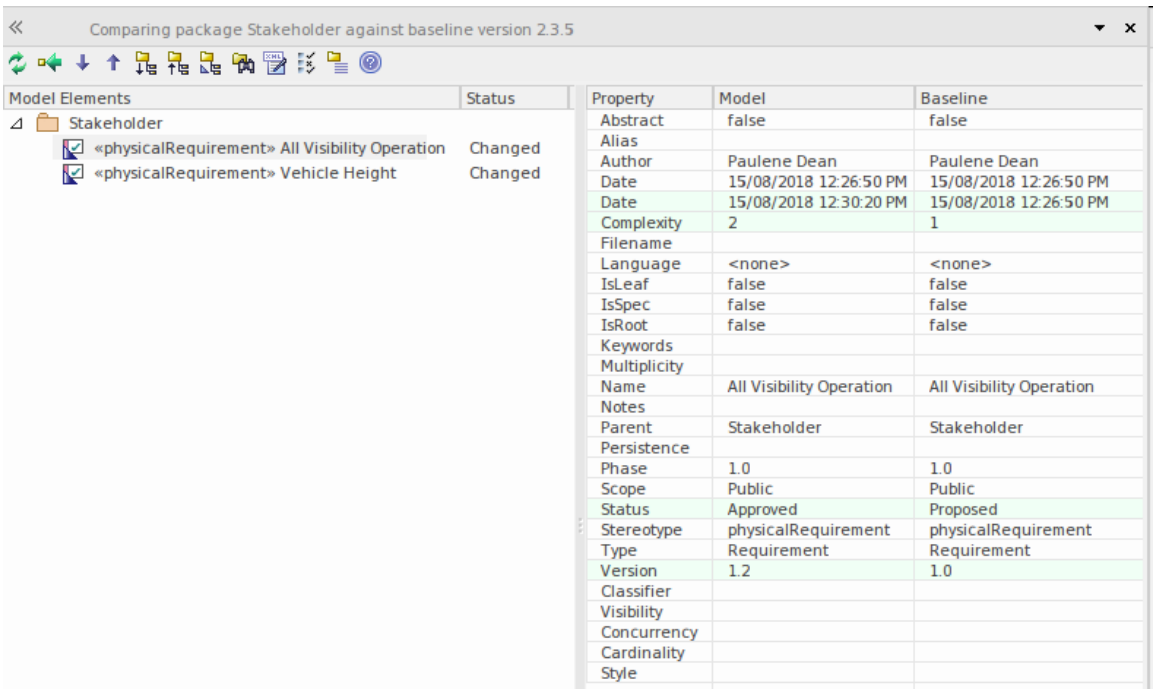
Auditing is by default disabled and must be enabled (turned on) before the system will start to keep an audit log. This, along with a range of other options, is available from the Audit Settings window.



For more information see the Auditing Help topic.

Requirements Baselines

Baselines are user initiated snapshots of a Package in a model. The Baseline effectively makes a copy of a branch of the Package hierarchy and its contents. At a subsequent point in time, the model can be compared with the Baseline and, if the model has changed, these changes will be presented in a visualization tool, allowing a user to view each part of the model that has changed, including the content that exists in the Baseline and the model. It is then possible to inject the contents from the Baseline into the model at the level of a discrete change.



Baselines provide a convenient way for Systems Engineering teams to ensure that models are evolving the right way, and when a model direction needs to be reverted to a previous version they can be used to reinstate atomic parts of the model. Baselines can be set up and viewed by pressing Ctrl+Alt+B or from the ribbon location:

Ribbon: Design > Package > Manage > Manage Baselines

As discussed previously, Baselines are user initiated and are stored inside the repository, so if the repository is copied the Baselines would be copied as well. It is quite common for most users to be given permission to create Baselines, but the ability to restore from a Baseline is typically reserved for a librarian or administration role. For more information see the Baseline Tool Help topic.

Version Control

Version Control allows Packages within a model to be versioned. To commence work on a part of the model, a user is required to check out a Package (including its sub-Packages) and then to work on a local copy. When the work is complete or at any point a user can check in the Package allowing the changes to be seen by other model users.

Version Control provides a sophisticated and robust way of working with models, and in contrast to Baselines does not require a user to initiate a version other than to check-out the Package. The system is automatically creating a version in the background as the work is done and changes are being made. Version Control can be set up and viewed from these ribbon options.

Settings > Version Control > Project-VC, Package-VC

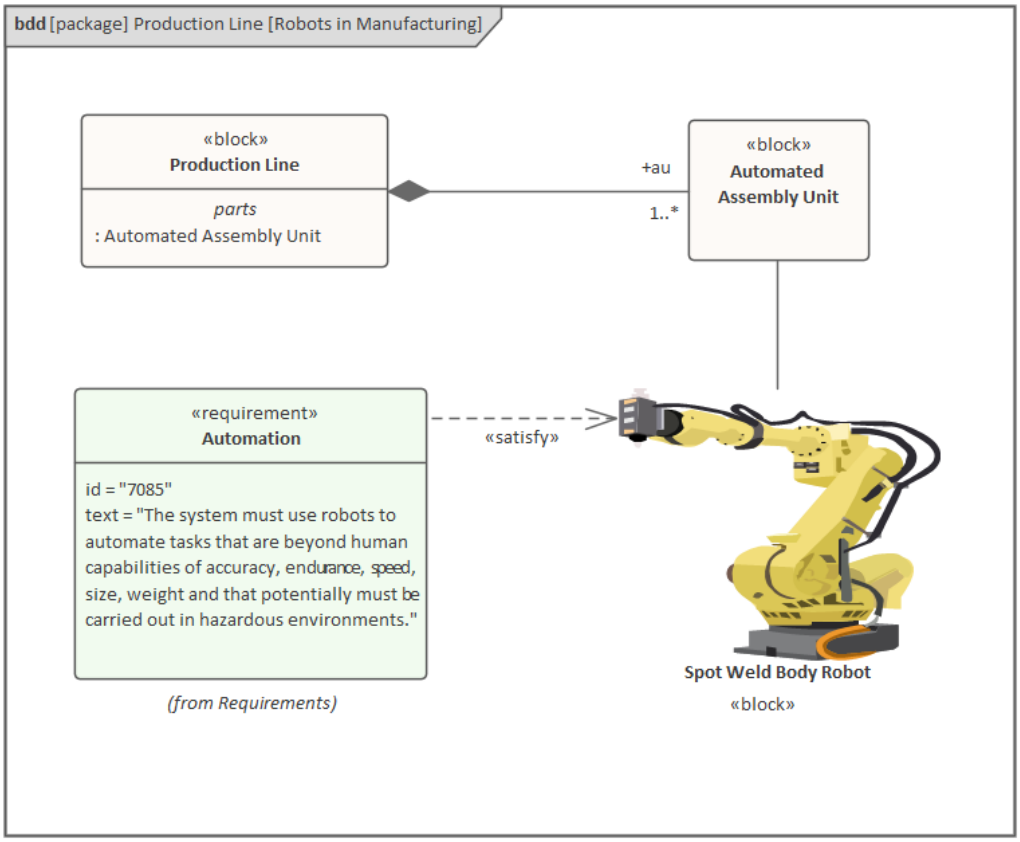
Version Control provides an effective mechanism for managing model content and allows a user or a team to keep fine-grained control over the way a Package and its content change over time. For more information see the Version Control of Model Data Help topic.

Introducing Requirement Diagrams

A Requirement diagram provides a way of visualizing Requirements and their connections. It is not just the relationship between any two Requirements, but the relationships between Requirements and other types of element such as Use Case, Activity and Block that can be visualized on these diagrams. Two of the elements provided in the Toolbox are:

- Requirement
- Test Case

These elements can be connected to each other, or to other elements, creating rich expressions.



In this diagram we see a Requirement that has been connected to a Block using a *Satisfy* relationship, which describes how other elements will ensure that the Requirement's intent is met. The Block has an alternative image defined, this being the image of a robot.

Again, the number of relationships is quite limited, but each has specific meaning in the diagram.

- Containment
- Trace
- Copy
- Derive
- Verify
- Refine
- Satisfy

As with all SysML elements, there is both a graphical and a textual aspect to the elements. The Requirement has two properties defined:

- id - a unique identifier for the Requirement

- Text - a textual description of the Requirement

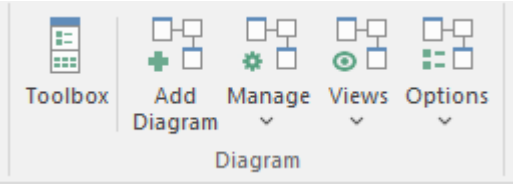
Any number of Requirement diagrams can be created to describe the needs and concerns of stakeholders and others. For more information see the SysML Requirements Modeling topic.

Creating Requirement Diagrams

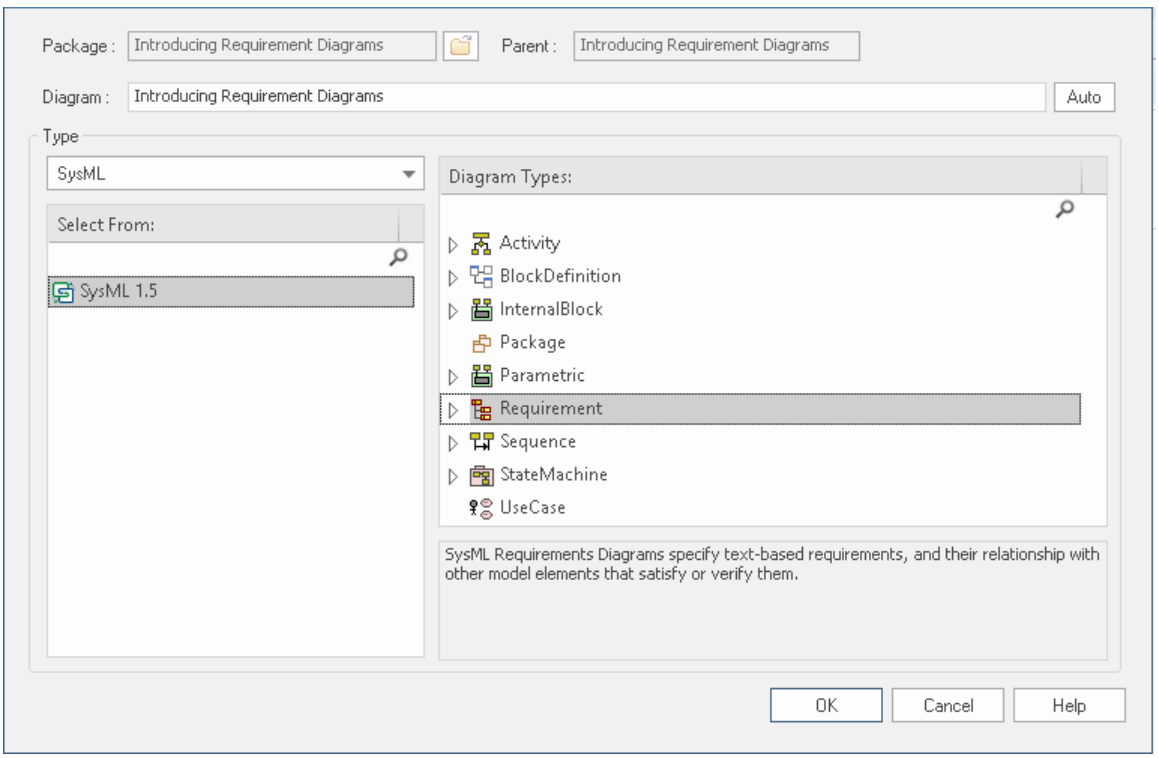
A Requirement diagram can be created from a number of places in the User Interface, such as:

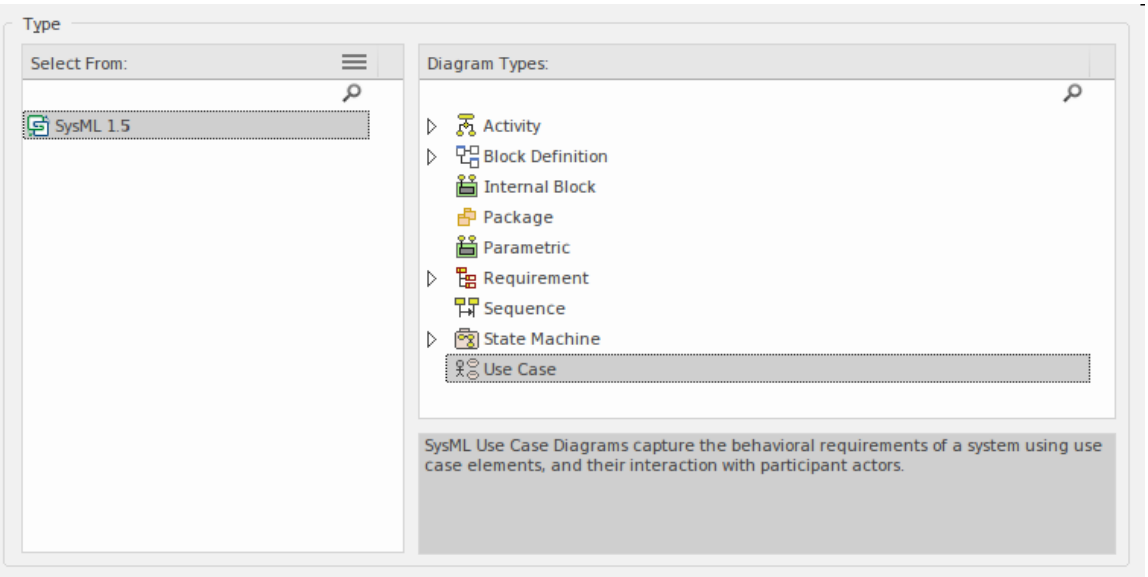
- Design ribbon - 'Add Diagram' icon on the 'Diagram' panel
- Browser window Toolbar - 'New Diagram' icon
- Browser window context menu - 'New Diagram' option

We will use the Design ribbon to create a Requirement diagram. Firstly, select the location in the Browser window where you want the Requirement diagram to be created. As with all diagrams, this can be either a Package or an element, but it is common to insert Requirement diagrams into a Package. Once the Package location has been selected in the Browser window, click on the ribbon option 'Design > Diagram > Add Diagram'.



This option opens the 'Diagram Builder' tab page of the 'Model Builder' dialog, where you can choose the diagram type and specify the diagram name - the name initially defaults to the name of the Package or element that contains the diagram. When you select the SysML Perspective and the version of SysML, a list of diagram types will be displayed, allowing you to select the Requirement diagram. You click on the 'Create Diagram' button to create a new Requirement diagram in the specified location in the Browser window. The Diagram View will be opened, allowing you to start adding elements and connectors that describe the Requirements and their relationships. Enterprise Architect will also display the 'Requirements' page of the Toolbox, which contains the elements and relationships defined by the SysML specification to be applicable for constructing Requirement diagrams. Any number of other Toolbox pages can be opened as required, in addition to the 'Common' elements and 'Common Relationships' Toolbox pages that are displayed by default.





The most important elements and connectors used with the Requirement diagram are:

Elements

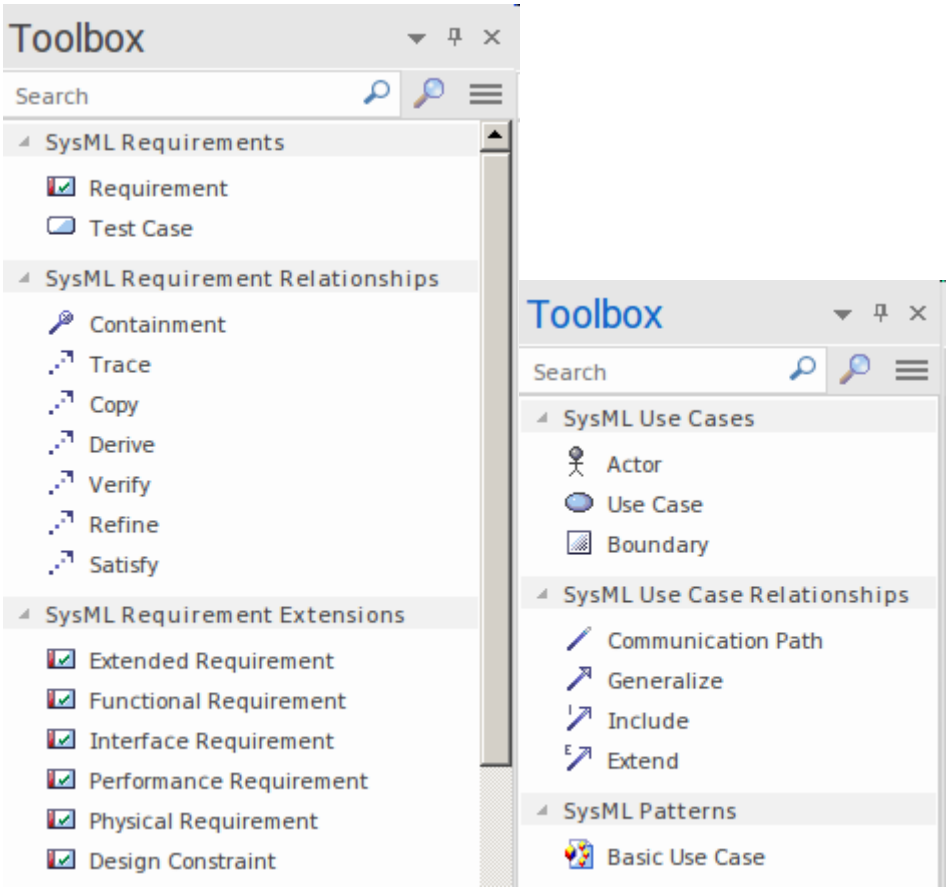
- Requirement - used to define a Requirement
- Test Case - used to describe a Test

Connectors

- Containment - used to provide additional information that helps clarify the Requirement
- Trace - used to connect a Requirement to any other modeling element
- Copy - used to show that one Requirement is a copy of another
- Derive - used to describe the fact that one Requirement is based on or is an extension or derivation of another Requirement
- Verify - used to indicate that a Requirement has been fulfilled
- Refine - used to add refinement or additional information that helps clarify the Requirement
- Satisfy - used to show that one or more model elements in the architecture or design fulfills the notion expressed in the Requirement

Requirement Extensions

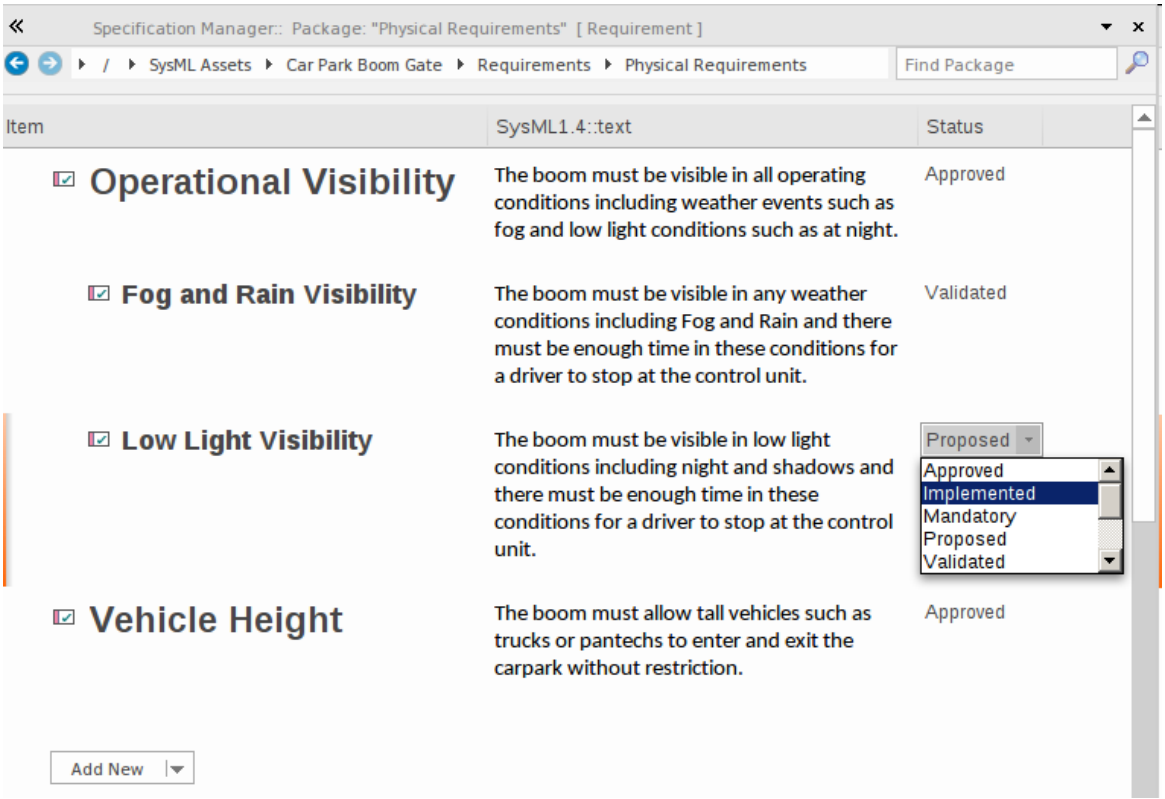
- Extended Requirement - used for extended Requirements
- Functional Requirement - used for Requirements related to function
- Interface Requirement - used for Requirements related to Interfaces
- Performance Requirement - used for Requirements related to performance
- Physical Requirement - used for Requirements related to physical aspects of a system
- Design Requirement - used for Requirements related to design



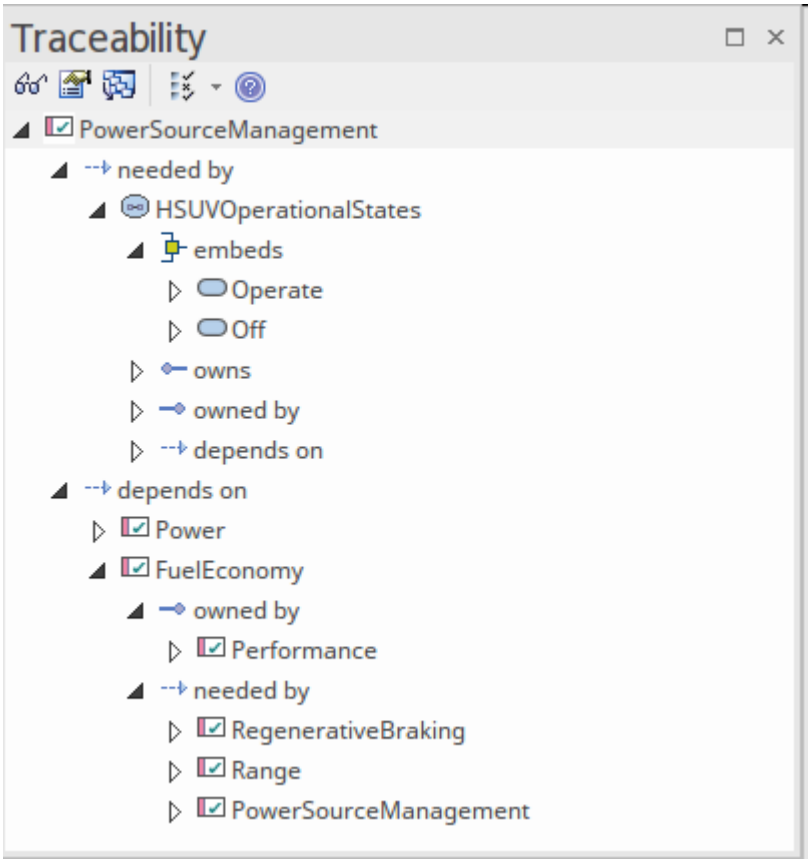
Elements can be added to the diagram by dragging-and-dropping them from the Toolbox into the Diagram View. Relationships can be created by first selecting the required relationship in the toolbox and then dragging-and-dropping between a source and target element. It is common not to create Requirement diagrams that simply list the Requirements but rather to create diagrams that show the relationships between any two Requirements or the relationships the Requirements have with other elements such as Use Cases, Activities and Blocks.

Developing Requirements

Requirement Development includes all the activities and tasks associated with discovering, evaluating, recording, documenting and validating the requirements for a particular project or program of work. Requirements are discovered, analyzed, specified and verified, and Enterprise Architect has a wide range of tools and features to assist the Systems Engineer as they develop requirements. The centerpiece for Requirement Development is the Specification Manager, allowing the Requirement Engineer to enter, view and manage requirements in textual form in a spreadsheet format.



The Specification Manager can be used in conjunction with a platform of other tools, such as diagrams, the Traceability window and the Discussions facility. These windows provide other views of the requirements, giving the modeler and the viewer a deep understanding of how a requirement relates to other parts of the repository, and providing detail not visible through the Specification Manager.


Traceability
PowerSourceManagement
needed by
HSUVOperationalStates
embeds
Operate
Off
owns
owned by
depends on
depends on
Power
FuelEconomy
owned by
Performance
needed by
RegenerativeBraking
Range
PowerSourceManagement

Elicitation

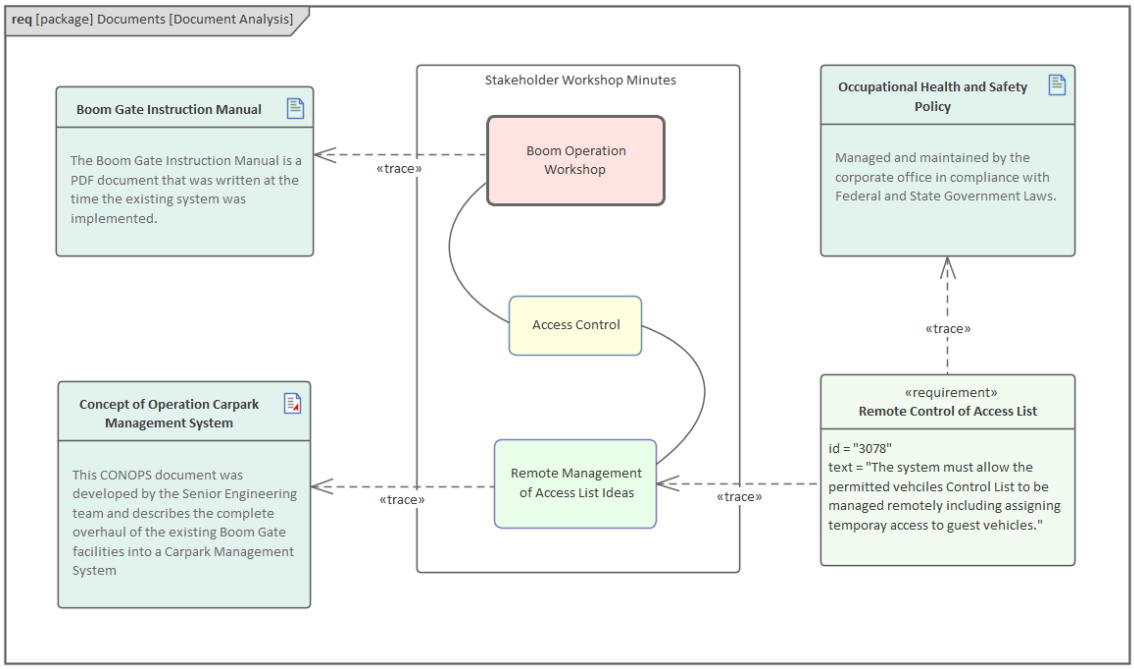
Elicitation is the process of information discovery, the information gleaned from this process will form the precursors to requirements. The information will typically be raw and often heterogeneous, and it will not be until the requirements analysis phase is performed that true requirements will be able to be derived from it. Elicitation will take many forms, and all of the skills of the requirements engineer will be needed to determine which documents, machines, tools, people and processes to examine to discover the information.

Document Sources

Requirements can often be sourced from a wide range of locations, including documents such as a:

- Business Case
- Concept of Operation
- Requirement Specifications (of an Existing System)
- User Manual
- Standards Document
- Policy Document
- Regulatory or Compliance Document

While all of these documents can be developed in Enterprise Architect using the Document Artifact facility, they are typically developed in other tools and live outside the repository. They can be dragged onto a diagram and either imported into the repository or saved as a reference or surrogate for the external document.



Another, and perhaps more useful, option is to add them to the Model Library, which is a document and web page library used to create a catalogue of items that can be referenced by requirements. It is also worth considering reviewing the contents of the documents and incorporating the information as model elements. This has the benefit of an Engineer being able to create traceability relationships between elements such as Business Motivations and problem and solution elements such as Requirements, Use Cases and Components. For more information see the The Model Library Help topic.

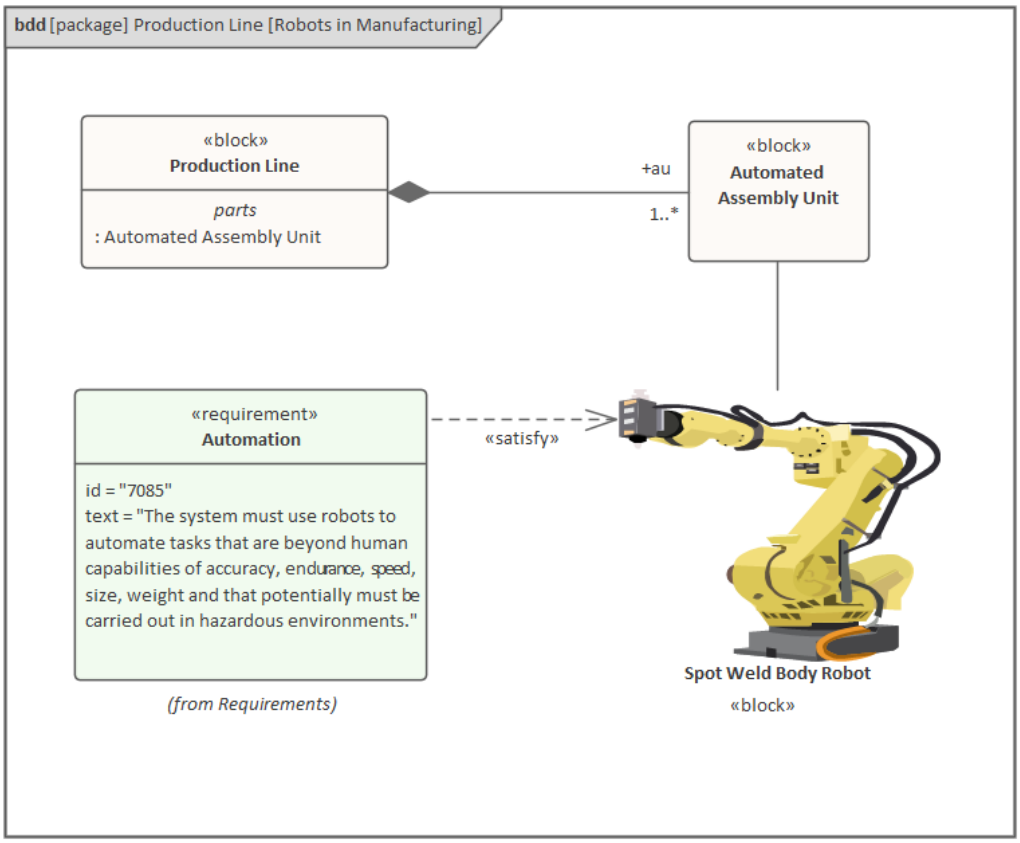
User Observations

Observing users perform their work is a helpful and unobtrusive way of gaining an understanding of:

- The tasks they carry out and
- How they use information and other software and hardware devices to achieve an outcome from their interaction with a system

Even if the processes that support the planned system will be different, the observations of the current processes will provide a useful context for discussions. It will also help the engineer empathize with the user, which can result in a deeper understanding of the issues they face and provide the basis for the discovery of potential solutions. An engineer will often discover unmentioned documents, checklists and clue cards that can help illuminate the process. Equipped with a mobile phone or camera, it is also useful for the engineer to take photographs of the user working, which will help engineers and others recall and discuss the task during the requirements analysis phase.

Enterprise Architect supports the modeler in representing files such as photos and scanned documents directly in the model, creating a rich and expressive representation of the user at work. There is the option to represent these as an Artifact (which, with a single key stroke (F12), will launch the file) or to use a hyperlink or even to include the image itself in a diagram. For more information see the Changing Element Appearance topic.



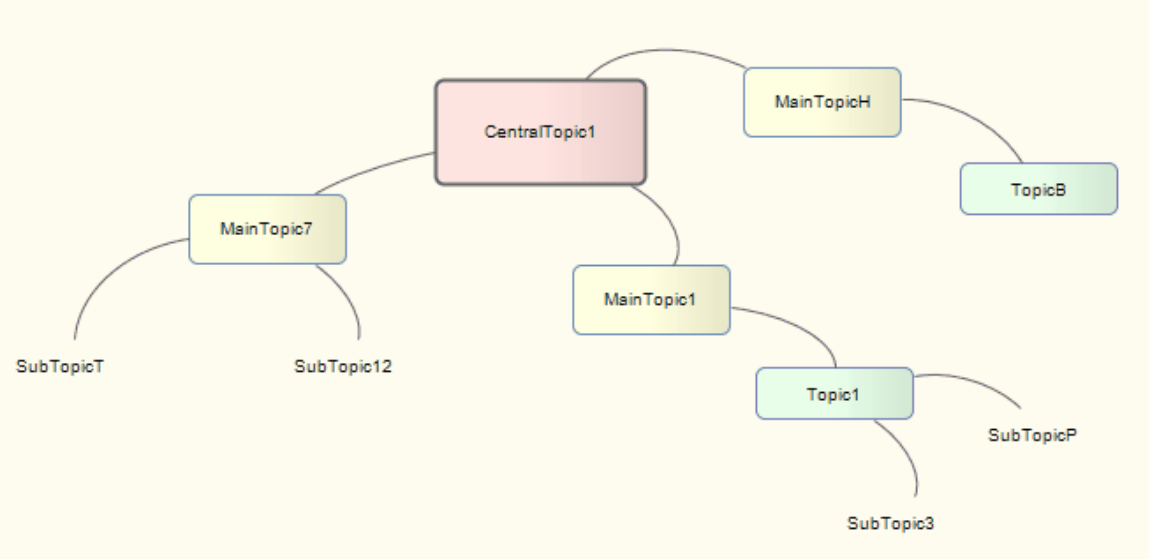
This diagram represents a photograph taken by an engineer of an advanced production line robot assisting with a manufacturing system. The image can be placed into diagrams, and relationships can be drawn between the robot and other parts of the system description, including requirements.

Stakeholder Workshops

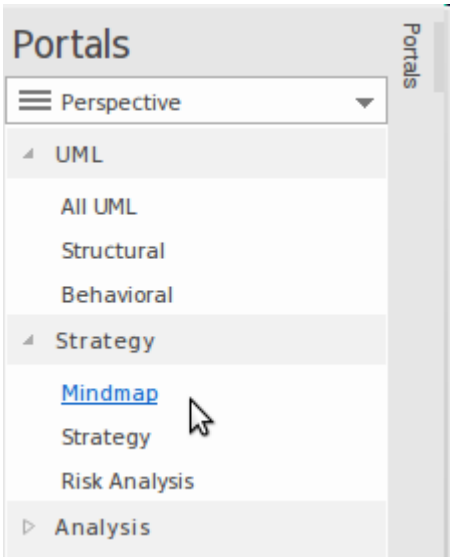
The Requirements Engineer is charged with the difficult task of eliciting requirements, which necessitates excellent communication with the stakeholders, including the customer and the analysis team. One very successful way of facilitating the elicitation of the stakeholders' needs is to run a workshop with all the key stakeholders present. The Requirements Engineer's skills as a communicator, diplomat and mediator are important to create a collaborative and respectful environment conducive to the exploration of the stakeholders' needs and concerns. It is imperative that the engineer uses terminology that the stakeholders understand, and also displays an understanding of or a willingness to learn about the elements that make up the engineering domain.

There is sometimes a misconception that what will be articulated during these workshops is a set of clearly defined requirements that can be entered into the tool as Stakeholder Requirements. This is far from the reality of what happens. Stakeholders will typically articulate a wide range of ideas, including Policies, Business Rules, Data Definitions, Project Management Constraints, Functional Requirements, Business Requirements, existing system problems and even suggested solutions. Even when an external consultant is used to run these meetings, the engineer will not have time to categorize all of these statements inside the meetings. What is needed is a way for the scribe who is tasked with documenting the statements to get them into the tool without any concern for what type of information is being recorded. Having them recorded in the tool rather than scribbled in the engineer's notebook is best practice because it allows them to be displayed during the meeting and for stakeholders to see each others' comments.

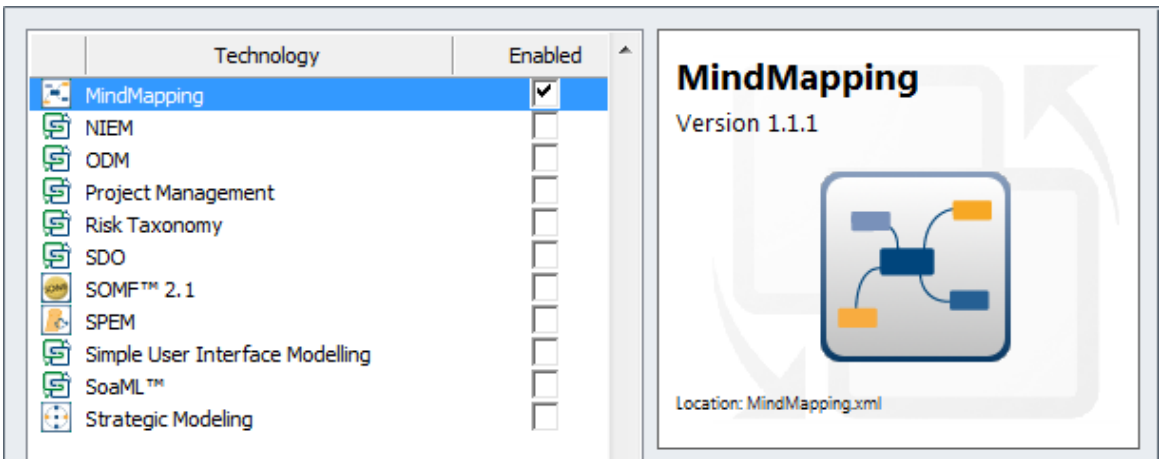
Enterprise Architect has a number of facilities that can help with these workshops. One method that is very practical is to use the Mind Mapping diagram to record the stakeholders statements, which is very effective because it is a well known method and doesn't introduce any of the formality that comes with modeling languages such as SysML. This diagram shows a starter Mind Map created from a pattern that can be altered to suit the workshop need.



The Mind Mapping facility is available by switching to that Perspective or, if it is used commonly, it can be added to a user-defined Perspective Set using the My Perspectives facility.

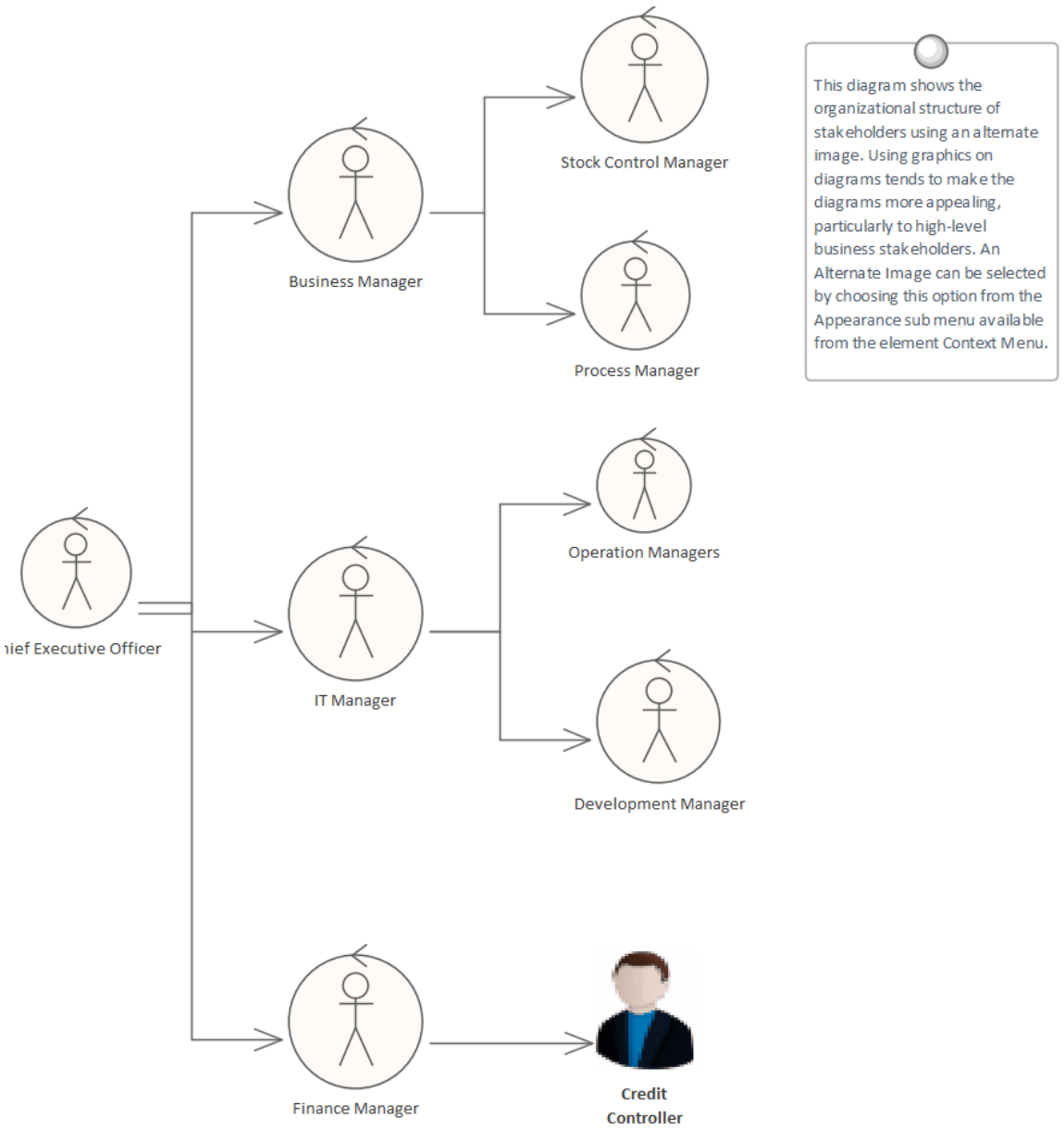


This Perspective, like others, requires the appropriate technology to be enabled, which in this case is Mind Mapping.



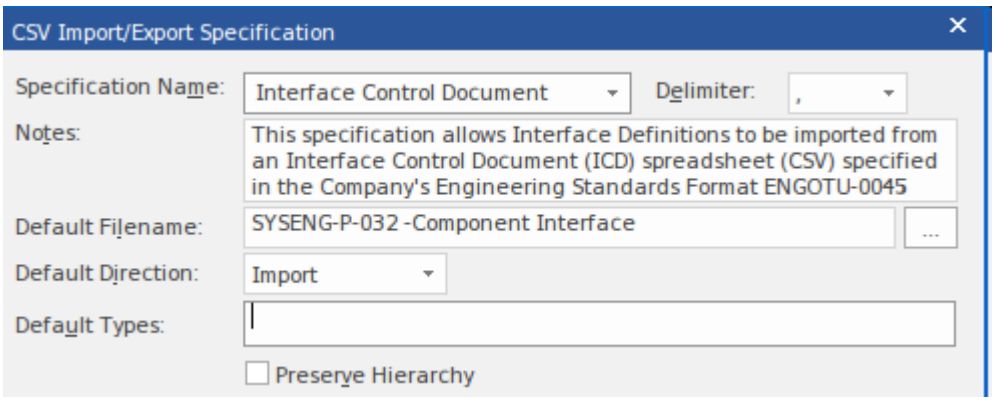
As important terms are uncovered they could be entered into the Project Glossary and, even if there is not time to discuss and debate the agreed meaning, the words will act as an initial list of important entities in the domain. Alternatively, the terms could be created as Blocks in a Block Definition diagram and related to each other with connectors that describe the important relationships between the terms.

The stakeholders can also be modeled and their organizational relationships to each other can be described in a diagram. This is a useful technique that allows key stakeholders to identify themselves in the models, which creates buy-in.


Stock Control Manager
Business Manager
Process Manager
This diagram shows the organizational structure of stakeholders using an alternate image. Using graphics on diagrams tends to make the diagrams more appealing, particularly to high-level business stakeholders. An Alternate Image can be selected by choosing this option from the Appearance sub menu available from the element Context Menu.
Operation Managers
ıief Executive Officer
IT Manager
Development Manager
Finance Manager
Credit Controller

Creating Requirements

Enterprise Architect has extensive support for developing Requirements and provides a number of specialized tools for this purpose. As with all model content, a Modeler is encouraged to check whether the Requirements have been entered into the repository by someone else before embarking on the task of creating new Requirements. It is also possible that the Requirements have been defined in another tool such as a spreadsheet and could be imported into Enterprise Architect without the need to create each Requirement manually.



Enterprise Architect has two locations for Requirements; they can be created in the model as an element that will appear in the Browser window, or they can be created inside another element as an Internal Requirement or Responsibility.

External and Internal Requirements

Enterprise Architect can support any type of Requirement process and allows Requirements to be defined as elements in the model. These are called External Requirements, but the tool also allows Requirements to be defined for a specific element, and these are called Internal Requirements. An engineer who wants to define a user Requirement such as:

The system must allow bus schedules to be updated.

would use an External Requirement. A modeler wanting to describe how a Component should behave would use an Internal Requirement for the Component such as:

The editor must support Unicode.

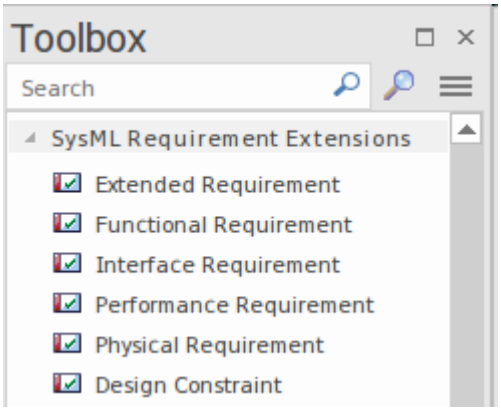
There is often contention between Analysts and Developers as to whether a Requirement should be Internal or External, and Enterprise Architect provides a facility to move Internal Requirements to be external to the element. When they are moved they are still linked to the original element.

Requirement Categories

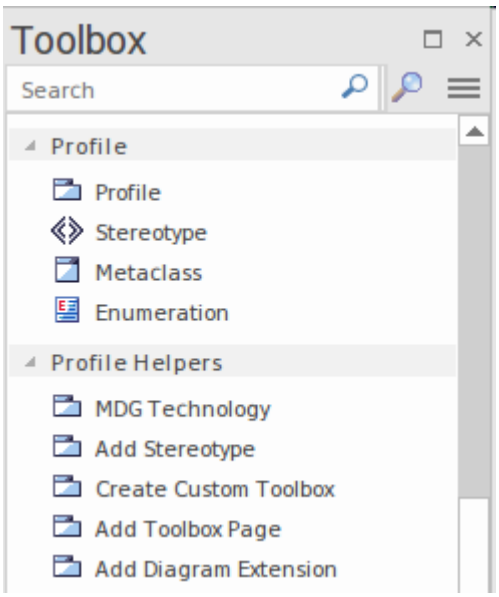
The SysML specification provides a non-normative list of Requirement categories (types). These are stereotyped Requirements that refine or extend the base SysML Requirement, providing a mechanism to create Requirements that serve a particular purpose or describe a particular aspect of a system. For example Physical Requirements can be used to describe some physical aspect of a system such as the weight or size of a component. These and other user-created categories can have any number of additional properties defined such as:

- RiskKind
- VerificationMethodKind

Enterprise Architect conveniently provides these Requirement categories as elements on the SysML Requirement's Toolbox pages.



The tool also provides a sophisticated and fully functional profile system, allowing users to create extensions to the base SysML Requirement and any number of user-defined Requirement categories applicable to the modeling domain or problem space. These stereotyped Requirements can have user-defined properties added that are needed to model the specific Requirement element (or other model element).

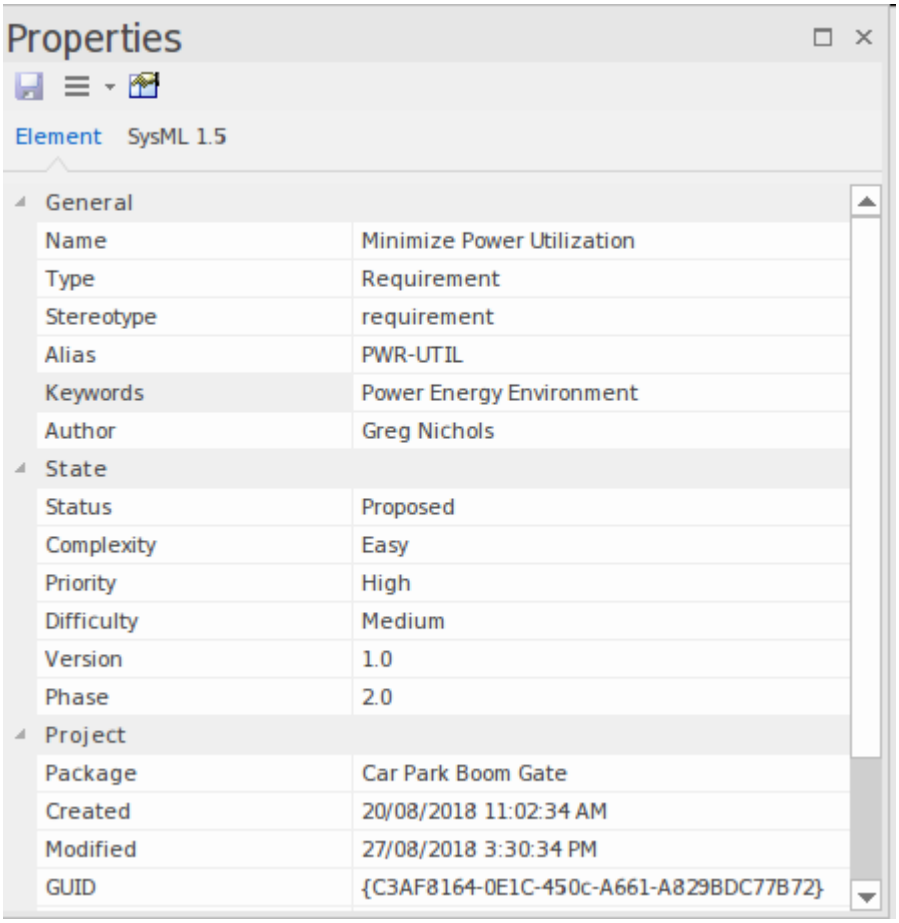


For example a team might decide to include a property of volatility to a Requirement to ensure work is not commenced until the Requirement is stable (i.e. not volatile). As another example, a team might be creating medical devices and need to comply with various statutory standards. Each component that is used as part of the solution might be required to be compliant. A compliance-level property could be created and the component could be assigned a level indicating its compliance from a range of values defined in a spin control or a drop-down list. For more information see the

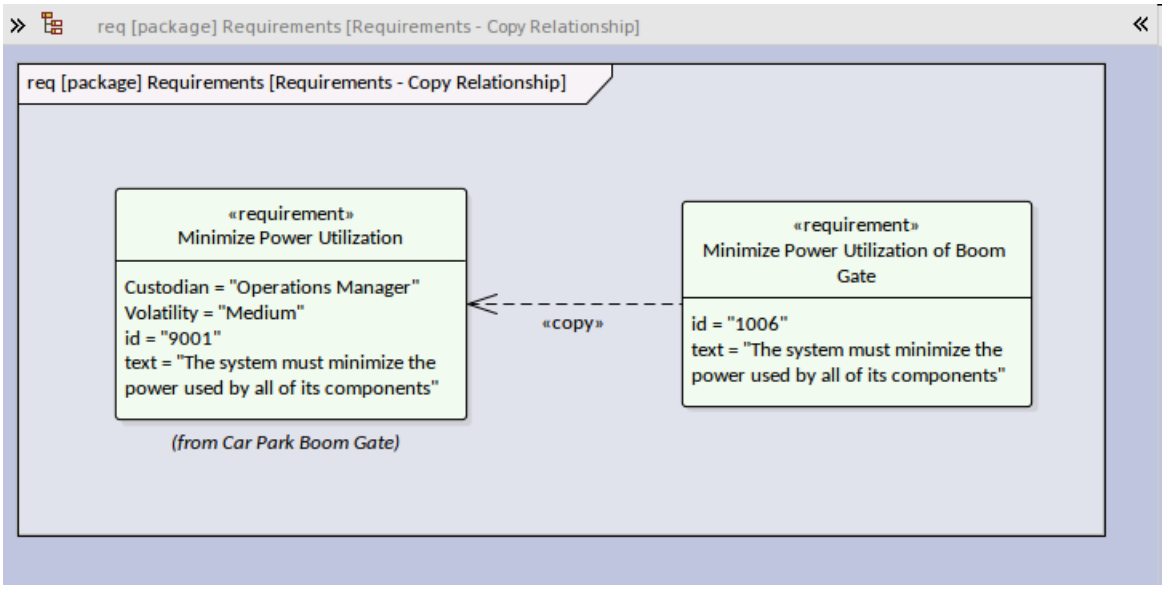
Developing Profiless Help topic.

Requirement Properties

Requirement Development and Management is critical to the success of any project, and the properties of Requirements are important to prioritization and the way they will be elaborated and used within an implementation or development team. All Enterprise Architect elements have standard properties such as Status, Author and Phase, but the Requirement element has additional properties such as Difficulty and Priority.



Some Requirement processes will prescribe specific properties such as *Custodian* and *Volatility* (*Stability*) and these can be configured using Tagged Values that can be applied to each Requirement. The 'Notes' field for a Requirement has special significance as it often contains a formal and contractual description of how the system must behave or perform. For more information see the Element Property Displays Help topic.


req [package] Requirements [Requirements - Copy Relationship]
req [package] Requirements [Requirements - Copy Relationship]
«requirement»
Minimize Power Utilization
Custodian = "Operations Manager"
Volatility = "Medium"
id = "9001"
text = "The system must minimize the power used by all of its components"
(from Car Park Boom Gate)
«copy»
«requirement»
Minimize Power Utilization of Boom Gate
id = "1006"
text = "The system must minimize the power used by all of its components"

Specification

Requirements specification is an important aspect of the evolution of a Requirement. It provides an important catalogue of statements about the system's behavior in both normal and abnormal conditions. The Requirements will be of interest to a wide range of stakeholders including:

- Engineering Managers
- Architects
- Designers
- Customers or their surrogates
- System Engineers
- Software Engineers
- Testers
- Compliance Managers
- Quality Engineers
- Safety Engineers

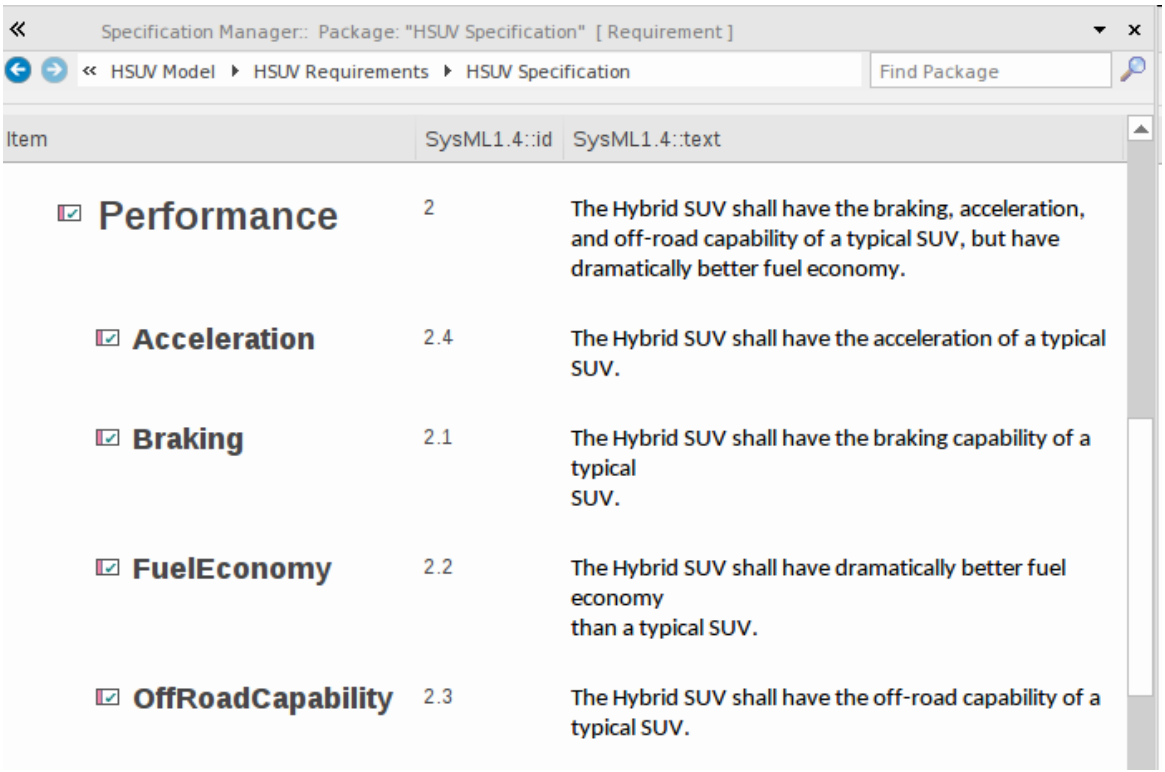
All these groups will have both input to the Requirements and a need to use the catalog of Requirements in their work. There is a variety of ways in which requirements can be specified in Enterprise Architect, including:

- Directly in the Browser window
- On a diagram
- Using the Specification Manager

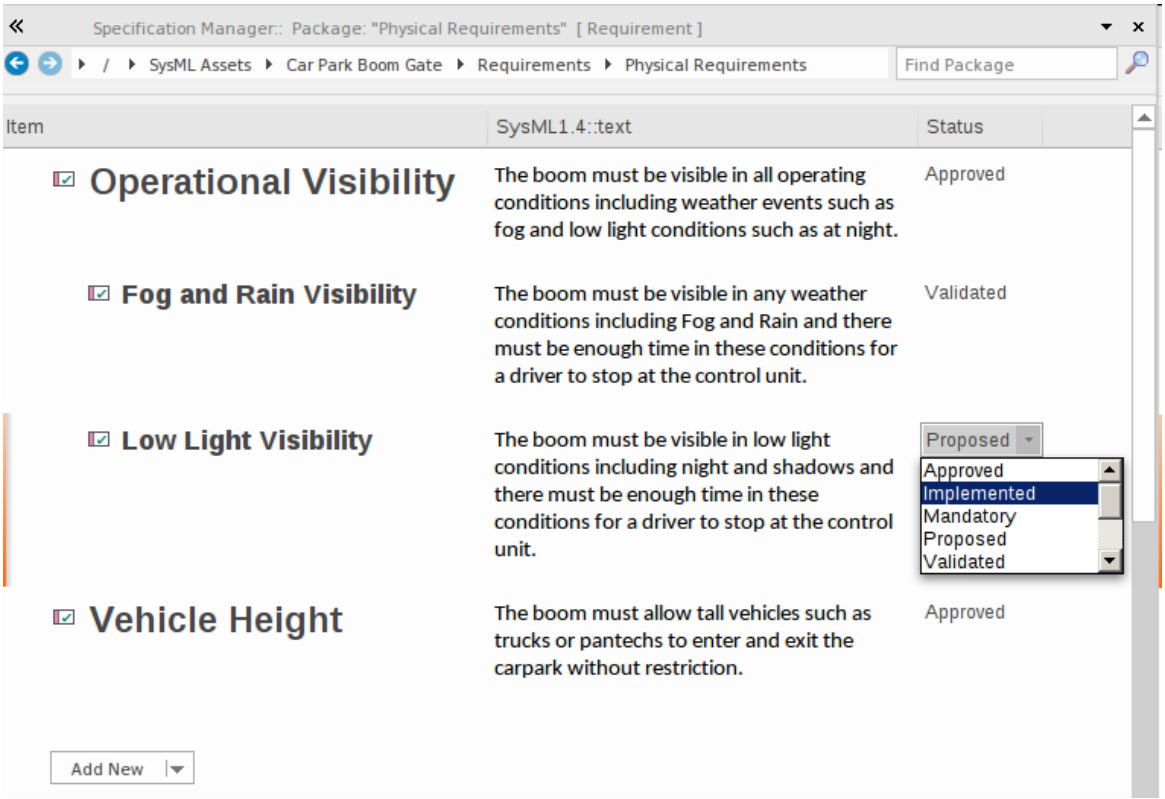
We will look at the Specification Manager in the next section, and you will see that it provides great flexibility when working with Requirements and other elements with textual content.

Meet the Specification Manager

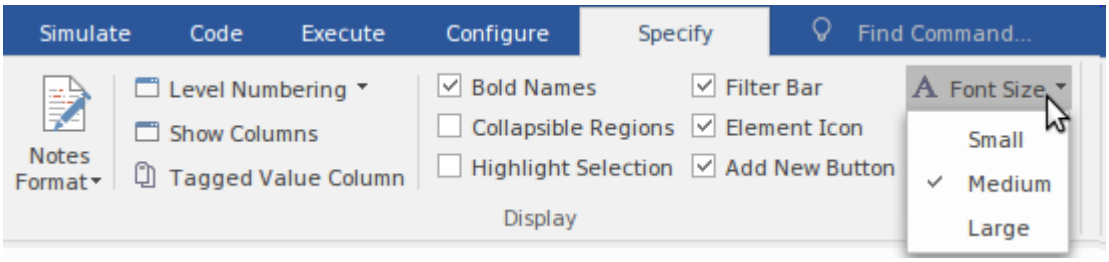
The Specification Manager is a unique and effective tool providing a spreadsheet or word processor view that can be used to manage any element, although it is particularly useful when working with Requirements that always have descriptive text to describe the Requirement in detail. New Requirements can be created with names and detailed descriptions, and properties such as Status and Priority can be added or changed from drop-down lists. Existing Requirements can be viewed and managed in a convenient view - such as diagrams and windows - and changing them in the Specification Manager will change them in all other places in the repository.



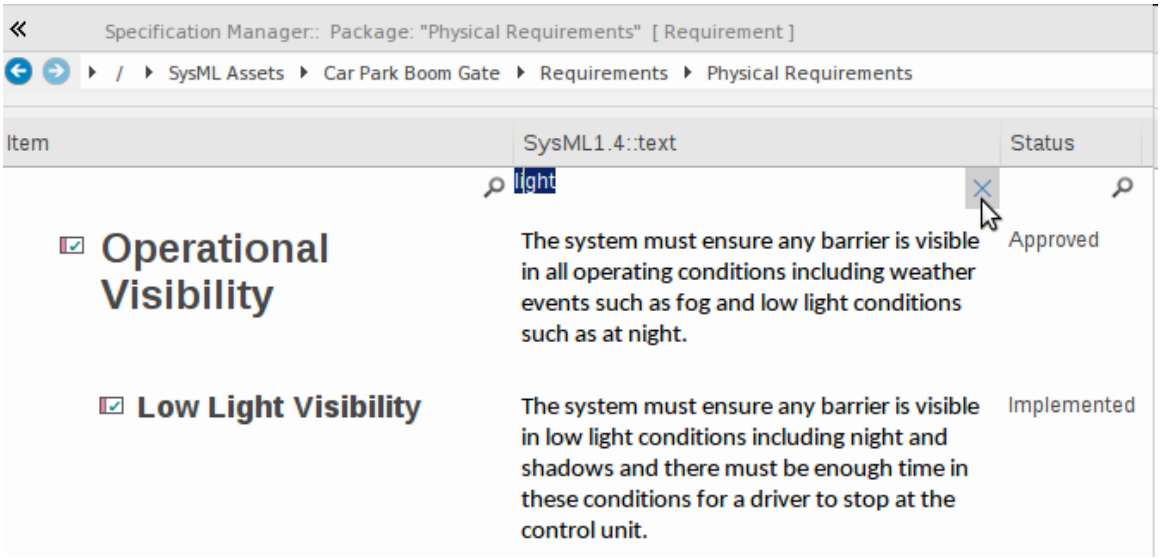
The Specification Manager is the perfect tool for those analysts who are more comfortable working with text rather than diagrams, and who are accustomed to working in a Word Processor or Spreadsheet. It has the added advantage that the Requirements are part of a model and can be traced to other element, including Business Drivers, Stakeholders and Solution Components. In this image it can be seen that the Requirement status and other element properties can be edited from drop-down lists.



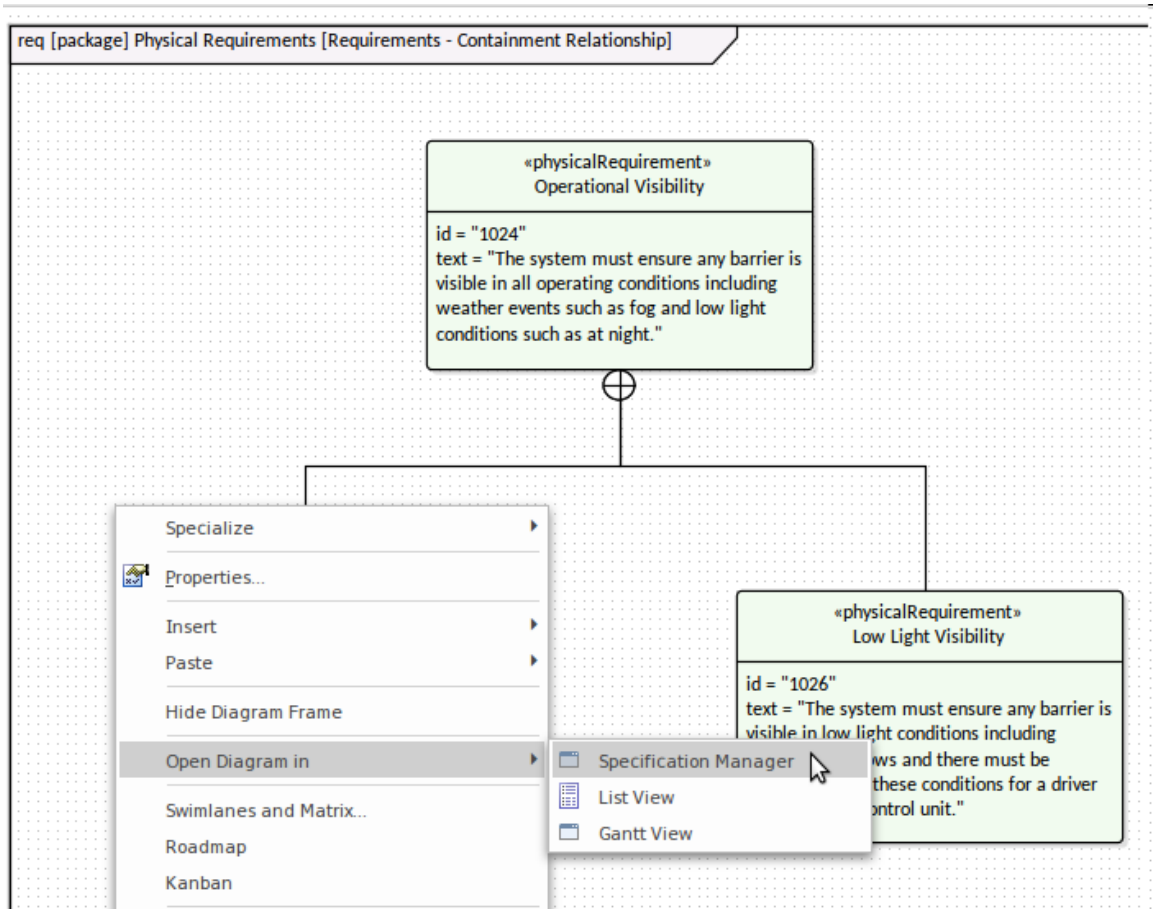
There is a wide range of options that provide great flexibility when working with the Specification Manager, including showing notes in columns as in a spreadsheet or inline as in a document, and adjusting the size of the text. These options are available from the 'Specification - Specify' ribbon, which is conditionally displayed when the Specification Manager is launched.



Filters provide a useful way to restrict the display to elements that contain a word or text fragment in a selected column. In this illustration a modeler has decided to restrict the display to all Requirements that contain the word 'light' in the text of the Requirement. This is a great productivity tool when working with large sets of Requirements, and it can be used to locate all Requirements with a particular status, priority, complexity or even all Requirements owned by a specified stakeholder or team, if they have been defined in the model.

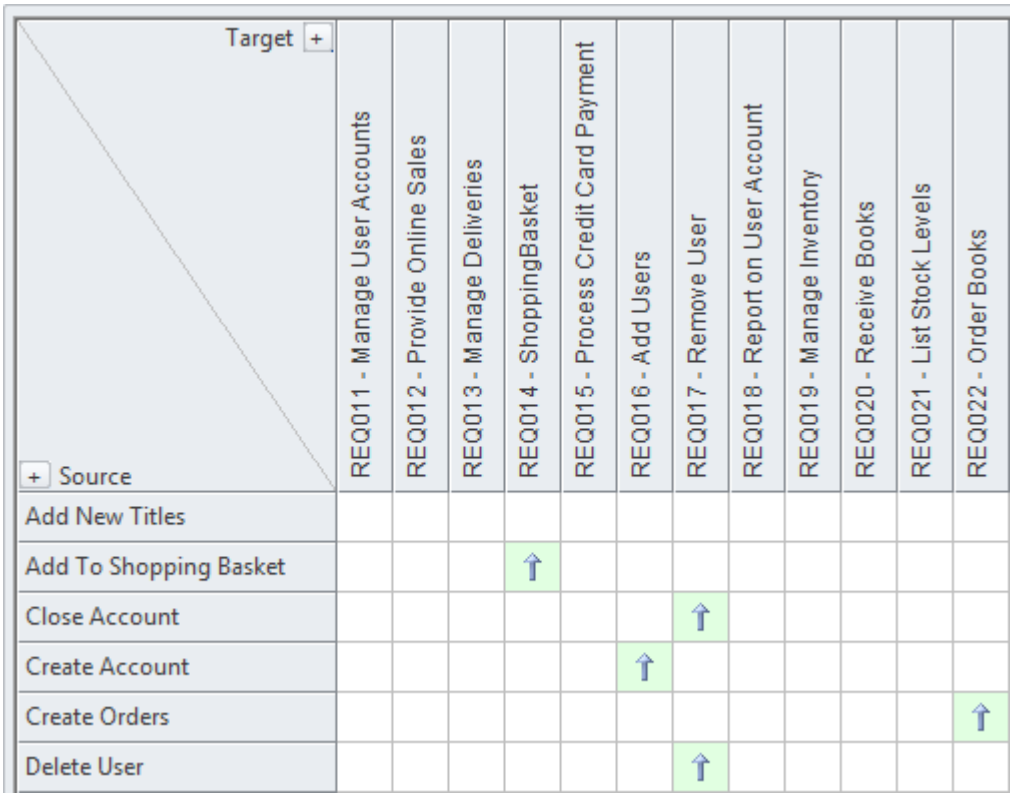


A diagram can also be opened from the Specification Manager, allowing you to edit the elements on the diagram as a group. This is a compelling and welcomed view for some non-technical staff, including managers and customers. For more information see the *Specification Manager* Help topic.



Analysis

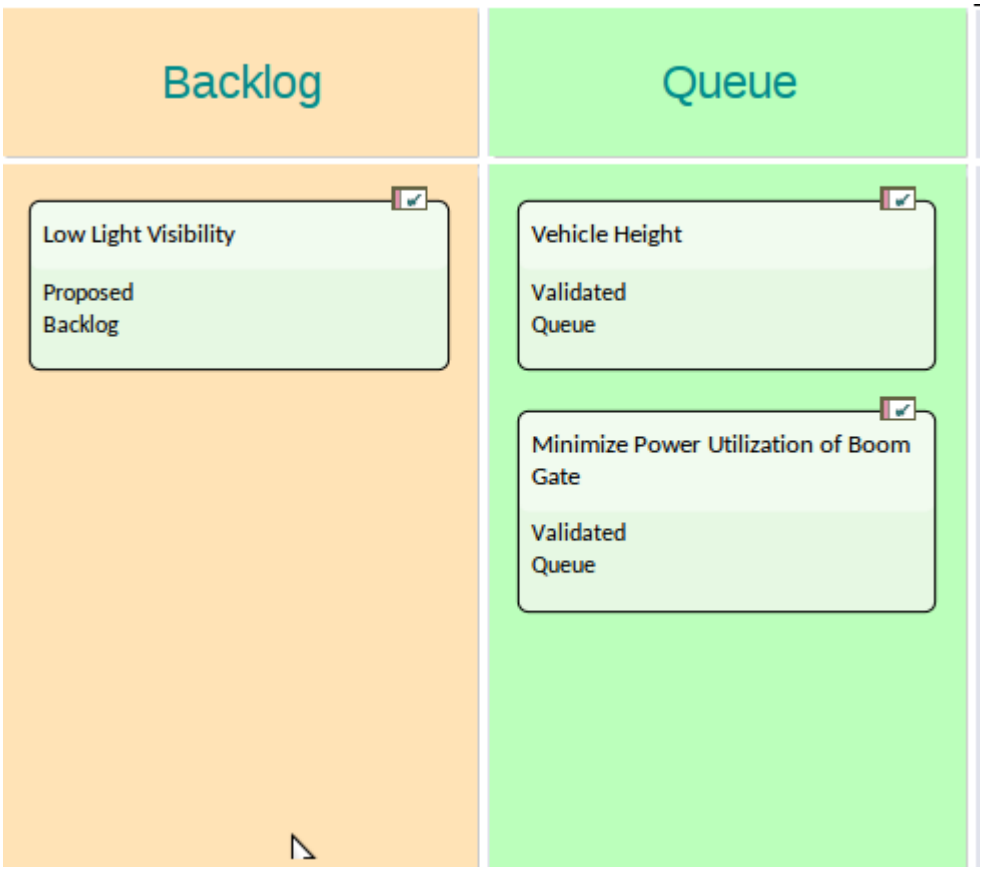
The analysis phase of requirements development ensures that the requirements discovered in the elicitation phase have been articulated correctly and have the correct format, level of detail and properties, and form a correct and cohesive set. As a result of the disparate sources and methods of elicitation the requirements recorded in the elicitation phase will need some massaging and balancing - it is quite common for example, to find repeated or overlapping requirements or for a systems engineer to inadvertently have omitted to record the concerns of one or more stakeholders. Tools such as the Relationship Matrix and the Traceability window will help reveal omissions and issues with requirements. The Discuss & Review window and Chat & Mail window - including the Model Mail facility - will also provide mechanisms for discussing, reviewing and chatting about the Requirements with other team members.



Source \ Target	REQ011 - Manage User Accounts	REQ012 - Provide Online Sales	REQ013 - Manage Deliveries	REQ014 - ShoppingBasket	REQ015 - Process Credit Card Payment	REQ016 - Add Users	REQ017 - Remove User	REQ018 - Report on User Account	REQ019 - Manage Inventory	REQ020 - Receive Books	REQ021 - List Stock Levels	REQ022 - Order Books
Add New Titles												
Add To Shopping Basket				↑								
Close Account							↑					
Create Account						↑						
Create Orders												↑
Delete User							↑					

Prioritize the Requirements

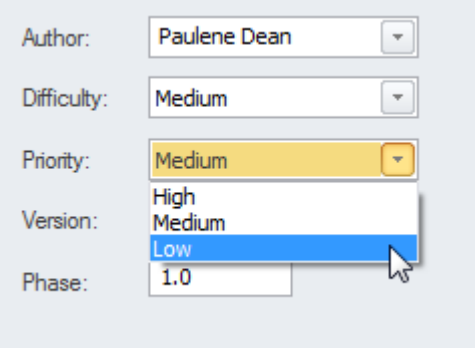
Prioritizing Requirements is imperative to the success of a project, as it ensures that analysis, development, testing and implementation resources are focused on the most critical aspects of the system. Prioritization is a decision process that allocates a priority to each Requirement; the most common criteria for categorization is business value. Business value is typically determined by the cost-benefit analysis of the value the implemented Requirement will produce for the organization or its customers. Other factors might be policy or regulatory compliance, urgency, business or technical risk and the likelihood of success. Requirements can be visualized in a Kanban board which can be used to indicate priority by moving items from a Backlog lane to a Queue Lane and also allowing items to be ordered within the lanes. For more information see the Kanban Boards Help topic.



Alternatively, Searches or Model Views could be used to create a list of requirements based on some criteria that would enable the Requirements to be prioritized.

Requirement Priority Property

There is a wide range of criteria that can be used for prioritization, and each organization and project will typically use some type of weighted average to determine the priority. Enterprise Architect has flexible and complete support for Requirement prioritization, as each element has a built-in 'Priority' property that can be set to indicate its priority, allowing the user to select the allocated priority from a drop down list.

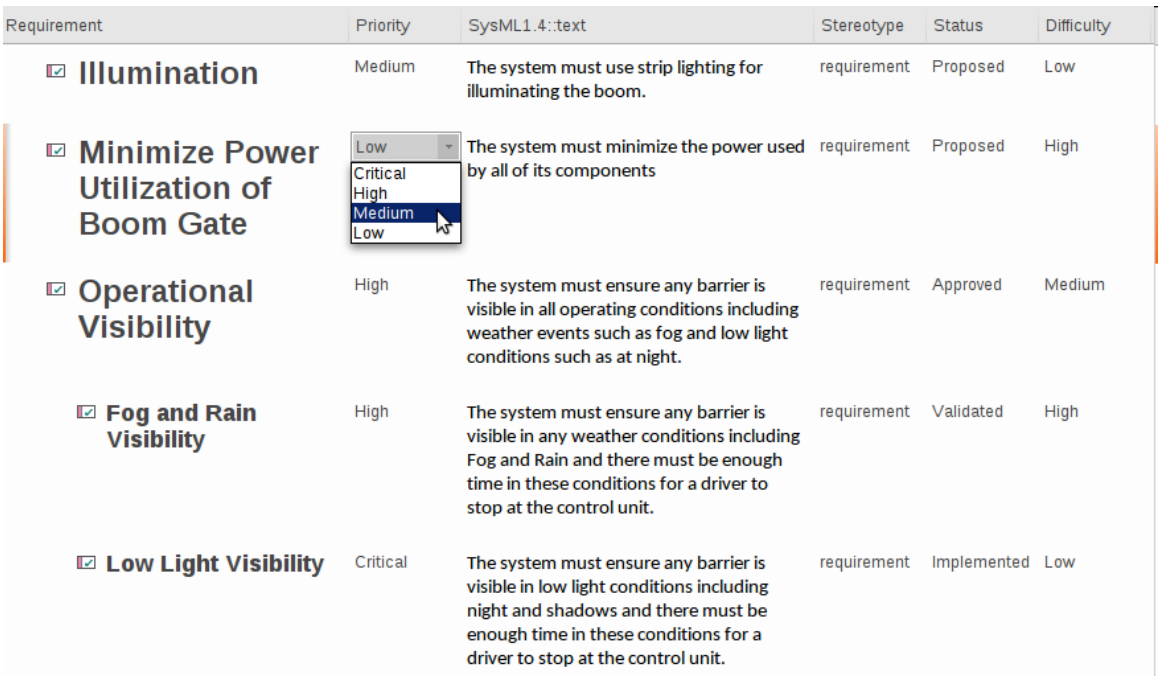


The list of priorities is conveniently pre-loaded when you install Enterprise Architect, but these can be edited or completely revised to suit an organization or project. They can even be imported as reference data from a previous project or, if the current project was created based on a template, the organization's priorities could be pre-loaded from the base model. They can be set up using this ribbon option:

Settings > Reference Data > Model Types > General Types > Priority

Changing the Priority Collaboratively

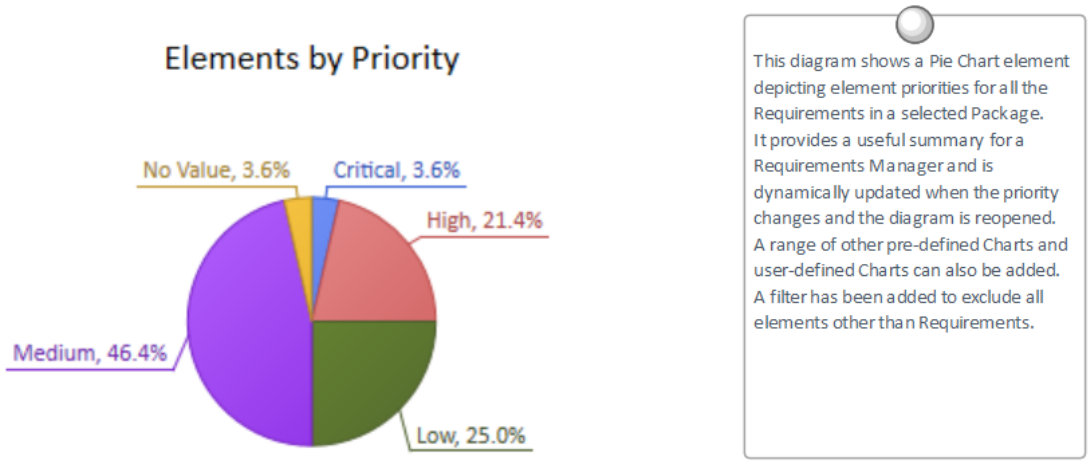
The process of selecting criteria and assigning priority is typically collaborative, and is often done in a workshop with stakeholders or their representatives debating the categorization. In previous eras this was a laborious and difficult process, but Enterprise Architect has some useful features for working with Requirement properties, including priority. There are a number of windows - including the Package List and the Diagram List - that support working with the Requirements and editing the priority in-line, automatically filtering or sorting the list of Requirements based on the newly assigned priority. The Specification Manager is a useful tool for this purpose, providing a text-based interface where the Requirements and their notes can be viewed and priorities can be selected from a drop down list. The interface also displays a number of other properties that are typically useful for prioritization, such as Status and Complexity. For more information see the Editing Elements Help topic.



When a Requirement property is changed and saved in any window or diagram, the property will be changed in all other views and any other users viewing the repository will immediately be able to see the change.

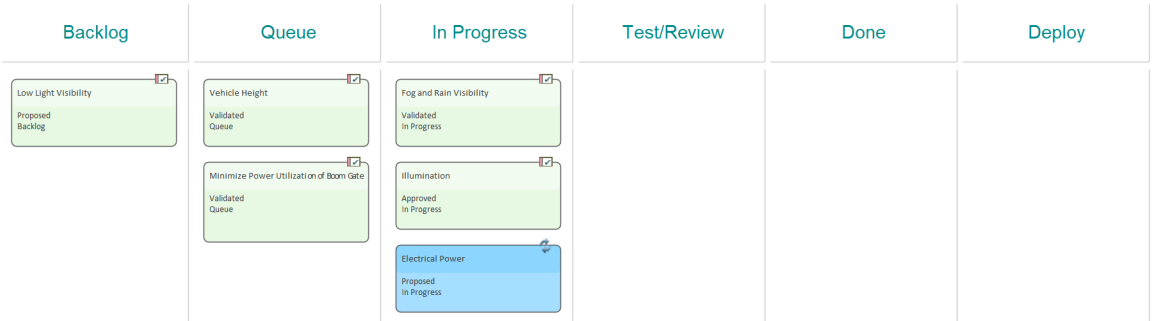
Dashboard Diagams

Enterprise Architect has a series of Dashboard diagrams that can be used to create a compelling view of the Priority of Requirements in a Package, with the option to include sub-Packages. There are a number of pre-configured Charts that can be used to display the ratio of Priority values for Requirements in a part of the model. Filters add another level of user configuration, allowing a modeler to, for example, exclude Requirements of a particular Status or ensure only Requirements for the current phase are displayed. For more information, see the Dashboard Diagrams Help topic.



Visualization with Kanban Boards

Enterprise Architect has a Kanban Board diagram that can be used to manage Requirements and other specification or project management elements such as Change. The Kanban Board is particularly useful for managing the priority of Requirements and other elements. The elements can simply be dragged onto the diagram and then between columns, allowing teams to manage and visualize the progress a Requirement makes between specification and implementation.

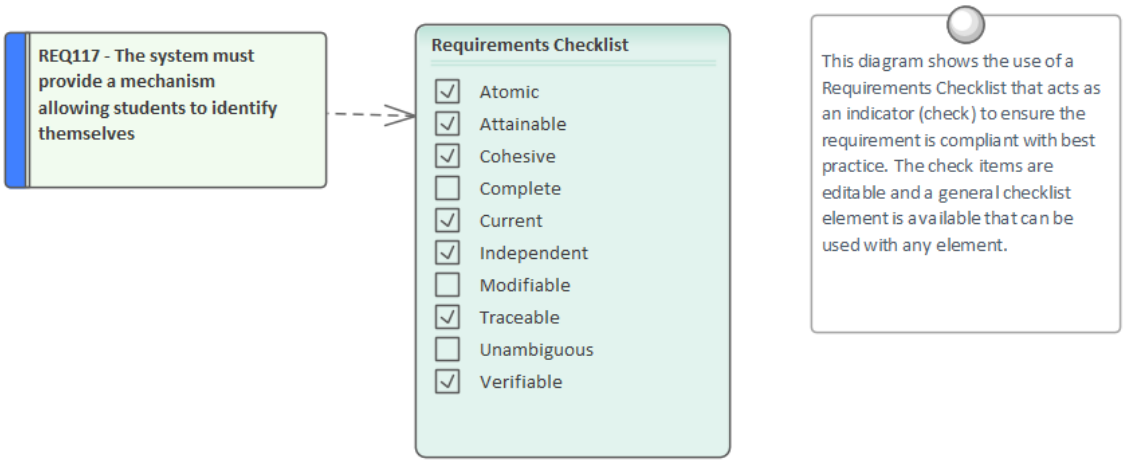


The Kanban diagram can be configured so that when an element is dragged between columns the priority of the element is automatically changed. For more information see the Kanban Boards Help topic.

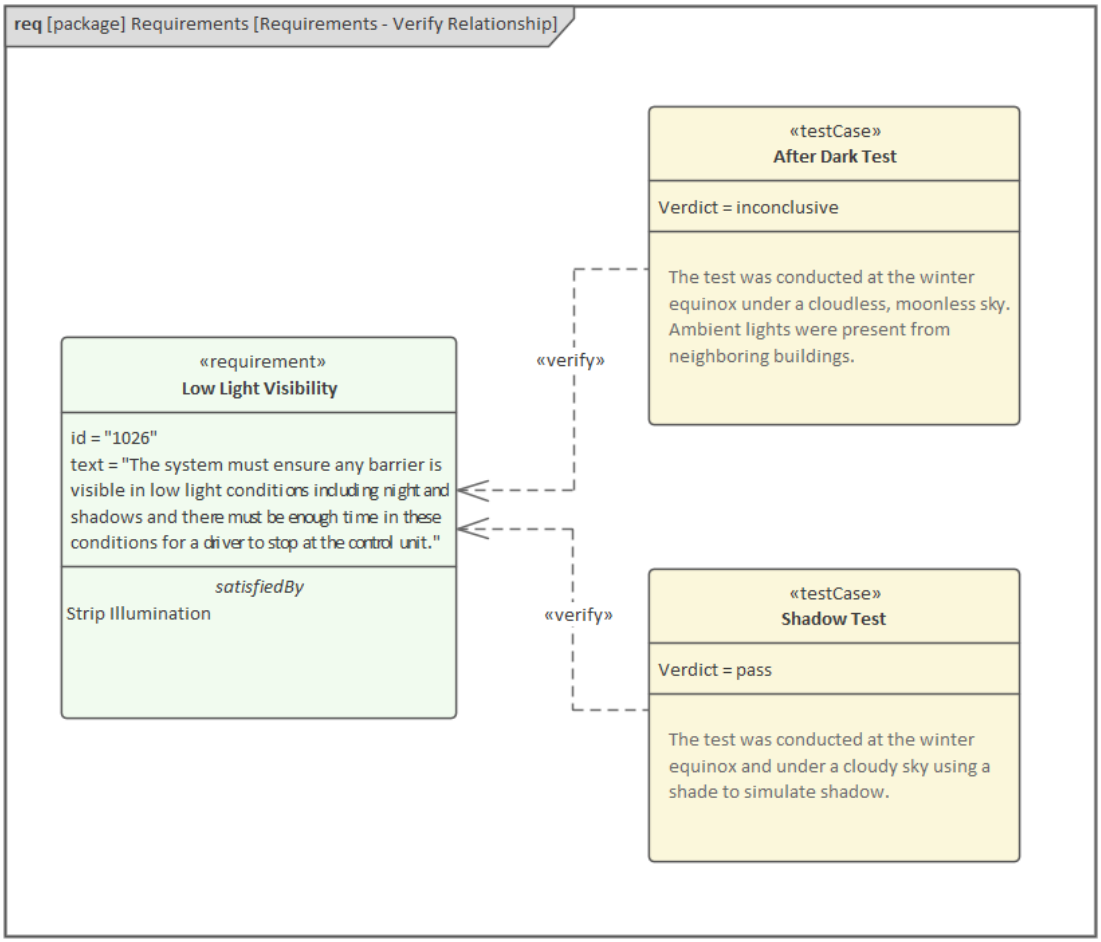
Validation

Requirements validation is necessary to make sure the Requirements are of a high standard, suitably define the Customer's problem (or opportunity) and are sufficient for the implementation teams to design and implement the product. It is imperative that the requirements have the desired level of quality and are complete and necessary. There are a number of ways that Requirements can be validated, but probably the two most common ways are to perform team reviews and to assign test cases to the requirements.

The team reviews are typically conducted by team members or other analysts who have some familiarity with the domain, but were not themselves responsible for the requirements development or management. Enterprise Architect has a handy tool to assist with this process, called the Formal Review, which works across the entire model and allows reviewers to record their findings in discussion documents and to reference model elements. There is also a Requirements Checklist element available from the 'Extended Requirements' page of the Requirements Toolbox, which provides a useful mechanism for checking the quality of Requirements.



Test Cases can be defined at a number of levels from User Acceptance tests down to Unit tests. Defining the test cases early in the requirements development process creates a double check on the Requirements, because when test cases are defined issues with the Requirements are often uncovered. Enterprise Architect has a number of facilities to define test cases and a modeler can select whichever is the most appropriate for the endeavor.

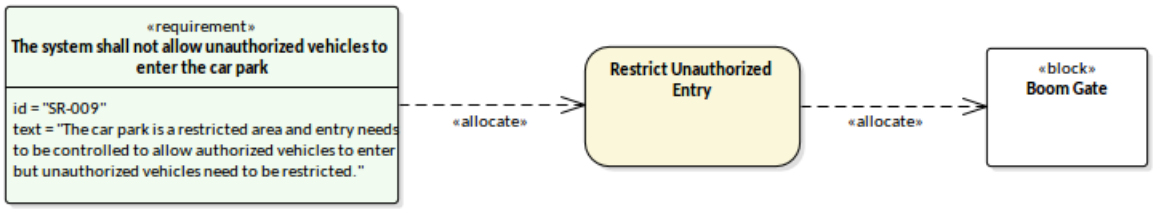

req [package] Requirements [Requirements - Verify Relationship]
«testCase»
After Dark Test
Verdict = inconclusive
The test was conducted at the winter equinox under a cloudless, moonless sky. Ambient lights were present from neighboring buildings.
«verify»
«requirement»
Low Light Visibility
id = "1026"
text = "The system must ensure any barrier is visible in low light conditions including night and shadows and there must be enough time in these conditions for a driver to stop at the control unit."
satisfiedBy
Strip Illumination
«verify»
«testCase»
Shadow Test
Verdict = pass
The test was conducted at the winter equinox and under a cloudy sky using a shade to simulate shadow.

Visualizing Requirements

Requirements can be visualized in a number of different ways using different features of the product. Some of the methods allow the user to create new and edit existing requirements while others simply provide a way of viewing the requirements. Requirements, like all elements in the SysML, can form part of a graph and many of the important semantics are expressed in these relationships; for example the relationship between a requirement and the test cases that verify it. It is, however, quite common for requirement analysts, managers and other stakeholders to want to view the requirements independently of any relationships that a requirement might participate in. We will look at a number of these facilities now and others will be covered in the Visualizing Requirement Relationships topic.

Requirements Diagrams

The Requirements diagram can be used to visualize Requirements and their relationships to other elements, including other Requirements. You can of course view Requirements in a number of other ways, but for many stakeholders the Requirements diagram will be more appealing as it provides a graphical way to view the Requirements' connections to other important parts of the model, including stakeholders, architecture, design and tests. This diagram shows how Requirements can be viewed along with their connection to other elements. A Requirement stating that unauthorized vehicles should not be allowed to gain access to the car park is allocated to the Activity 'Restrict Unauthorized Vehicles', which in turn is allocated to a Block representing the Boom Gate.

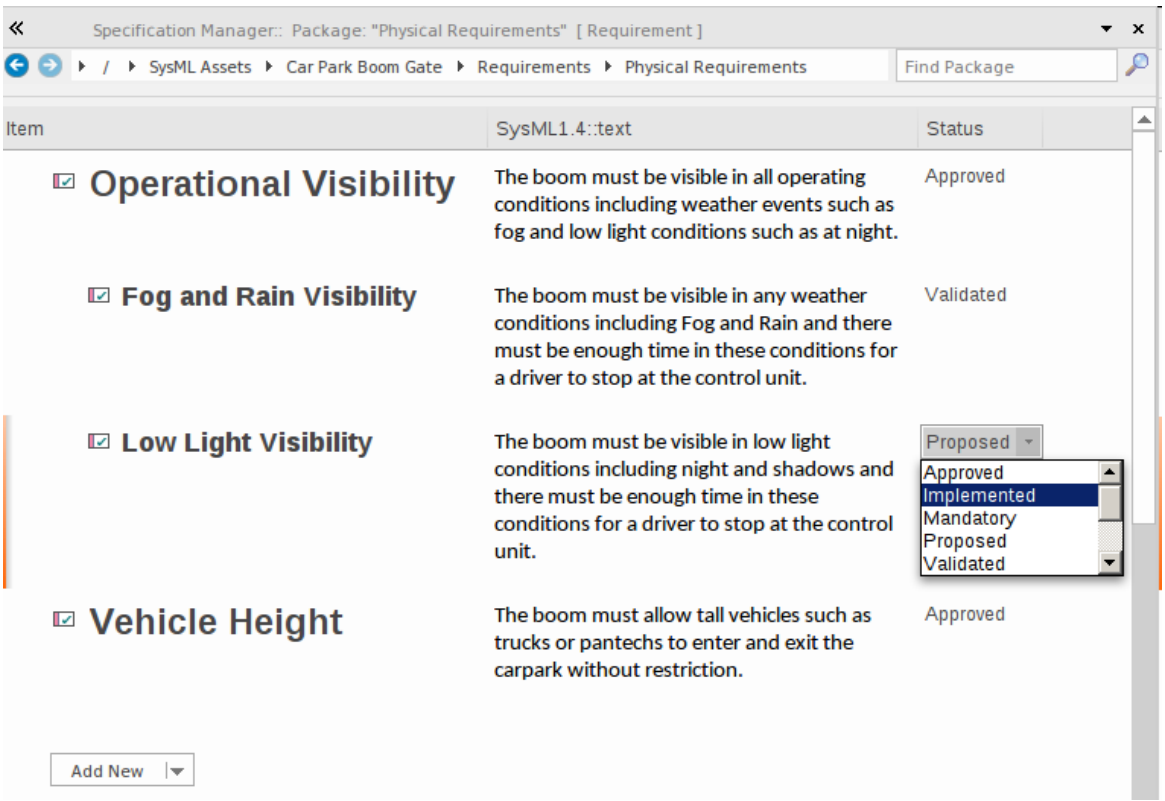


A Requirements diagram can be created from a number of different locations in the user interface, including from the ribbon option 'Design > Diagram > Add Diagram'.

The 'Model Builder' dialog will be displayed with the 'Diagram Builder' tab page selected. The Requirements diagram can be selected from the list of SysML diagram types. For more information, see the SysML Requirements Modeling Help topic.

Specification Manager

The Specification Manager is the central tool for working with Requirements in Enterprise Architect, and has been designed from the ground up to be a tool that allows Requirements to be created and managed through an intuitive and fully featured interface. For those engineers or other stakeholders who are accustomed to working with spreadsheets or word processor documents, the tool will seem natural and emulates both these modes of visualization, allowing a user to toggle between spreadsheet mode and document mode.



The Specification Manager can be opened from the ribbon: 'Design > Package > Specification View'.

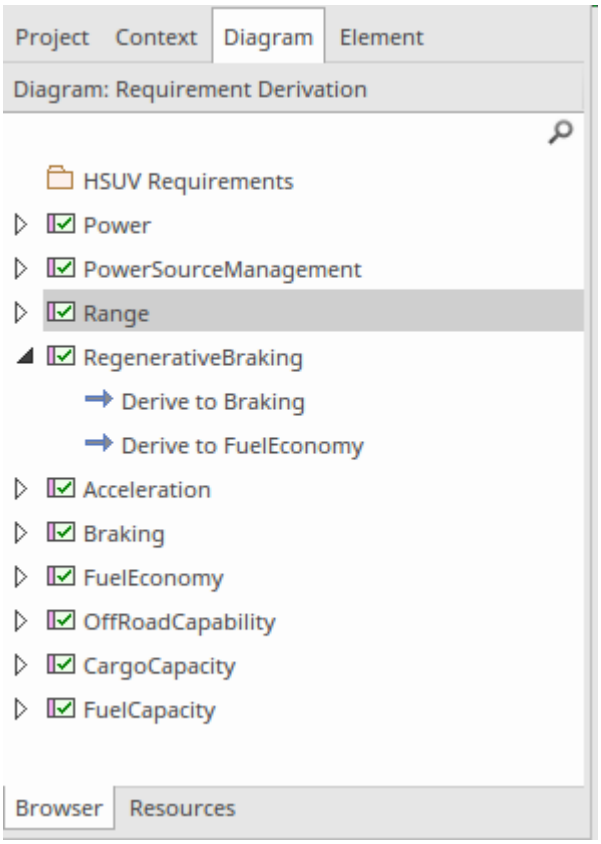
The Specification Manager can be used to view any type of element, but is particularly suited to elements with a large textual component such as Requirements. Any elements that are changed in the tool will also automatically be changed in Requirements diagrams, in the Browser window and in other catalogs such as tables. For more information see the Specification Manager Help topic.

Browsers and Views

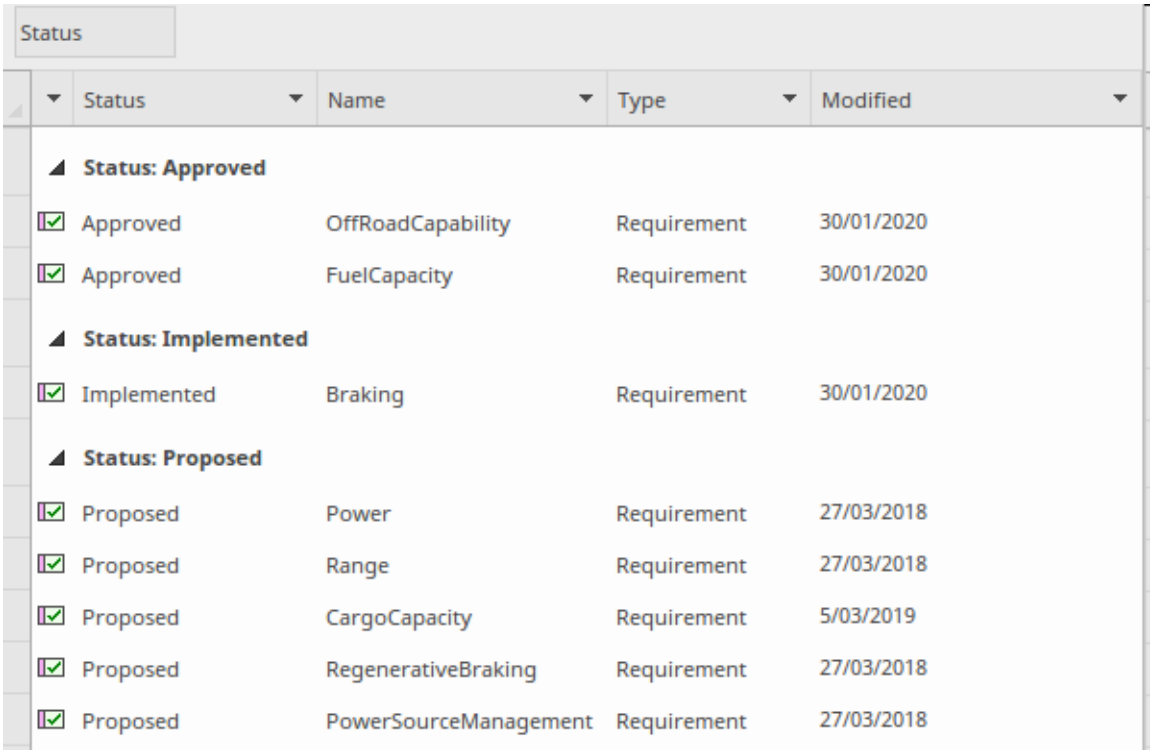
The Browser is the central navigation tool, which can be used to structure and explore the contents of a repository including working with requirements. The Browser has a number of tabs that allow the contents of the repository to be viewed in particular ways. We have looked at the Browser in an earlier topic, but will speak about its relevance for visualizing requirements.

Most system engineers will try and keep their requirements for a particular project or endeavor in a single location, although there might be circumstances where they need to be separated; for example, for contractual or schedule reasons. Once a Requirements Package has been selected in the 'Project' tab, a user can switch to the 'Context' tab to get a focused view - effectively removing the noise of the other elements outside that context. An individual requirement can then be selected and the 'Details' tab selected on the Inspector window to focus on the properties of the selected requirement.

The elements and relationships contained in an open diagram can also be visualized through the 'Diagram' tab of the Browser, providing an alternative way of viewing the contents of a diagram.



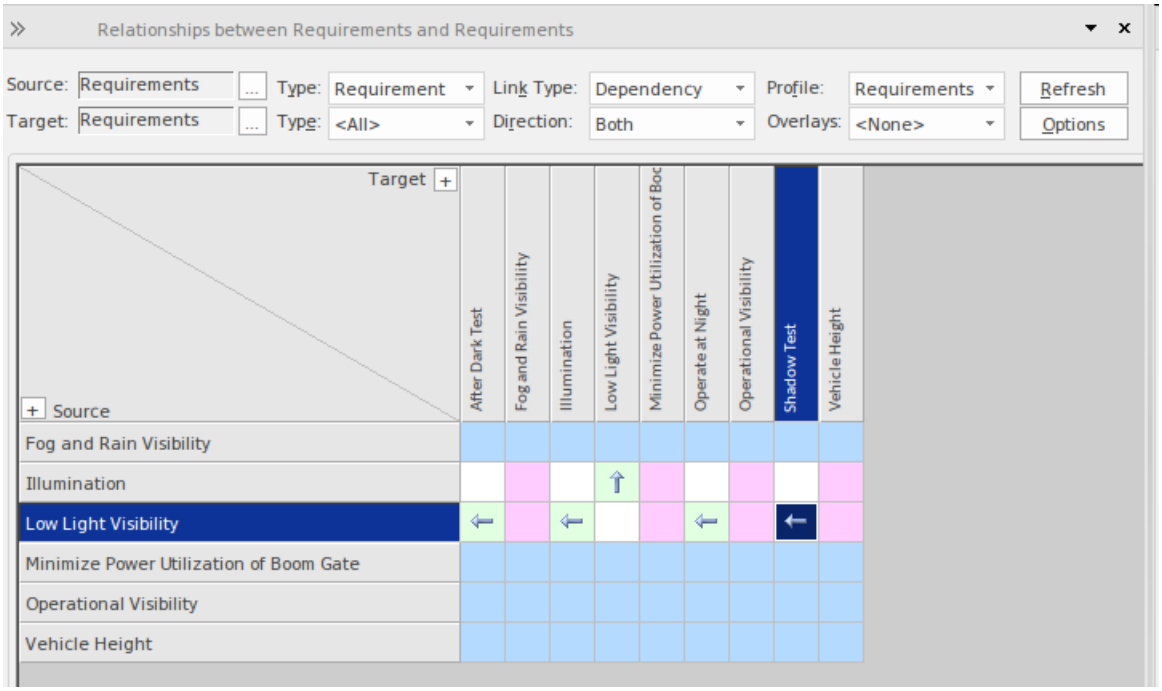
Any Package or diagram can also be visualized in a number of different ways, including the list view that provides an often welcomed view of the elements in a list, similar to a spreadsheet where the requirements are listed in rows and their properties are listed in columns.


Status

Status	Name	Type	Modified
Status: Approved			
Approved	OffRoadCapability	Requirement	30/01/2020
Approved	FuelCapacity	Requirement	30/01/2020
Status: Implemented			
Implemented	Braking	Requirement	30/01/2020
Status: Proposed			
Proposed	Power	Requirement	27/03/2018
Proposed	Range	Requirement	27/03/2018
Proposed	CargoCapacity	Requirement	5/03/2019
Proposed	RegenerativeBraking	Requirement	27/03/2018
Proposed	PowerSourceManagement	Requirement	27/03/2018

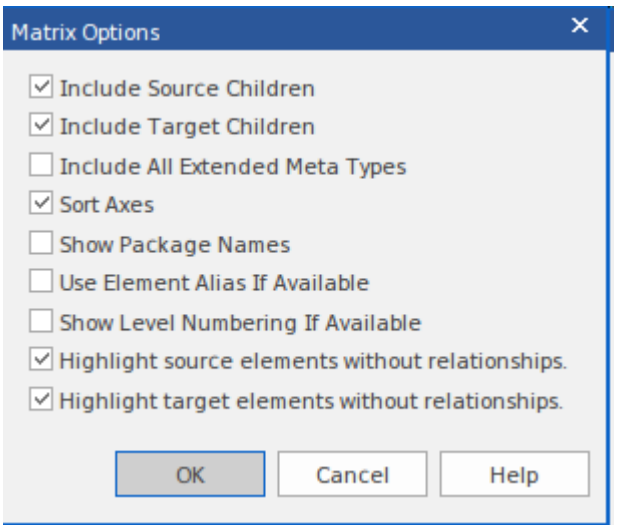
Relationship Matrix

The Relationship Matrix is a valuable tool for visualizing the connections between the elements in any two Packages, in an interface resembling a spreadsheet with rows and columns. The tool is particularly useful when used with Requirements, and allows an engineer to see how Requirements are related to other elements, including other Requirements.



The Relationship Matrix can be opened from the ribbon option 'Design > Package > Package/Matrix'. Select whether the current Package is the source Package, target Package, or both.

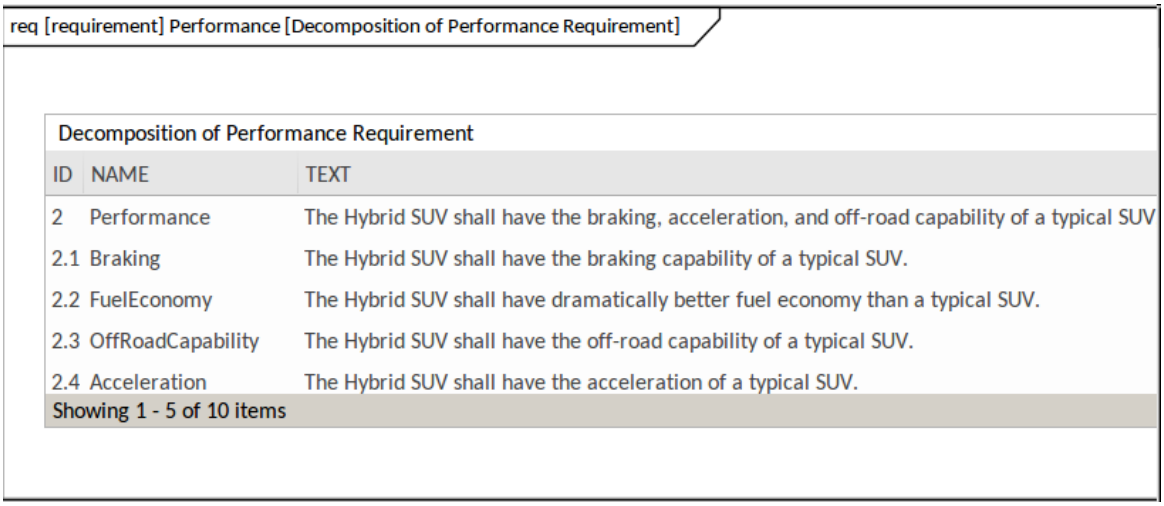
Where a relationship exists, an arrow icon is displayed in the cell at the intersection of the source and target elements, with the arrowhead showing the direction of the relationship. The matrix can also be configured to highlight the rows or columns that do not have any relationships, in a separate color. This and other options can be configured on the Options window, (click on the Options button in the Relationship Matrix header).



These options allow you to tailor the way that the matrix is displayed, including whether elements are sorted or their names prefixed with the Package name, and whether source and target element rows and columns without connections are highlighted. For more information see the Relationship Matrix Help topic.

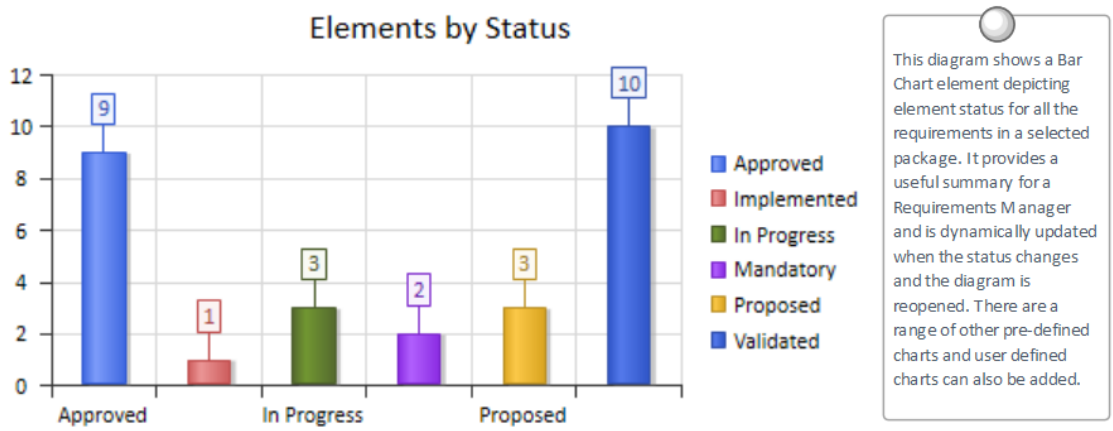
Requirements Tables

Requirements Tables are views resembling spreadsheets that can be created using a SQL statement to select Requirements (or any other elements) based on a select statement, which effectively filters out a particular group of Requirements. For example, a table could be used to display all Requirements related to the power subsystem that are approved and are of high priority or for the decomposition of performance requirements. Any number of tables can be created and they are refreshed dynamically as underlying properties are updated in the repository. This provides more flexibility than the List view because of the ability to select a group of Requirements from anywhere in the repository based on specified criteria.



Managing Requirements

This consists of the activities to maintain a set of requirements that represent an accord or agreement between the project team and the customer. It also involves ensuring that the requirements are acceptable to the design and implementation teams and that they are sufficient so that what they specify can be implemented into working business, software or hardware systems. Enterprise Architect is a sophisticated platform for managing requirements, and regardless of the domain, the size of the project or the method being followed, there are tools that will make it straightforward to manage even large repositories of requirements in complex projects.

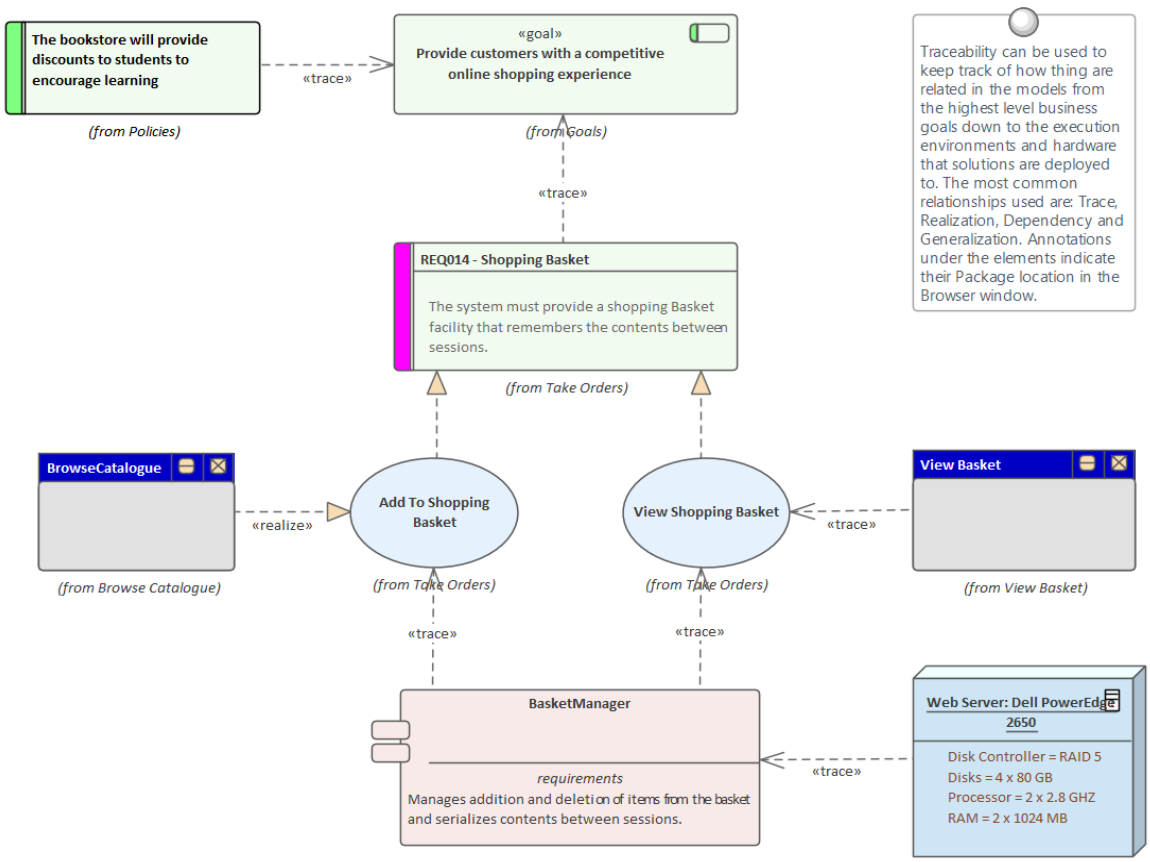


Tracing Requirements

Most Requirement processes mandate that Requirements are traced from high level concepts such as Business Drivers, Visions and Goals down to the parts of Components that implement them. For many projects this is an intractable problem because much of the information lives in a set of heterogeneous tools such as word processor documents, spreadsheets, diagram tools, corporate presentation tools and more. Some Project Managers attempt to solve the problem by creating a spreadsheet that acts as a register of all the disparate information but the management of this file takes up considerable project resources and the file is almost impossible to keep up to date. With Enterprise Architect there is the ability to model all of this project information in the one tool and to create easy-to-maintain and analyzable traces between all the elements, from the organization's mission statement right down to the level of programming code, if required.

Visualizing Traces in diagrams

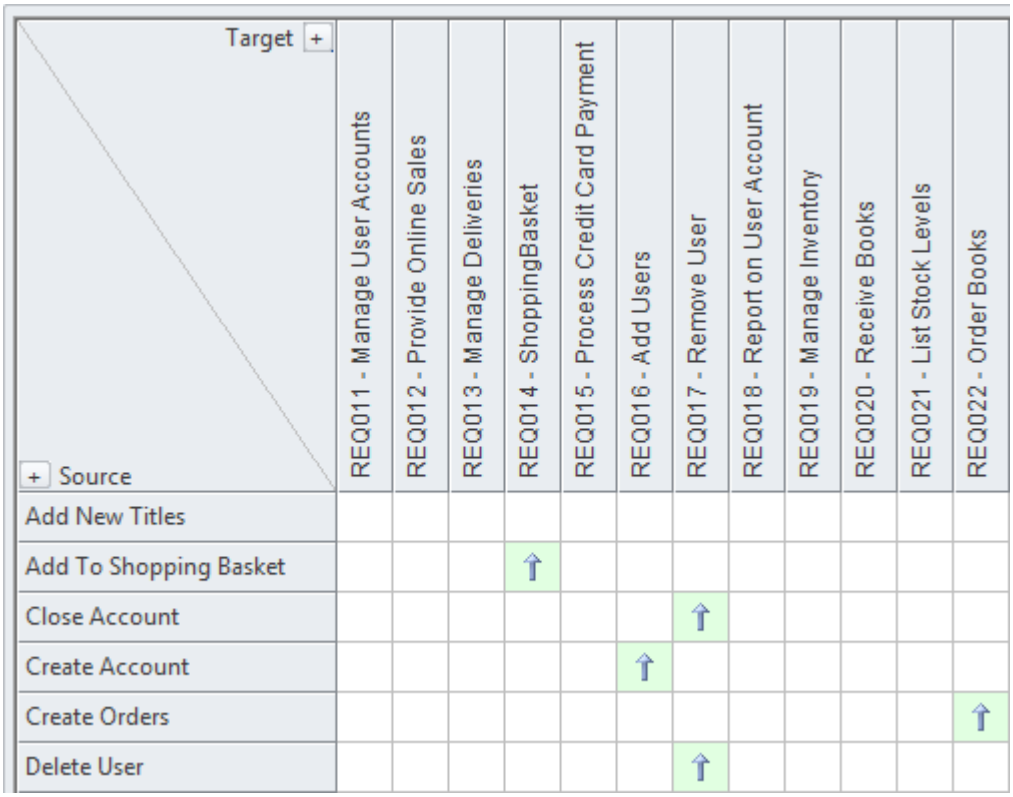
Regardless of whether you have entered the project's Requirements using a diagram, using a text-based tool such as the Specification Manager, or by importing them from another tool, viewing the requirement traces in a diagram gives an easily understood view of their relationships. The diagrams can be created easily by dragging and dropping elements from the Browser window, or automatically by using the 'Insert Related Elements' option. This function can be configured and used to draw a graph of elements to any depth, and can be restricted to selected types of element and connector. It is a handy productivity tool in a team environment, and even modelers with deep knowledge of the domain and the repository are surprised at the connections that are displayed in the diagrams.



Visualizing Traces using the Relationship Matrix

The Relationship Matrix provides an alternative way of visualizing the relationships between Requirements and other

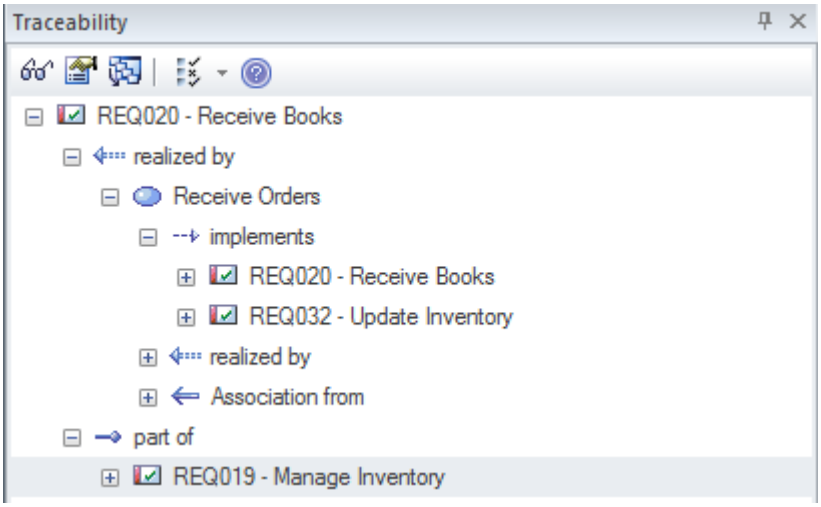
elements, or even between different levels or types of Requirement. It is quite common for some stakeholders to prefer a spreadsheet-like view of the Requirements and their relationships, and the Relationship Matrix provides an excellent way of presenting the relationships without resorting to a diagram. In Use Case driven requirements methods, Use Cases are said to realize one or more Requirements, and these relationships can be displayed visually in the Relationship Matrix. The list of Use Cases would appear on one axis of the matrix and the Requirements would be listed on the other axis. A marker in the intersection of a row and column would display if a relationship exists, indicating that a particular Use Case realizes a Requirement. Relationships between elements can be created or deleted using the Relationship Matrix, and the Matrix can be saved and reopened at any time or saved to a CSV file so it could be opened in a spreadsheet. Documentation can also be created that includes the Relationship Matrix, providing a useful communication tool for people who do not have access to the model.



Source \ Target	REQ011 - Manage User Accounts	REQ012 - Provide Online Sales	REQ013 - Manage Deliveries	REQ014 - ShoppingBasket	REQ015 - Process Credit Card Payment	REQ016 - Add Users	REQ017 - Remove User	REQ018 - Report on User Account	REQ019 - Manage Inventory	REQ020 - Receive Books	REQ021 - List Stock Levels	REQ022 - Order Books
Add New Titles												
Add To Shopping Basket				↑								
Close Account							↑					
Create Account						↑						
Create Orders												↑
Delete User							↑					

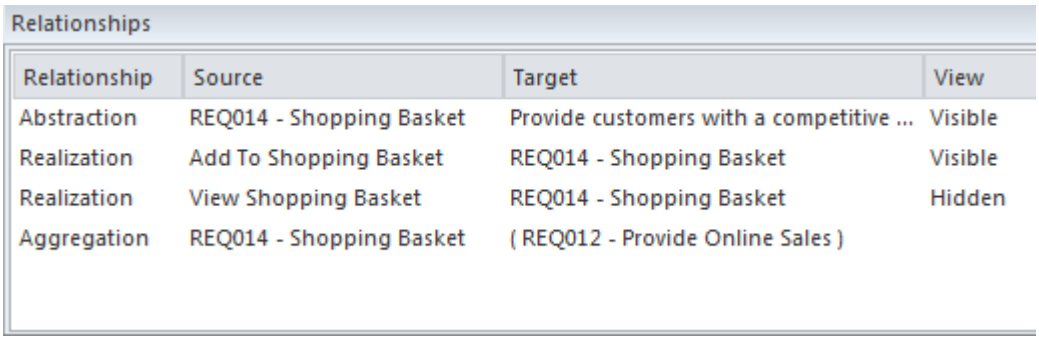
Visualizing Traces using the Traceability Window

While diagrams and the Relationship Matrix allow modelers to view traces between requirement elements it is possible that the creators of these views of the repository have deliberately omitted elements from the view. For example a diagram does not need to show all the requirements owned by a particular stakeholder. The Traceability window will, however, present a complete and unabridged view of the relationships between elements. The element relationships will be displayed regardless of the location of the elements in the Browser window.



Visualizing Traces using the Relationships Window

Modelers often choose to hide one or more relationships on a diagram for the purpose of making the diagram simpler to understand or to hide detail. The Relationships window is a useful window to have open as it will display all the relationships that exist between the elements in the diagram indicating whether they are visible or hidden in the diagram.



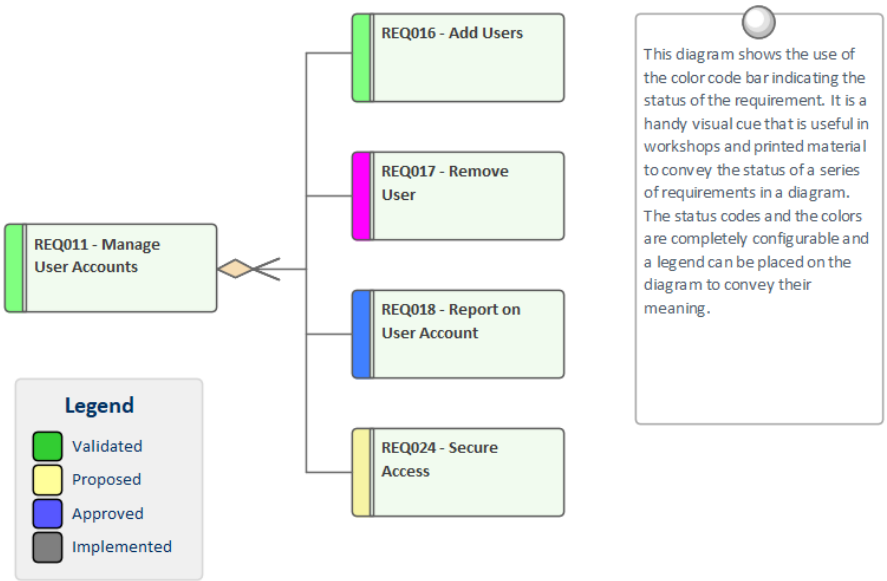
Relationship	Source	Target	View
Abstraction	REQ014 - Shopping Basket	Provide customers with a competitive ...	Visible
Realization	Add To Shopping Basket	REQ014 - Shopping Basket	Visible
Realization	View Shopping Basket	REQ014 - Shopping Basket	Hidden
Aggregation	REQ014 - Shopping Basket	(REQ012 - Provide Online Sales)	

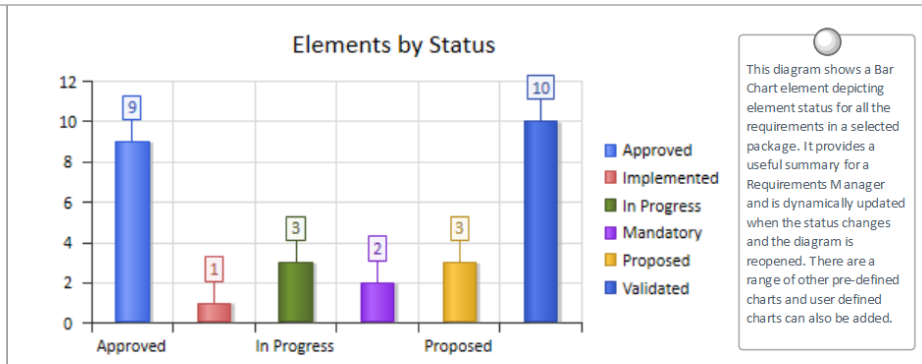
If relationships have been hidden in a diagram they can be made visible by selecting the 'Show Relationships' option on the 'Connectors' page of the 'Properties' dialog for the diagram.

Tracking Requirements

The status of a requirement is a fundamental indicator of where it is positioned in the requirement's development process. For example requirements that have a status of 'Proposed' indicate that they are not yet ready and available for development work to begin. Enterprise Architect has a variety of tools to allow status to be tracked, analyzed and managed, starting with the fact that each requirement can be assigned a status and the list of status codes are completely configurable. The status is conveniently displayed in list views of the requirements including when using the Specification Manager. There are also a set of pre-defined and extensible dashboard charts and graphs that can be used to get a compelling visual representation of the status and other properties of requirements.

Tools for tracking requirements

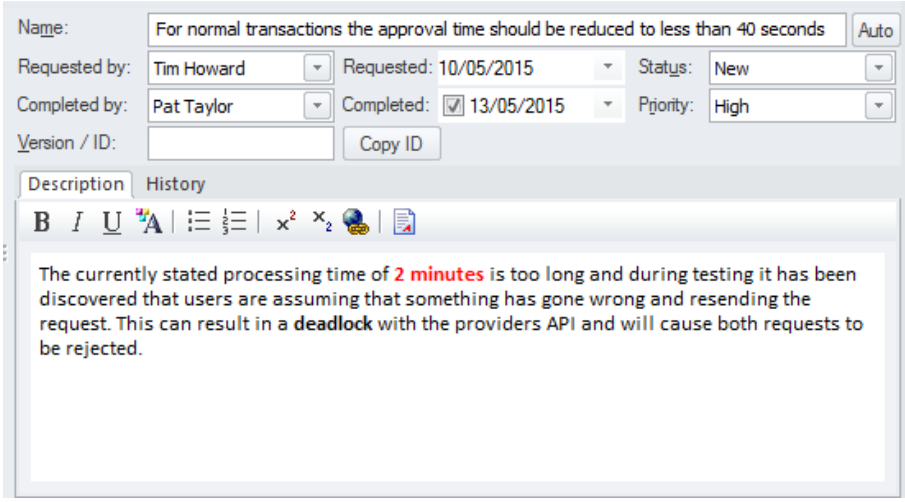
Tool	Description
Status Codes	Status codes are a controlled list of statuses that can be applied to any element including Requirements. Enterprise Architect comes with a pre-defined list of codes but the list can be configured and codes in the list can be changed and deleted and new codes can be added. The status of Requirements can be displayed in a diagram as a color coded band on the side of the element. 
Dashboards charts and graphs	Dashboard diagrams are an extended diagram type and allow high quality charts and graphs to be created to display repository information in a visually compelling way. Any number of diagrams and charts can be created and the data can be sourced from any level in the repository Package hierarchy. Enterprise Architect comes with a toolbox page of pre-configured charts and graphs, but new charts can be created based on any information in the repository.

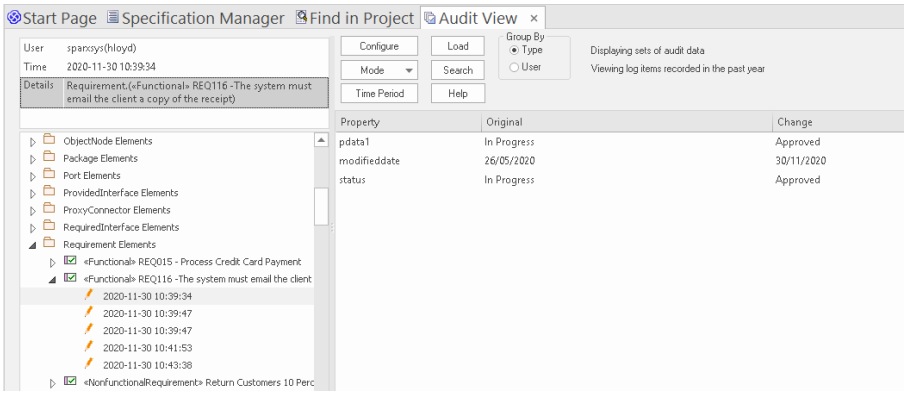
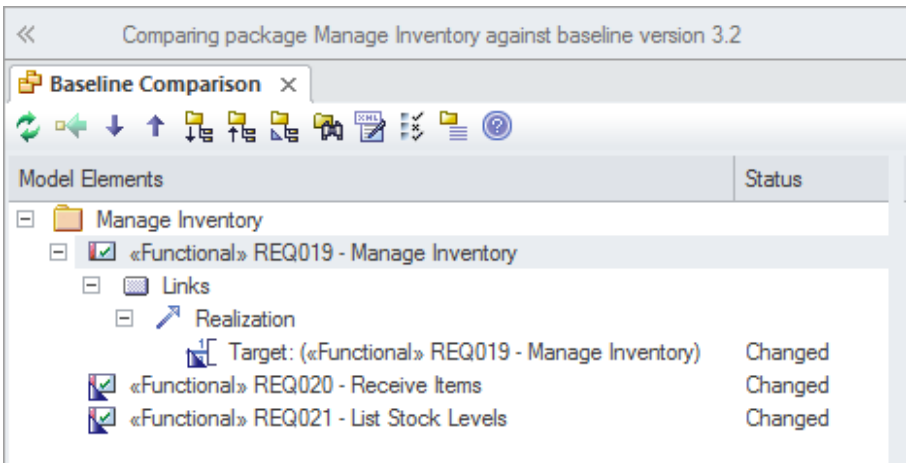

Elements by Status
12
10
8
6
4
2
0
9
1
3
2
3
10
Approved
In Progress
Proposed
Approved
Implemented
In Progress
Mandatory
Proposed
Validated
This diagram shows a Bar Chart element depicting element status for all the requirements in a selected package. It provides a useful summary for a Requirements Manager and is dynamically updated when the status changes and the diagram is reopened. There are a range of other pre-defined charts and user defined charts can also be added.

Managing Changing Requirements

It is inevitable that requirements will change during the specification and solution phases of a project, and most requirements management processes have some type of mechanisms for embracing these changes. Typically, a set of requirements will have been specified and groomed for the solution teams to implement; any subsequent changes are specified as Change Requests. Regardless of the rigor of the process being used, inadvertent changes will occur that need to be managed along with the Change Requests. Enterprise Architect is a sophisticated requirements management platform, with a range of tools to assist the requirements manager. Change Requests can be managed in the Maintenance window, which allows the requested change to be recorded and described, along with whoever requested it and when it was done and whoever completed the change. Inadvertent changes can be discovered and analyzed using a number of tool features, including Auditing, Baselines and Version Control; these tools have some overlapping features and can be used in isolation or together. The built-in Security system will also assist in preventing inadvertent changes to models, by allowing modelers to intentionally lock Packages and elements in the model.

Mechanisms for managing changing requirements

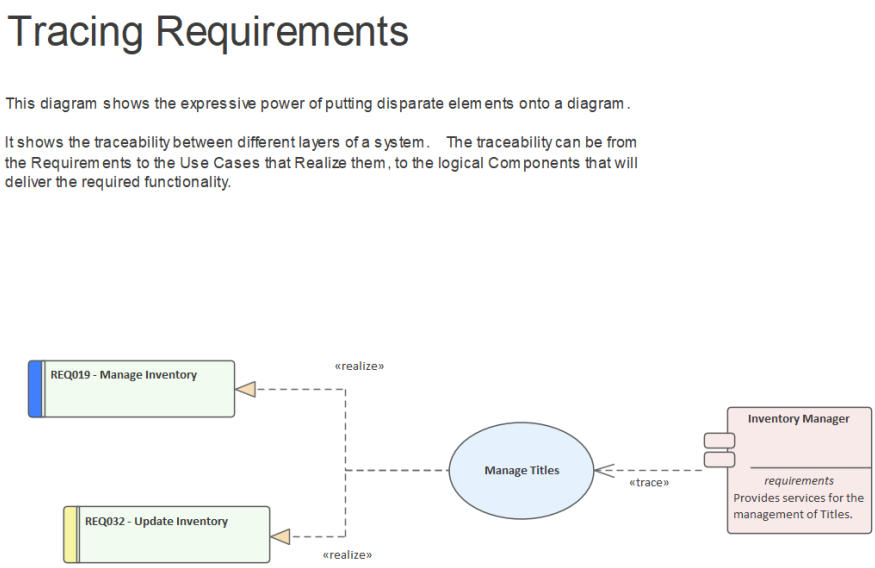
Mechanism	Description
Element change task and effort items	Changes to requirements can happen inadvertently but it is more common for there to be an intentional change in response to a wide variety of factors such as Stakeholders revising their needs, the business changing or a problem being poorly understood. Inadvertent changes can be picked up using a number of tools but deliberate changes can be assigned using the Change item, which can be recorded against each element. Once the impact of the change has been analyzed Tasks can be created to specify what needs to be done to implement the change and Effort can be assigned using the Requirements Effort item. 
Auditing	Auditing is a built-in tool that, when enabled, automatically records changes to the repository. It has a number of different modes and views, and can be configured to assist in the management of Requirements. It can track what was changed in the model, who made the change and when it was made, showing the before and after views. So if the text of a Requirement was updated or its status was changed, this would be recorded. Auditing functionality overlaps with the Baseline tool, but unlike the Baseline tool the changes are being recorded automatically and every discreet change is recorded. In contrast, the Baseline tool will only compare the current model to a Baseline regardless of how many intervening changes had been made. Auditing will not assist with the impact of the changes but just what changes have occurred. Once the changes have been established, tools such as the

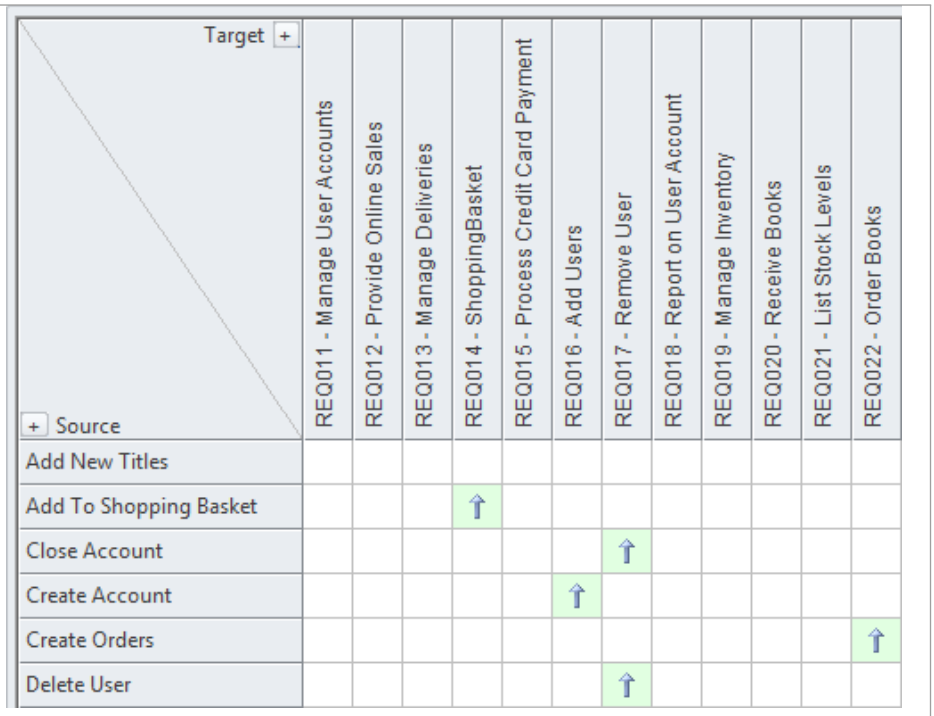
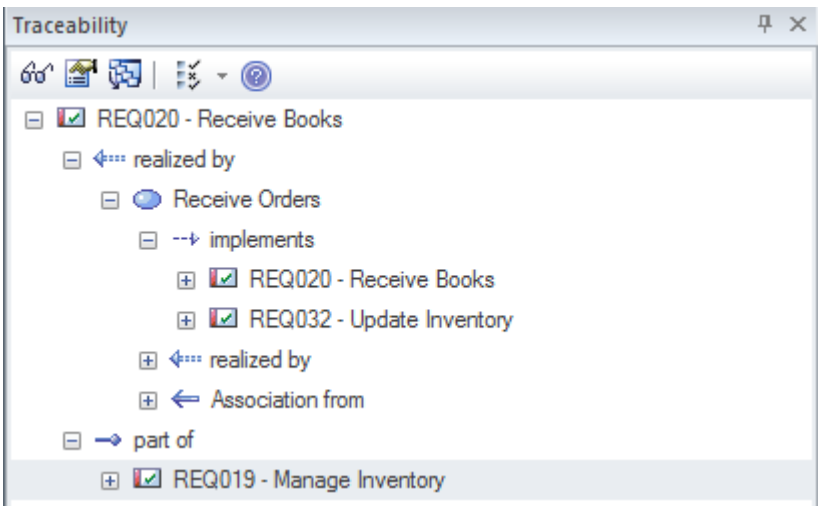
	Relationship Matrix can be used to determine the impact. 
Version Control	Version Control can be implemented in Enterprise Architect to manage changes and revisions to any Package including Requirements Packages. Once implemented changes to Requirements will be recorded and a requirements analyst will be able to view previous version and roll back to these versions if required. There is some overlap between this tool feature and Auditing and Baselines. The difference between this facility and Auditing is that Auditing simply records the changes but does not allow you to revert to a previous version. The difference between Version Control and Baselines is that a modeler must intentionally create a baseline whereas with Version Control the changes are being recorded automatically in the background. Also with Baselines the intervening changes are not recorded, just the difference between the current requirement and the one captured in the Baseline.
Baselines	Baselines provide a versatile mechanism for managing changes to Requirements. Any number of baselines can be created and when requirements are changed these changed requirements can be compared to one of the baselines. Baselines are typically created at important milestones in a project such as after a stakeholder meeting or before a development iteration is commenced. When differences are found and these changes were not intended or contravene project management practice the requirements from the baseline can be restored to the current model. Baselines will not help with assessing the impact of a change but once a change has been identified tools such as the Relationship Matrix and element traces can be used to determine the impact of a change. 

Impact Analysis of Changes

When the development of a system has started and requirements change there will be an impact of the change and the effect will need to be determined, understood and managed. Having traceability established both to up-process elements such as Stakeholders and Business Drivers and down-process elements such as Use Cases, Components, Test Cases and source code operations is critical to determining the impact of the change. Enterprise Architect has a number of facilities that can assist with this including the ability to visualize traces in diagrams, a Relationship Matrix, a Traceability window, element Change, Task and Effort items that can be used to record impact and what is required to implement it.

Tools to record and analyze the impact of change

Tool	Description
Analysis using requirement traces	The ability to visualize requirements and the way they are connected to other elements is a practical tool for analyzing the impact of change. Requirements often form a hierarchy and when one requirement is affected it typically has a ripple effect to the requirement's children and being able to visualize this in a diagram or in a hierarch is very useful. Requirements are also typically traced to up-process and down process elements and a diagram provides a way of viewing and analyzing these connections. The Insert Related Elements function can discover these connections and automatically draw and layout a diagram allowing the modeler to spend their time analyzing the impact. 
Analysis using a relationship matrix	The Relationship Matrix can be used to visualize the requirements and their connections by placing the Requirement on one axis of the matrix and the connected elements on the other. This method is very useful in workshops when working with people who might not be familiar with modeling languages such as UML or who work better with spreadsheet types of view. Any number of matrices can be created and their specification can be stored so they can easily be recalled.

	
Analysis using the traceability window	The Traceability window is a handy window that shows the hierarchy of elements in the Repository. It is particularly useful because it unconditionally shows how elements are related to each other. Other views of the repository could be configured just to display particular elements for the purpose of communicating an idea whereas the Traceability window will display all relationship that an element participates in which makes it particularly useful for analyzing the impact of change. 

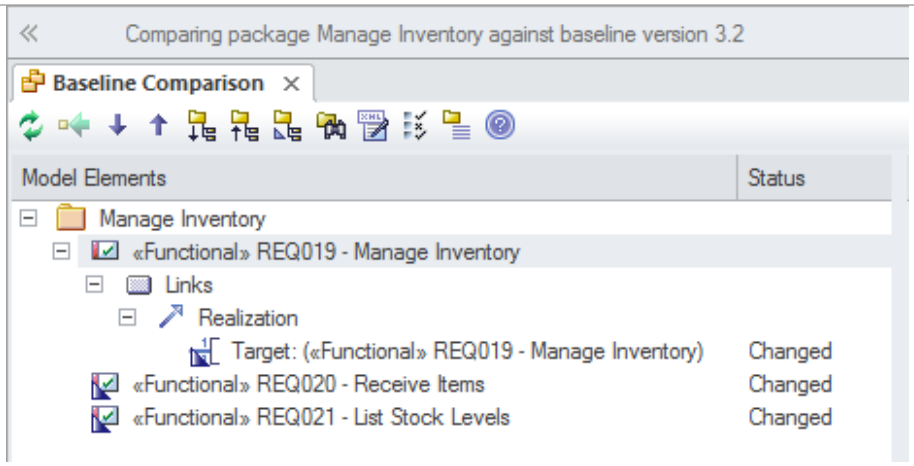
Requirement Volatility

There are ever increasing market place pressures to release products and systems as early as possible, putting stress on project teams to develop, test and deploy products in shorter and shorter periods of time. The requirements processes have changed significantly in recent years to ensure that stable, correct and well-articulated specifications are provided to architects, designers and developers when they need them. There has been a move to iterative and incremental processes and this necessitates providing a set of stable requirements for every iteration. The churning of requirements is often an indicator that a problem is not clearly understood, that stakeholders have not been compromised and there are unresolved political issues, the scope is not defined or the business itself is in fluctuation. Enterprise Architect has a number of mechanisms that can be used to assist with this problem. Enterprise Architect does not have a built-in property for requirement volatility (stability) but using the general purpose UML extension mechanism of Tagged Values a tag could be created to record this property.

Note: Internal requirements do have a stability property but external requirements do not.

Mechanisms for managing requirement volatitlity

Mechanism	Description
Volatility as a Tagged Value	Enterprise Architect provides a series of properties for Requirements, but additional properties can be created to record other properties such as a Requirement's volatility or the source of the Requirement. This is achieved using the UML Tagged Value mechanism, which allows any element including Requirements to have one or more tags applied, representing some property that can be assigned a value. Enterprise Architect has extended this mechanism and allows the modeler to create a list of values that can be chosen from a drop down list using the Predefined Structured Tagged Values. This allows a team to define their own list of volatility values, such as extreme, high, medium low, minimal. 
Using Baselines	The Baseline facility is an effective tool that enables a user to take a snapshot of a model or more typically a model fragment and then as the model is developed to compare the new version of the model to the baseline thus identifying anything that has changed since the baseline was taken. Baselines have general applicability but are particularly useful with requirements management where requirements are often said to be signed-off or frozen and any alterations to them must be registered as a change. The Baseline tool has a Compare utility that conveniently lists changes between the current model and the baseline.

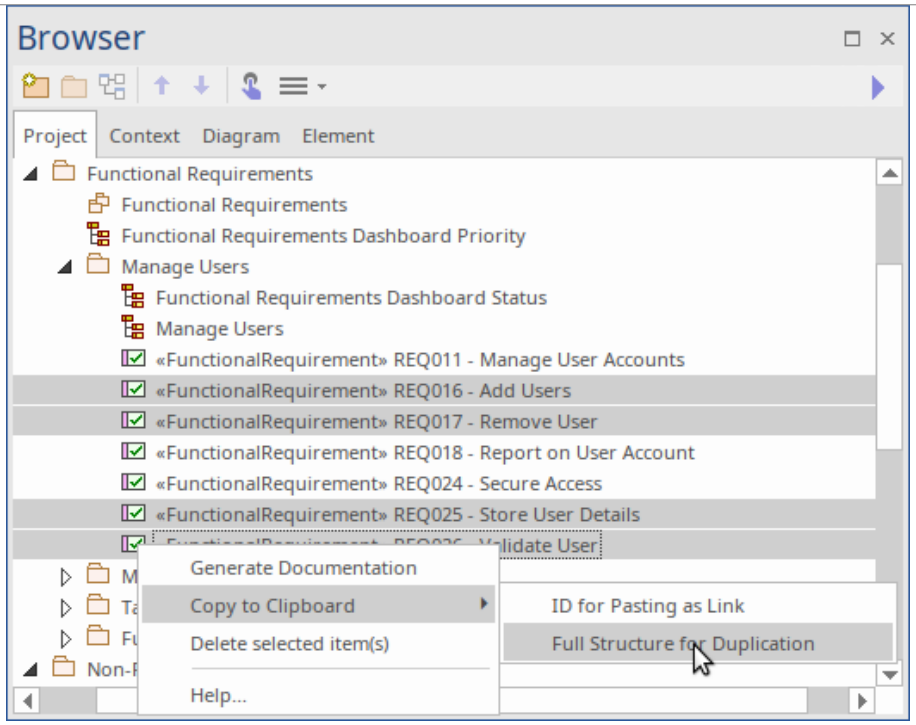
	
Searches for churning requirements	Enterprise Architect has a sophisticated search facility that allows a user to search across either a selected Package or the entire repository, to locate elements that meet fine-grained criteria. This can be used to locate requirements that have not changed by searching for a change in the modification date before a specified date, thus providing a list of stable requirements. Alternatively, if volatility has been set using a Tagged Value, all elements with a specified volatility could be located. The search facility returns a list of elements that can be located in the Browser window; the search can be used as the basis of a Model View to be used to view either volatile or non-volatile requirements.

Requirement Reuse

The concept of reusing artifacts of a system development process has been written about in many papers and text books but has traditionally been confined to software components. In more recent years the notion of reusing specifications, including requirements, has started to get traction. The reuse is particularly important where organizations create a family of products with similar features, or where there is a community of users within an industry or domain. Other types of requirement such as security and regulatory requirements will typically apply to a number of projects. Business Rules and Stakeholders Concerns will also typically apply across many projects and are best catalogued outside individual project structures. Enterprise Architect provides a number of sophisticated mechanisms for managing the reuse of elements across projects, including structuring the repository for reuse, importing requirements from other sources, and a Reusable Asset Service.

Mechanism for requirements reuse

Mechanism	Description
Structuring the repository for requirements reuse	When you set up a repository, you have the choice of structuring it for a single project or for multiple projects, which in turn could be organized by a number of programs of work. Enterprise Architect gives the modeler complete control on how the repository is structured, allowing Packages to be set up above the level of projects where some requirements such as Business, Regulatory and Architectural Requirements can be added.
Creating a base model	When you create a new repository in Enterprise Architect, you have the option of creating a blank model using the Model Builder to help set up a repository structure, or you can use a base model as a template for the new model. The base model is a good place to store reusable assets such as Business, Regulatory and Architectural Requirements, and Policies and Business Rules.
Importing requirements from other models	It is quite common to have a number of Enterprise Architect Repositories in an organization and it is very easy to copy and reuse Requirements (or any other elements) from one model in another. This can be achieved by simply copying a selection of Requirements or an entire Package from one repository to another, or even from one project to another in the same repository. Enterprise Architect works in the same way as any other Windows program, simply copying the selection to the clipboard and then allowing it to be pasted in another location in the same model or in another open repository.

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>Using the Reusable Asset Service</td><td>The Reusable Asset Service (RAS) is particularly useful for distributed teams and provides a simple and convenient mechanism for modelers to distribute or download reusable model structures and elements such as Requirements through a shared repository, accessible via a Pro Cloud Server connection. Enterprise or organizational level Requirements could be stored in the RAS and different teams could incorporate them into their models, governance of the assets would typically be managed by the owner of the asset (register) at the Reusable Asset Service level.</td></tr>
</table>

Requirement Relationships

Enterprise Architect supports all the SysML Requirement relationships, which can be visualized in a number of different locations within the user interface, providing a flexible way of working with these important connectors. The relationships between elements (including Requirements) are not visible in the Browser window, as this would clutter the elements when there are more effective ways of viewing the connections.

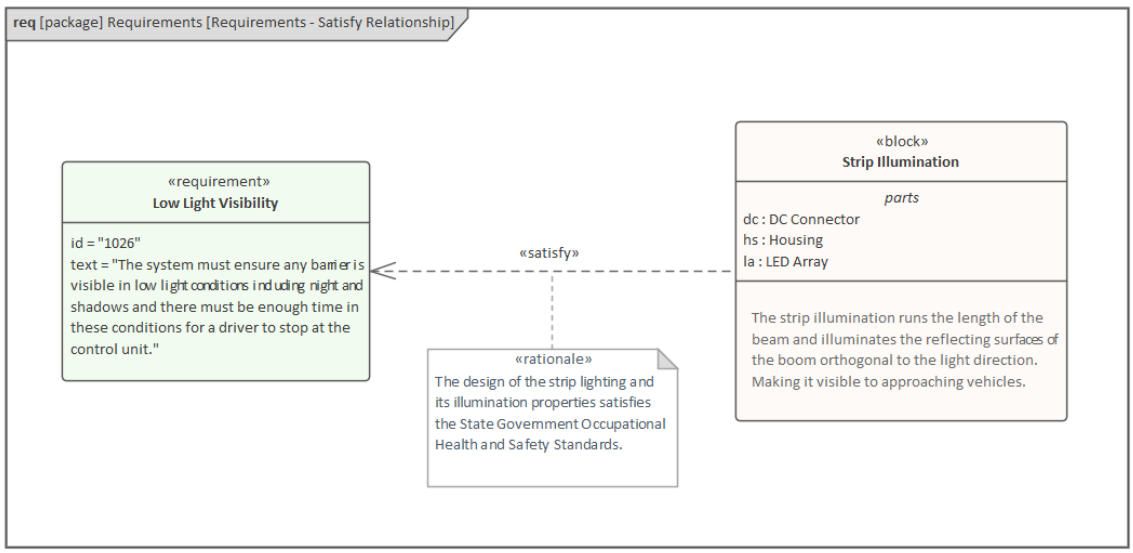
The Relationships window is useful to have docked while viewing elements, either in the Browser window or in a diagram. As an element is selected in the Browser window or in a diagram, the context changes and the Relationships window will display just the relationships that exist between the selected element and other elements in the model, including other Requirements.

The relationships between Requirements and other elements, including other Requirements, can be visualized in any diagram including Requirements diagrams, in three different ways:

- A *connector* between two elements
- A *compartment* in the Requirement element
- A *callout* notation in the form of a note attached to either a Requirement or another model element

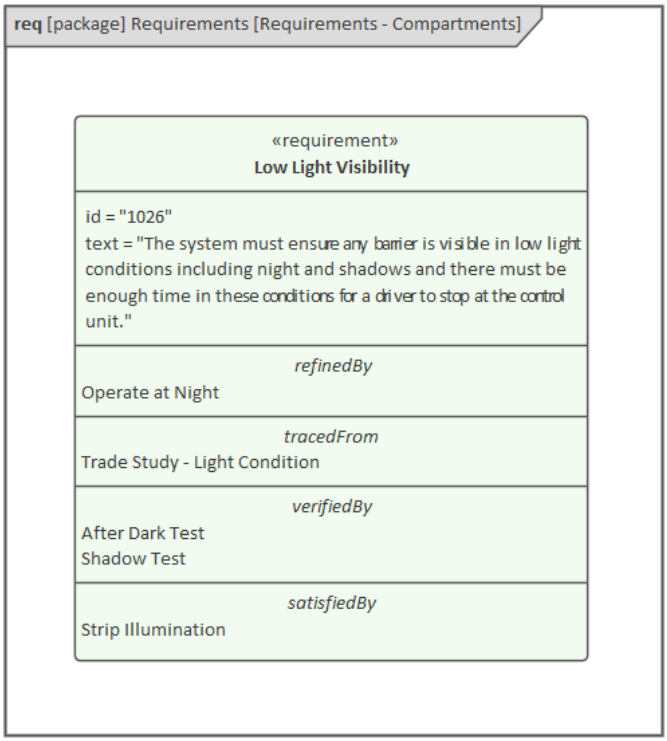
All three diagram notations have their purpose, and provide flexibility for the modeler to choose the appropriate representation for a particular purpose and audience.

The relationship drawn between two elements visible in a diagram is the most common way to visualize Requirement relationships; the dashed line is drawn from the client (the dependent element) to the supplier (the providing element). So in this example the 'Strip Illumination' Block is the client and it depends on the 'Low Light Visibility' Requirement, so the arrow points from the Block (client) to the Requirement (supplier).

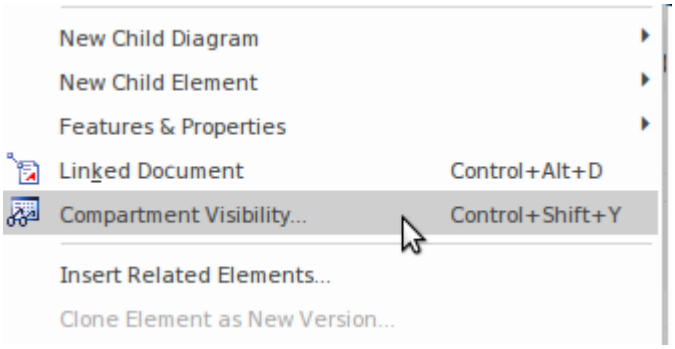


Notice also in this example that the modeler has chosen to display the Parts compartment, showing the Parts that make up the Block, and the Notes compartment that describes the Block. A rationale has also been added to qualify the 'Satisfy' relationship, and to provide an explanation as to why the Block was chosen in the context of standards.

Compartments can be used to display the relationships that a Requirement participates in, which is a compact and useful way of visualizing the Requirement relationships without the need to include the related elements in the diagram.

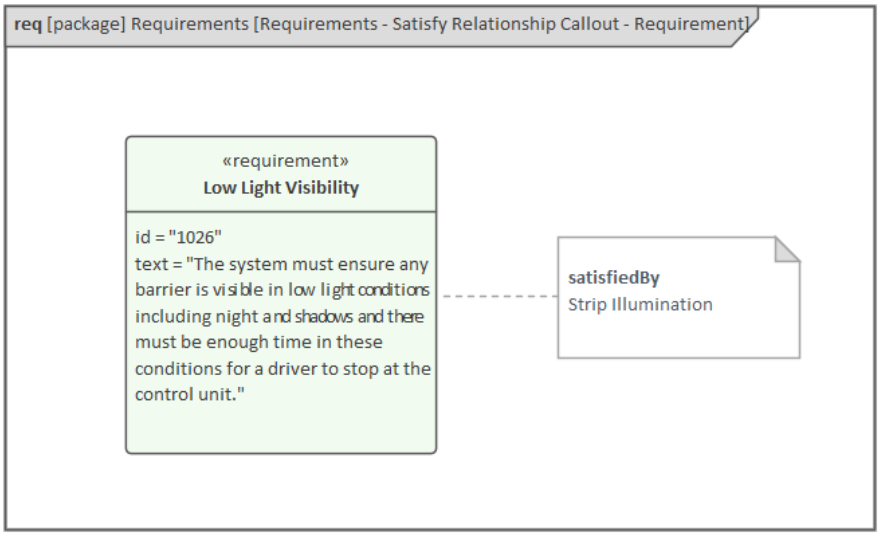


The list of visible compartments can be configured for each diagram element or for the entire diagram, providing fine granular control on how the relationships are visualized.

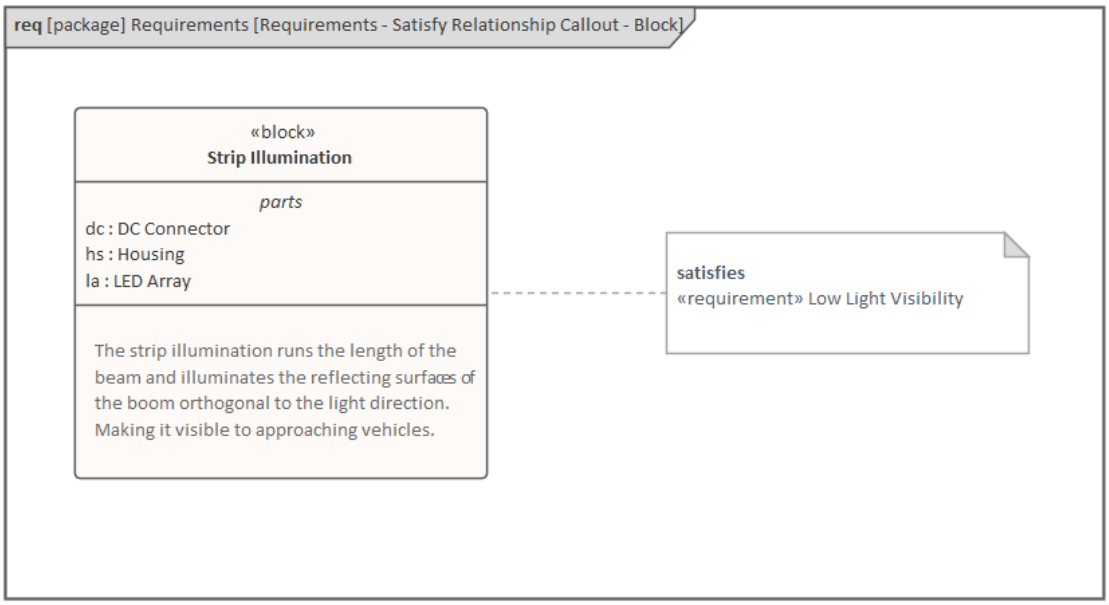


Alternatively, a *Callout* notation can be used to display the relationship in a note attached to either the Requirement element or the dependent element that the relationship relates to. This notation is particularly useful when elements appear in diagrams in which either the connector or compartment displays are not suitable, such as an Internal Block diagram, Sequence diagram or Use Case diagram, or in other diagrams as a modeler sees fit. The Requirement relationships are binary, meaning they have two ends: a supplier and a client. This means that the Callout can be attached to either a Requirement or the related model element which, depending on the relationship, could be a Block, Test Case, Use Case or other model element, including another Requirement.

In this diagram the modeler's focus is on the Requirement and the Block element is listed in the note stereotyped as <<satisfiedBy>>.



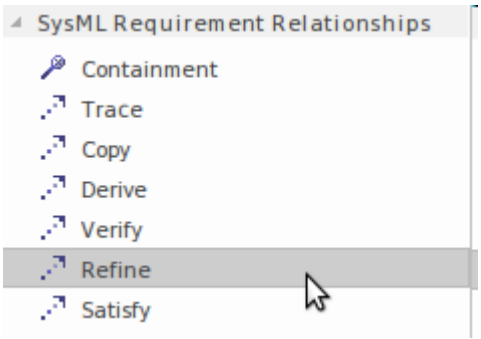
In this diagram, the modeler's focus has switched to a Block and the Requirement element is listed in the note stereotyped as <<satisfies>>.



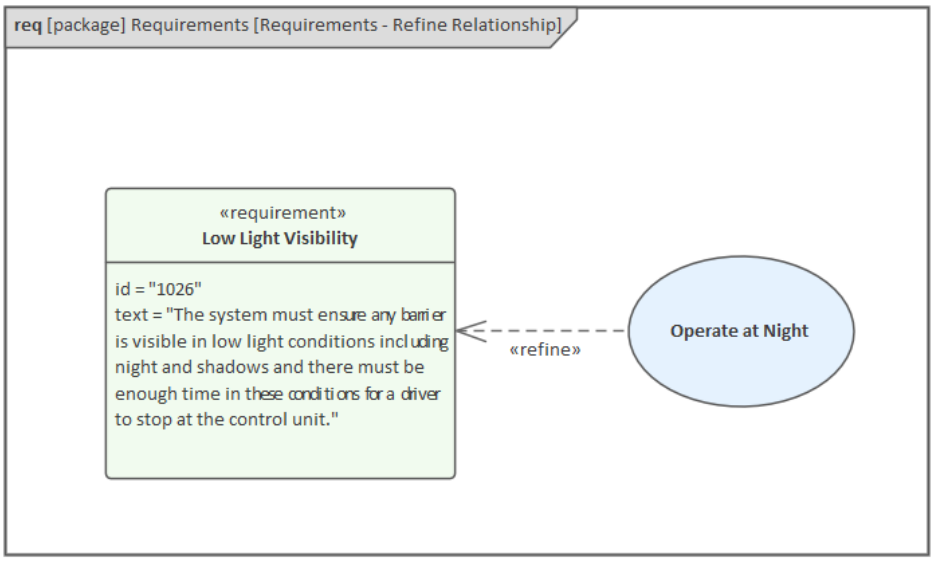
The next section details the Requirement relationships, providing an example of each relationship.

Adding Refinement to a Requirement

The *Refine* relationship is a relationship between a Requirement and another model element that adds refinement or additional information that helps clarify the requirement so that its meaning is more apparent. The Refine relationship is available from the 'Relationships' page of the SysML Requirements Toolbox.



The Refine relationship can be drawn between a Requirement and any model element such as a Use Case, a StateMachine, or an Activity. The choice of model element will depend on the information expressed in the requirement and the discretion of the modeler or engineer.

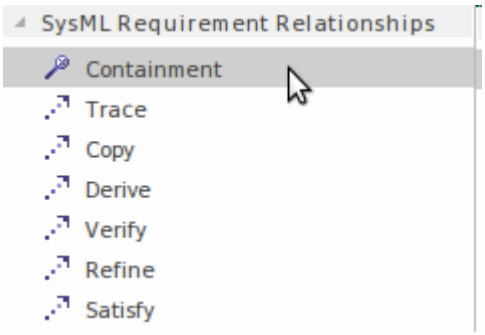


In this example the Use Case does not extend or embellish the requirement but rather adds detail in the form of descriptions and Scenarios that will make the meaning of the requirement easier to understand.

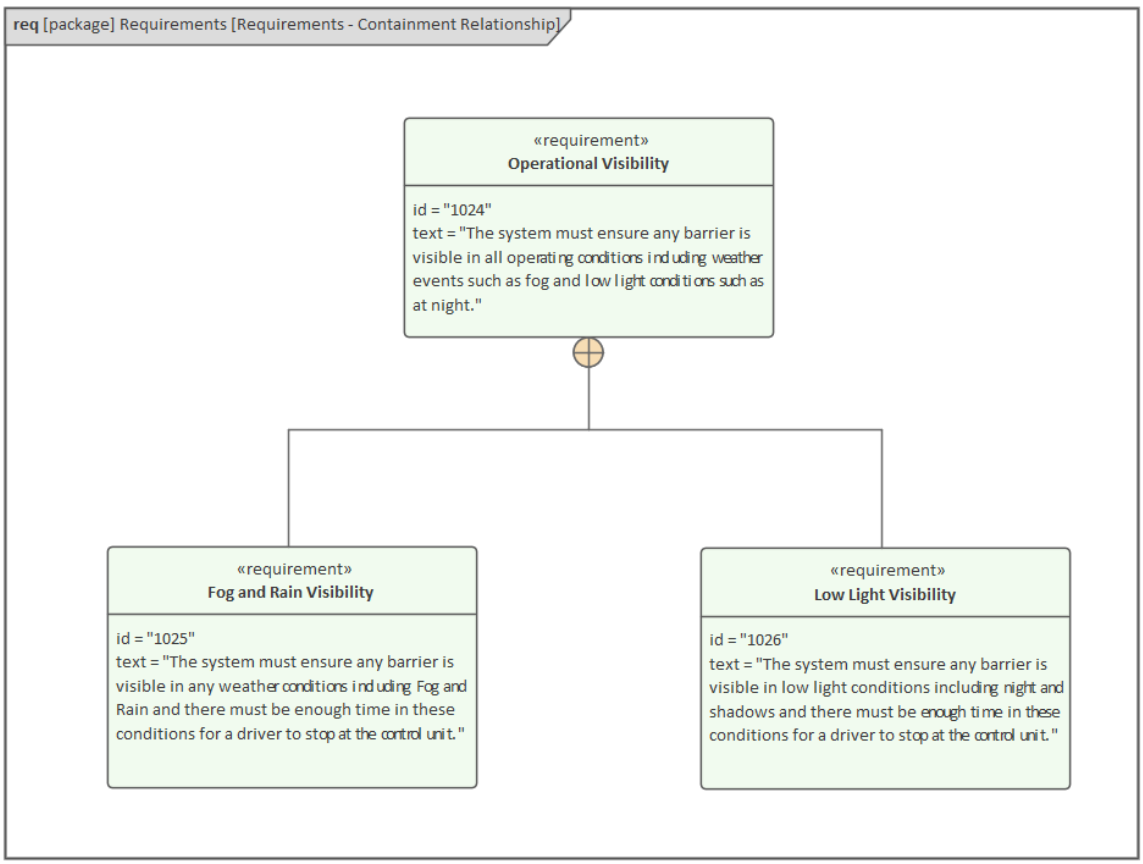
Containment Relationship

The *Containment* relationship's name well describes its purpose - it is used to show that one or more Requirements are contained in, or are grouped by, another higher-level Requirement. It is a fundamental and highly used relationship when modeling requirements for any system of even moderate complexity. A large system could have thousands, if not tens of thousands, of requirements, and these are best grouped together in hierarchies. An alternative to the use of the Containment relationship is to group Requirements using Packages. This method works when there are just two levels in the hierarchy or when you group Requirements by type - such as Stakeholder or Physical - but it has limitations when used more extensively.

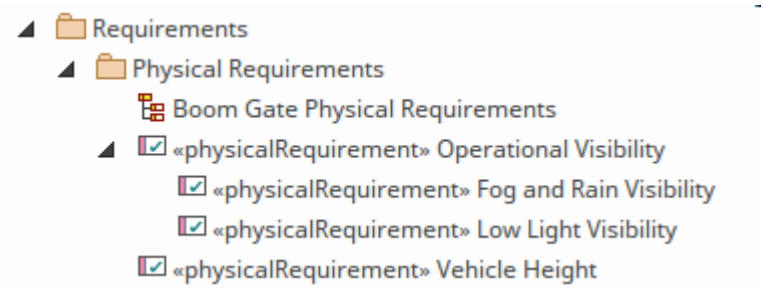
The Containment relationship is available from relationships page of the SysML Requirements Toolbox.



This diagram shows the use of the Containment relationship to show two lower-level Requirements that are 'contained' by a higher-level Requirement.



The containment of Requirements can also be visualized in the Browser window, where containment is represented by elements being nested or, more formally, the contained elements are children of another Requirement. This leveling is possible for any elements in the repository, but has special meaning with Requirements. This image shows the same Requirements as in the previous diagram, but in the Browser window.

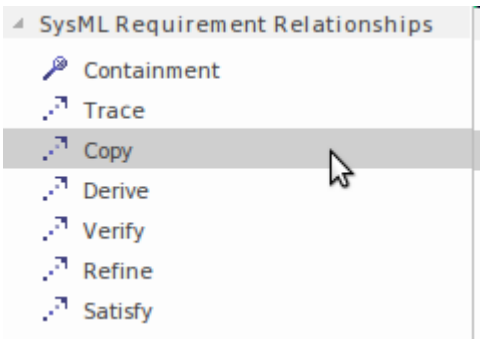


Note that nesting the Requirements in the Browser window does not create Containment relationships between Requirements. In fact, it is possible that the two different methods could be out-of-synch with each other because they are independent mechanisms.

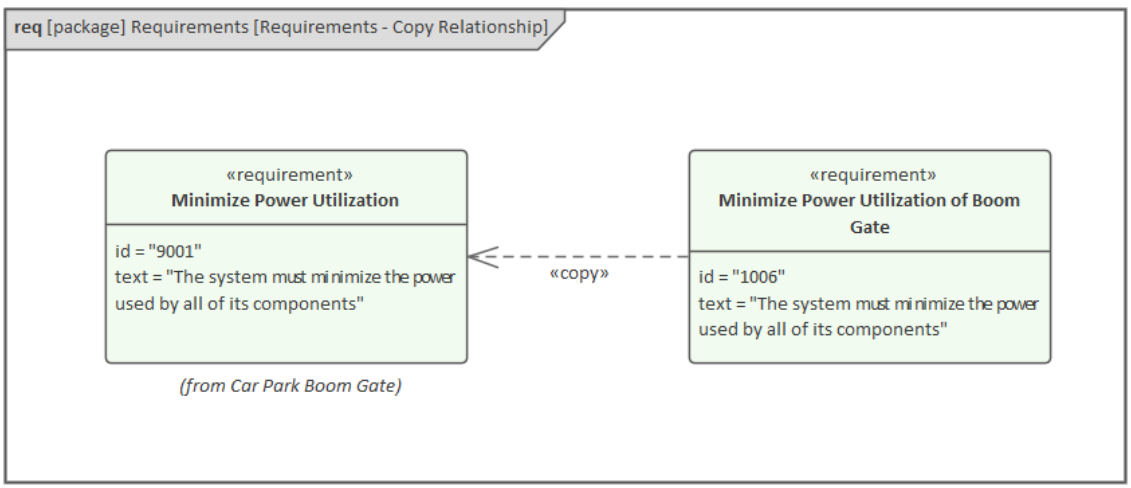
Copying Existing Requirements

The *Copy* relationship is a relationship between two Requirements; it is used to show that one Requirement is a copy of another. The relationship is a type of Dependency and is represented by a dashed line with the keyword <<copy>>, with an open arrow head pointing from the Copied (Client) to the Base Requirement (Supplier). Given that Requirements elicitation, and management are expensive and time consuming activities, and that many projects often have an overlap of interests, it is useful to re-use Requirements; the Copy relationship provides a mechanism to do this. The base Requirement is typically stored in another project's namespace but it is considered good practice to move the common (base) Requirements to a namespace that sits above the level of individual projects.

The Copy relationship is available from the Relationships page of the SysML Requirements Toolbox.



This diagram depicts a power utilization Requirement that has been copied for re-use in a number of projects.

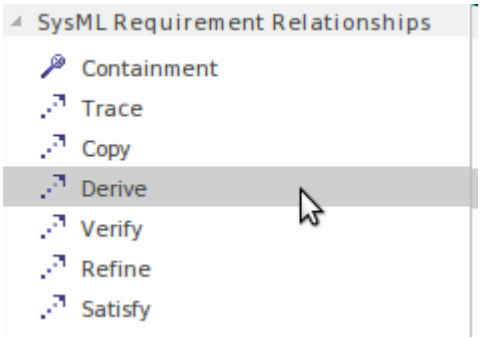


When the Copy relationship is used the new Requirements are assigned a new id, but the text of the new Requirement will be a read-only copy of the base Requirement.

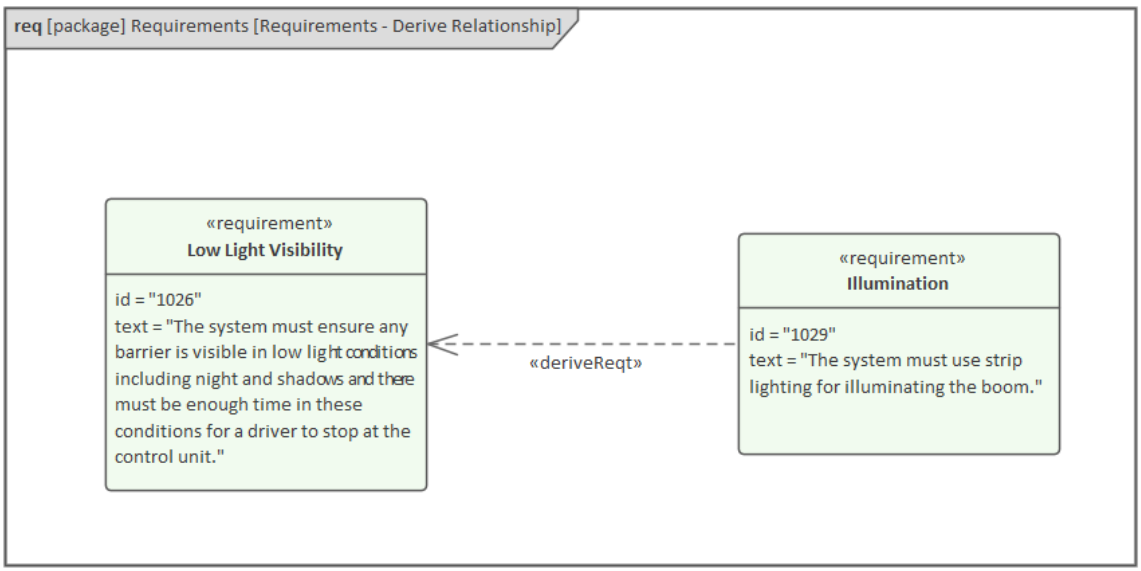
Deriving a Requirement from Another

The *Derive* relationship is a relationship between two Requirements, used to describe the fact that one Requirement is based on or is an extension or derivation of another Requirement.

The Derive relationship is available from relationships page of the SysML Requirements Toolbox.



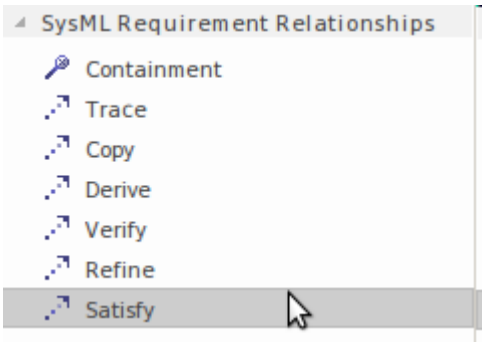
The two Requirements are typically at different levels of abstraction or resolution. A Requirement in a low level specification might have a Derive relationship to a Requirement in a higher level specification. The lower level Requirement is typically derived from the higher level Requirement as a result of investigation, elaboration or analysis. The important aspect of this relationship is that if the Requirement at the arrow end of the relationship is changed, it is highly likely that the derived Requirement will need to be reanalyzed.



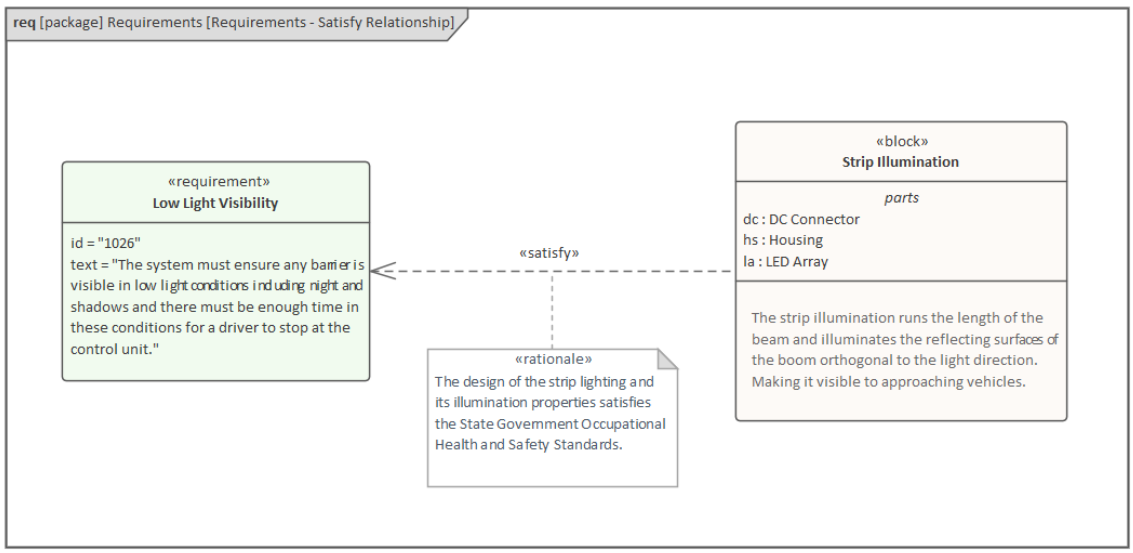
Ensuring a Requirement is Satisfied

The *Satisfy* relationship is used to show that one or more model elements in the architecture or design fulfills the notion expressed in the Requirement. It is an important connection or bridge between what might be described as the problem or opportunity and the architecture, design and - when verified - the implementation.

The Satisfy relationship is available from 'Relationships' page of the SysML Requirements Toolbox.

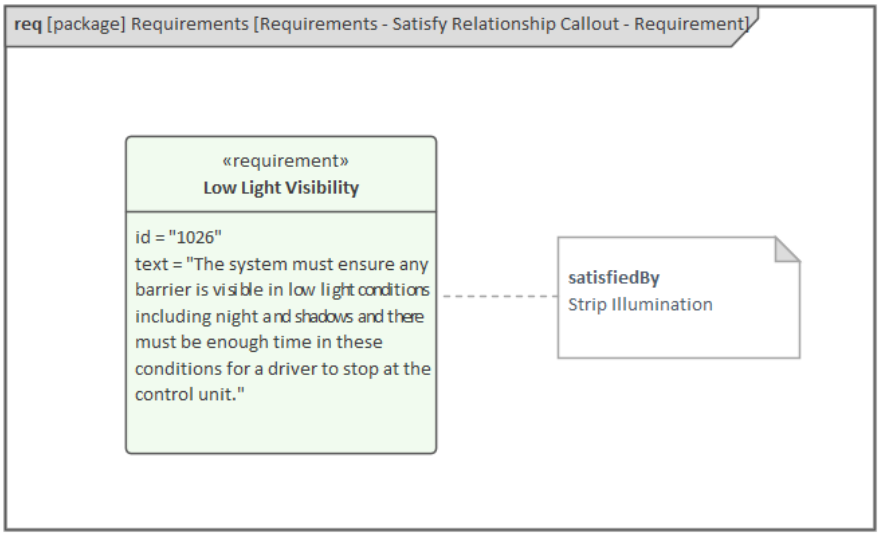


The relationship is a type of Dependency and is represented by a dashed line with the keyword <<satisfy>>, with an open arrow head pointing from the design element (Client) to the Requirement (Supplier).

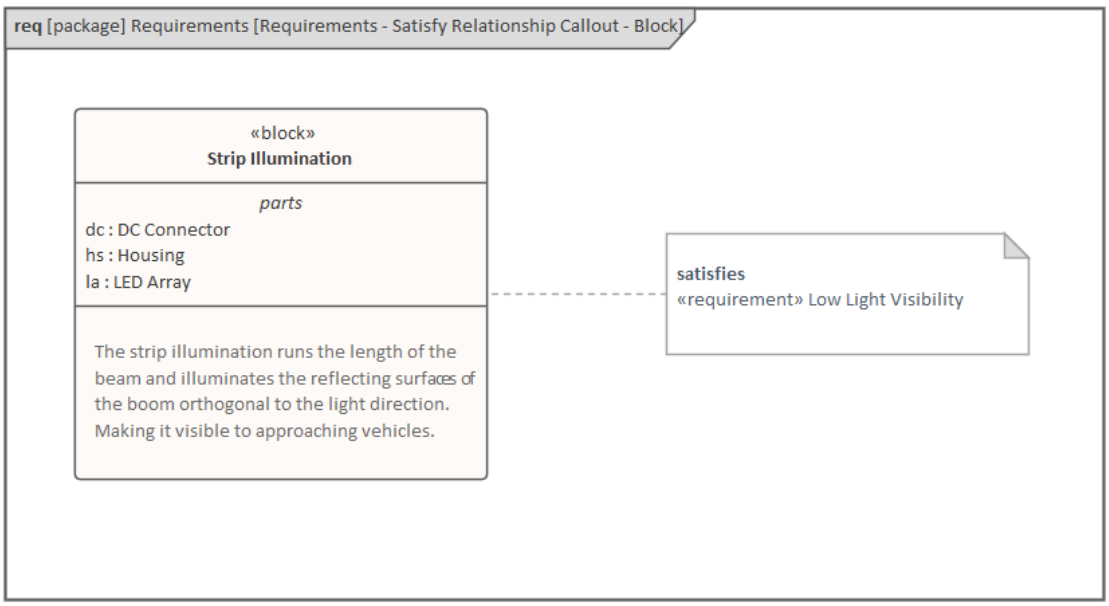


In this diagram it can be seen that the Block 'Strip Illumination' satisfies the Requirement that speaks of the visibility of the boom in low light conditions. There is also a rationale added that describes the compliance with respect to State Government regulations.

In this diagram the Satisfy relationship has been displayed using Callout notation.



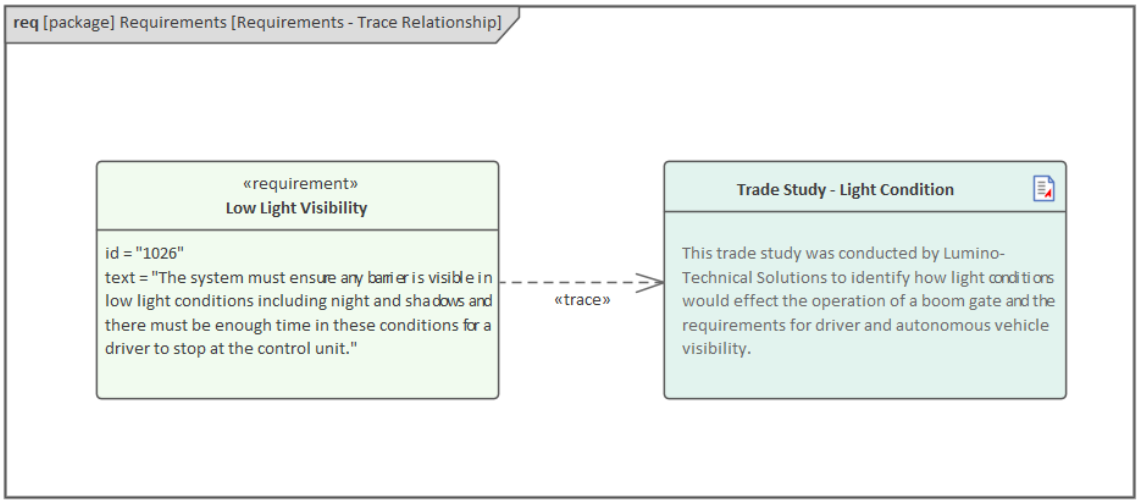
In the next diagram callout notation has been used but this time the Block is referenced in the callout.



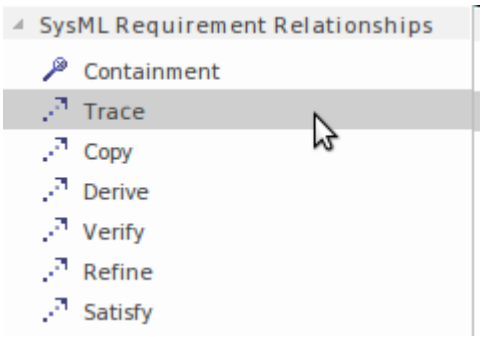
Traceability to Model Elements

The *Trace* relationship is a general purpose, widely-used relationship that connects a Requirement to any other modeling element. The relationship is expressed as a dashed line with the keyword <<trace>>, which indicates the meaning; the arrow head points to an up-process element (one that was created earlier in the process).

In this diagram the modeler wants to show that a Requirement has a relationship to a trade study represented by a Document Artifact. The document might have been written in Enterprise Architect or it might be a linked external document.



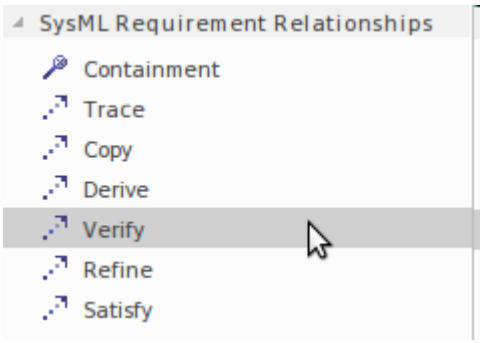
The Trace relationship acts as a catch-all, and is useful when a modeler wants to show that a Requirement has a semantic relationship to an up-process element, and none of the other relationships are appropriate.



Verify Relationship

The *Verify* relationship is used to indicate that a Requirement has been fulfilled. The relationship is a type of Dependency and is represented by a dashed line with the keyword <<verify>>, with an open arrow head pointing from the Test Case (Client) to the Requirement (Supplier).

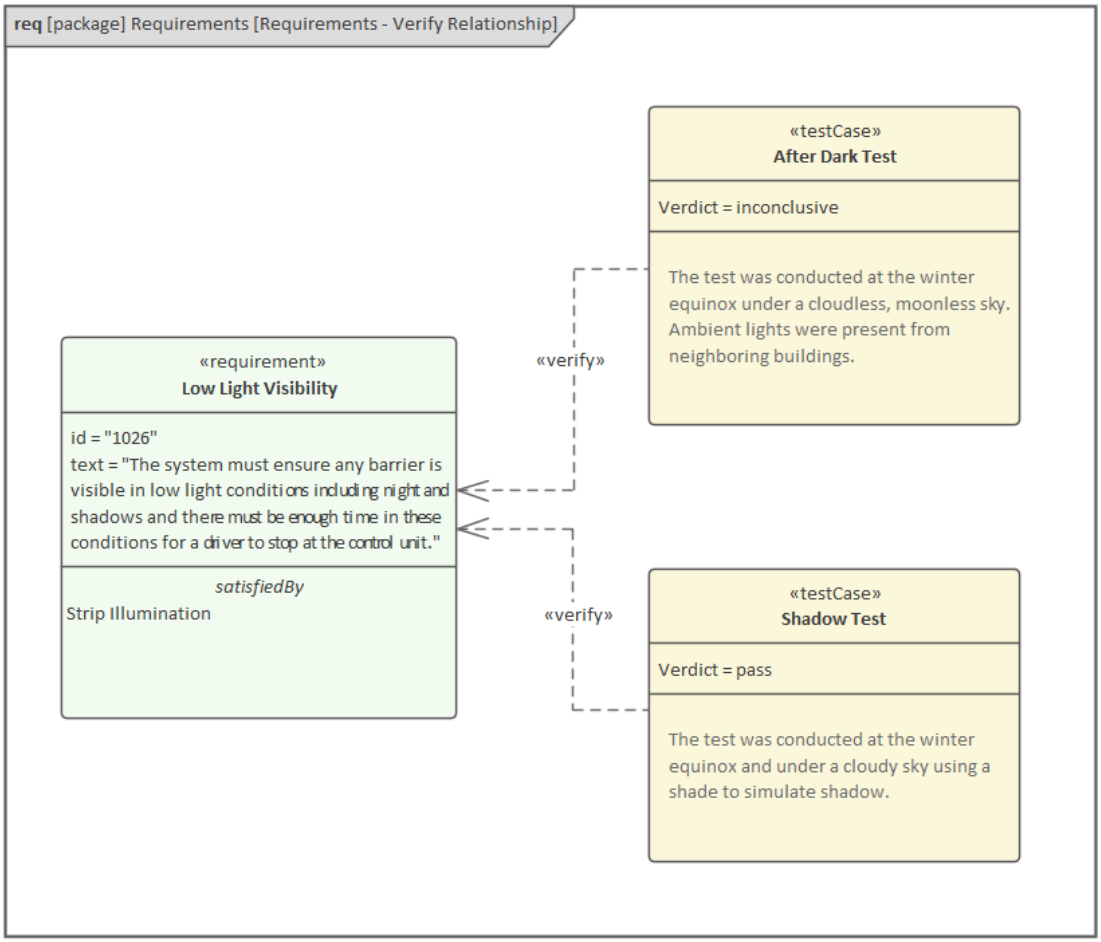
The Verify relationship is available from relationships page of the SysML Requirements Toolbox.



The Test Case can describe the method or testing process; it contains a tag that defines the verdict (test result), which can be:

- pass
- fail
- inconclusive
- error
- a user defined value

The *Satisfy* relationship has a related purpose in that it describes which part of the design or system is actually used to carry out the notion expressed in the Requirement.

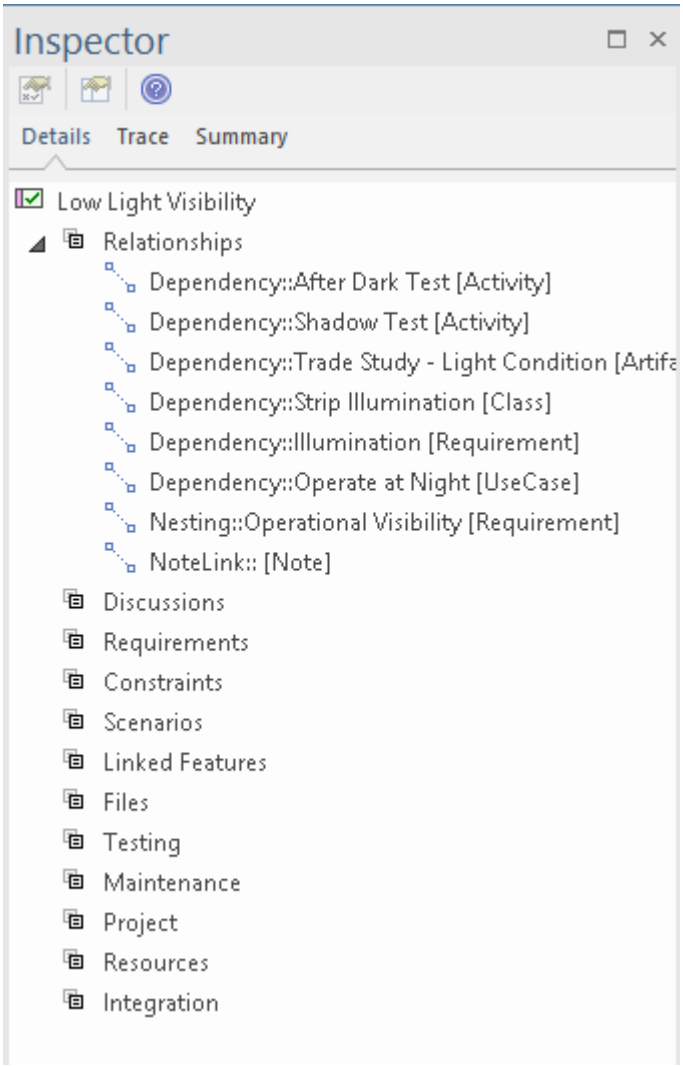


In this diagram a Requirement describes Low Light Conditions, and there are two separate Test Cases used to verify the Requirement. The modeler has chosen to display the 'satisfiedBy' compartment to help clarify what part of the implementation is being subjected to the test. Notice that the verdict is different for each Test Case: the After Dark Test is inconclusive whereas the Shadow Test passes. A modeler can choose to show or hide the Verdict tag in individual diagrams.

Visualizing Requirement Relationships

Relationships between Requirements and other elements - including other Requirements - are a critical aspect of Model Based Systems Engineering. In many ways these relationships are the important bridges between specification and design, or problem and solution. These relationships can be viewed in a wide range of specialized windows and user interface mechanisms. One of the first things a newcomer to Enterprise Architect will observe is that these relationships are not visibly nested under elements in the 'Project' tab of the Browser window. It has been a conscious design decision not to clutter the 'Project' tab with these relationships, but rather to make them visible in other displays that can be docked and viewed at the same time as the elements they relate to.

This illustration shows the 'Details' tab of the Inspector window with the 'Relationships' node expanded to show all the relationships that are connected to the 'Low Light Visibility' Requirement, which has been selected in the 'Project' tab. There is also a dedicated Relationships window.

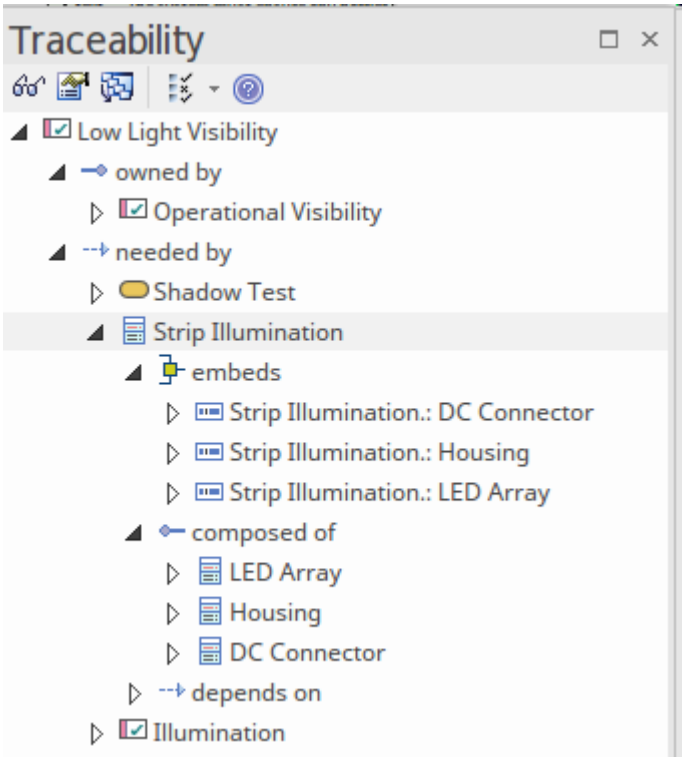


There is also a wide range of other windows where relationships can be visualized. For more information see the The Browser Window topic.

Traceability Window

The Traceability window is a helpful and unique feature of Enterprise Architect that allows relationships to be explored, so that the modeler can effectively take walks through the graph of elements and their connections. It is a useful window to have open when you want to view how an element is connected and also what the connected elements are connected to. So, for example, in this model of the Boom Gate when the 'Low Light Visibility' Requirement is selected, the

Traceability window will show that it is connected to the 'Strip Lighting' Block that satisfies the requirement.



The Traceability window can be opened from the ribbon option 'Design > Element > Trace'.

The modeler in this situation might also be interested in the structural aspects of the 'Strip Lighting' Block and so can follow this element's relationships to discover its structural relationships, walking the graph to find answers to questions and exploring the model.

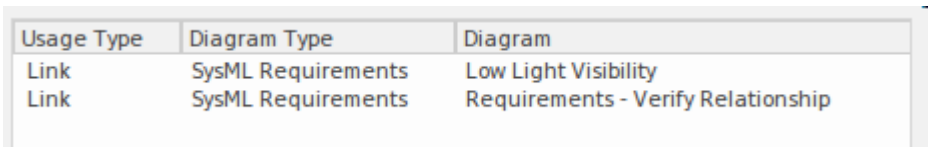
If you are not concerned about viewing the relationships it is best not have this window open, as its contents must be rendered each time you change focus to another element and, for well connected elements, this can take some time, increasing the time it takes to move around the model. For more information see the Traceability Window Help topic.

Relationships Window

The Relationships window is a useful window to have open when working with crosscutting concerns, as is typically the case with Requirements. When an element is selected in a window, the Browser window or a diagram, the Relationships window displays a list of the connectors that either target the selected element (target) or emanate from the element (source). Another useful aspect of this window is that the 'View' column indicates whether the relationship is visible in the currently open diagram.

The Relationships window can be opened from the ribbon option 'Start > All Windows > Properties > Responsibilities > Relations'.

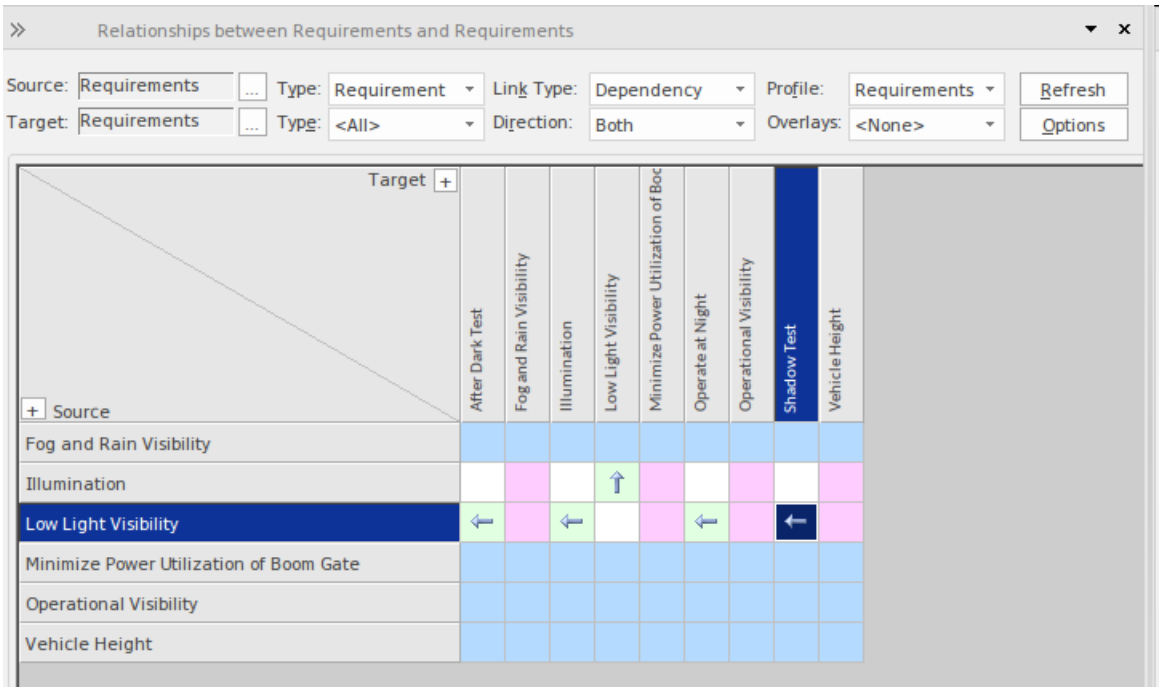
A modeler can also locate all diagrams that contain the selected relationship, by choosing the 'Find in all Diagrams' option from the context menu. In this illustration it can be seen that the 'deriveReqt' relationship that connects the 'Low Light Visibility' Requirement and the 'Illumination' Requirement exists in two diagrams. For more information see the The Relationships Window Help topic.



Usage Type	Diagram Type	Diagram
Link	SysML Requirements	Low Light Visibility
Link	SysML Requirements	Requirements - Verify Relationship

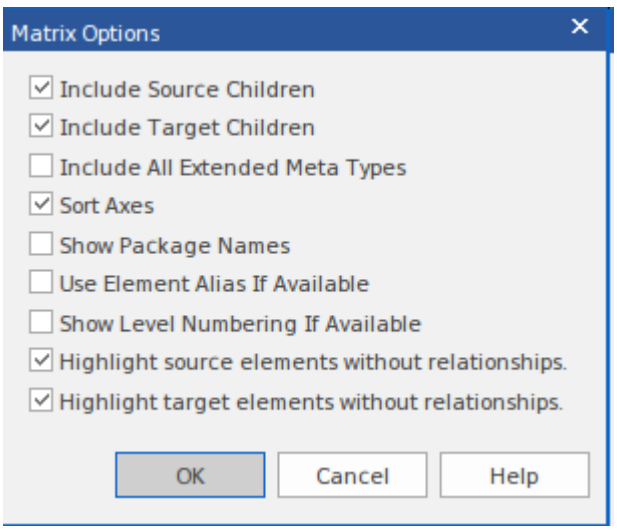
Relationship Matrix

The Relationship Matrix is a great tool for visualizing the connections between the elements in any two Packages, in an interface resembling a spreadsheet with rows and columns. The tool is particularly useful when used with Requirements, and allows an engineer to see how Requirements are related to other elements, including other Requirements.



The Relationship Matrix can be opened from the ribbon option 'Design >Package > Package/Matrix'. Select whether the current Package is the source Package, target Package, or both.

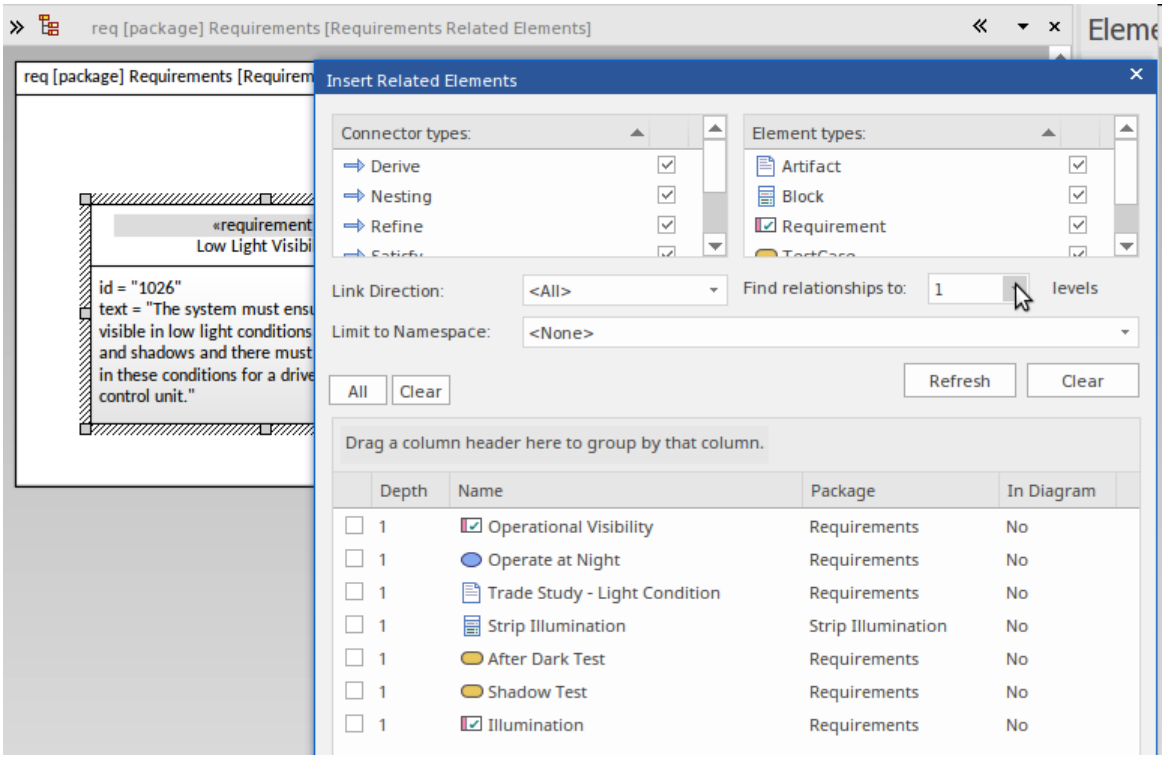
Where a relationship exists an arrow icon will be displayed in the cell at the intersection of the source and target elements, with the arrowhead showing the direction of the relationship. The matrix can also be configured to highlight the rows or columns that do not have any relationships in a separate color. This and other options can be configured on the Options window, available from the Options button in the Relationship Matrix header.



These options allow you to tailor the way that the matrix is displayed, including whether elements are sorted and their names are prefixed with the Package name, and whether source and target element rows and columns without connections are highlighted. For more information see the Relationship Matrix Help topic.

Insert Related Elements

The Insert Related Elements feature is a productivity tool that allows an engineer or other stakeholder to quickly construct a diagram by inserting a central element and then asking the tool to find all elements related to this element, down to an arbitrary depth of connectivity. This helps the engineer to effectively explore the graph of elements and create a diagram that shows how other elements in the repository directly relate to this central element and how other elements relate to those elements. Element and connector types can be specified for inclusion or exclusion, and the depth can be changed to bring more or fewer elements and connectors into the diagram. The diagram can be automatically laid out or the diagram layout tool can be used to reorganize the layout to make it more appealing or relevant.

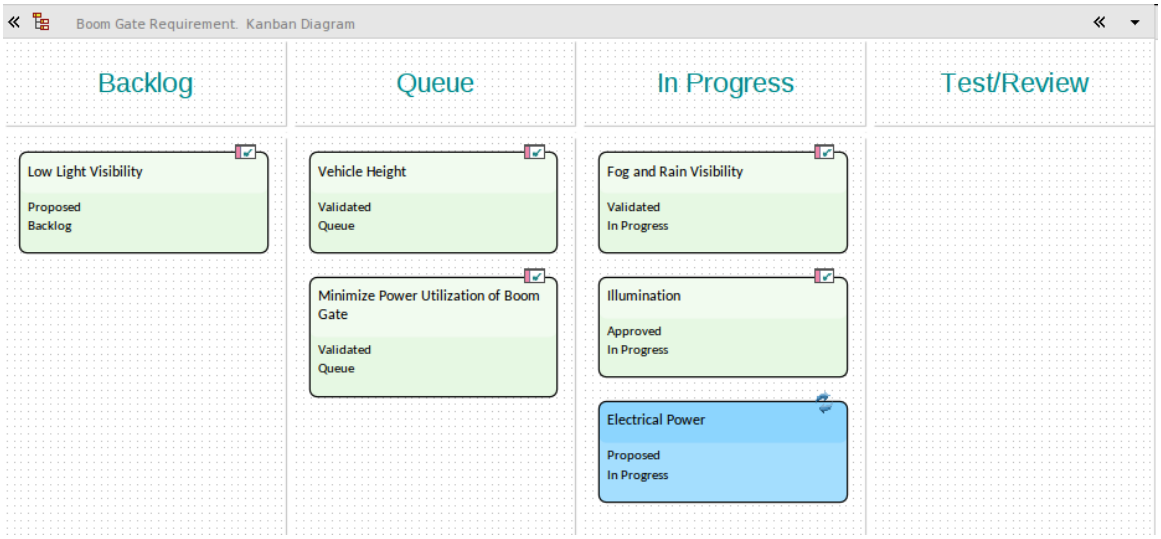


The Insert Related Elements feature can be used by selecting an element in a diagram and then using the ribbon option 'Design > Element > Add Element > Related Element'.

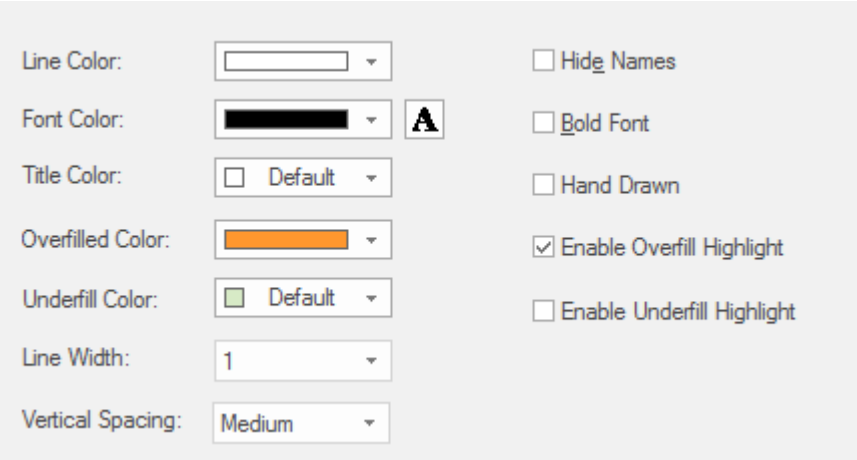
The feature can be used as an exploratory tool where the rendered diagrams are constructed as part of an enquiry process and are discarded after they have served their purpose. Alternatively, the feature can be used to create more permanent diagrams that can be saved and reused for visualization. Either way the tool will save the engineer time, and enable the creation of accurate and expressive diagrams that are bound to impress stakeholders who would otherwise not have been able to visualize the connections between elements. For more information see the Insert Related Elements Help topic.

Kanban Diagrams

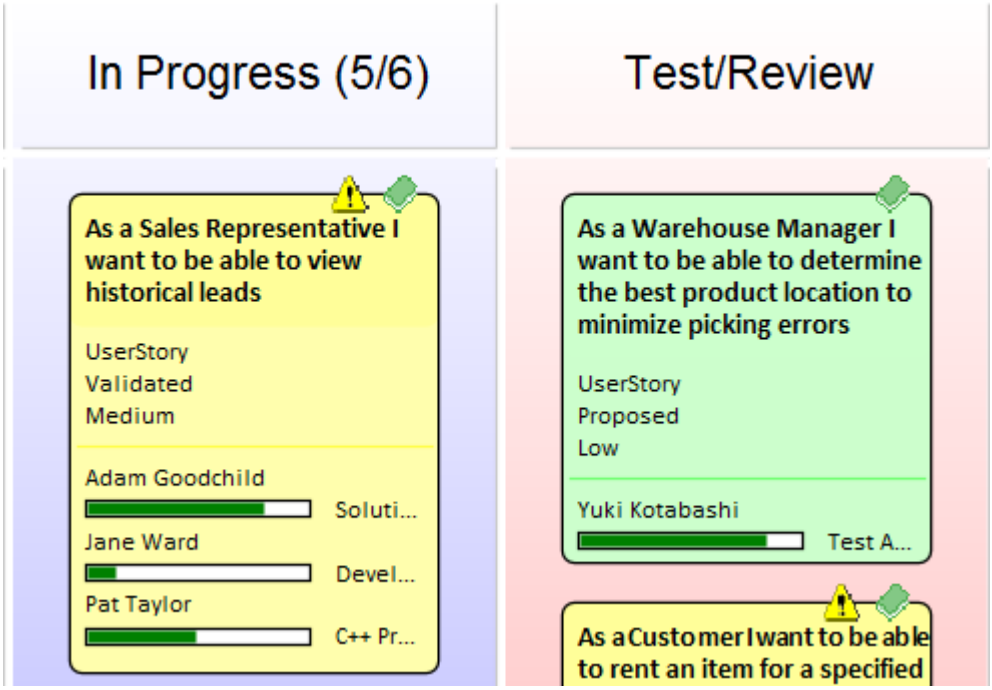
Kanban diagrams can be used to visualize requirements (and other elements) as they pass through the stages of an industry standard or proprietary process or methodology. The concept of Kanban has its origins in process efficiency analysis in the automotive industry in the latter years of 20th Century Japan. It has since then been adopted by a number of software and system communities as an effective way of managing requirements from the backlog or queue to their implementation. Enterprise Architect has a full and pragmatic implementation of Kanban that can support a number of different requirement processes.



In this diagram we see a number of columns that represent the stages in the requirements process, allowing elements to be dragged between columns - typically from left to right but occasionally elements can be returned to a Backlog, for example. The diagram is completely configurable by the engineer, allowing the number of columns and their names and a wide range of other aspects of the diagram to be configured, including Bound Properties, Work in Progress limits and colors to name a few.



It is also possible to show the progress that has been made on a particular requirement, by applying resource allocations and displaying each resource as a progress bar showing the percentage completion for the task. In this diagram we see a number of elements, one of which shows three resources working on the same element. For more information see the Kanban Boards Help topic.


In Progress (5/6)
Test/Review
As a Sales Representative I want to be able to view historical leads
UserStory
Validated
Medium
Adam Goodchild
Soluti...
Jane Ward
Devel...
Pat Taylor
C++ Pr...
As a Warehouse Manager I want to be able to determine the best product location to minimize picking errors
UserStory
Proposed
Low
Yuki Kotabashi
Test A...
As a Customer I want to be able to rent an item for a specified

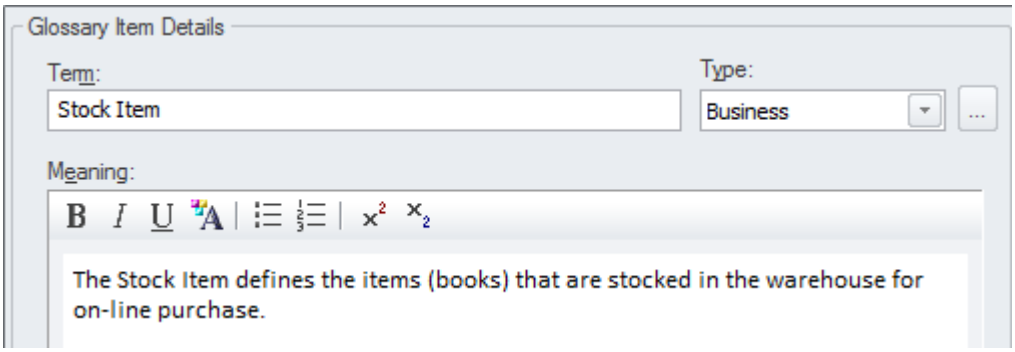
Documenting Requirements

A number of documents are commonly produced as part of the Requirements Engineering discipline, such as the Software (System) Requirements Specification and Use Case Report. These can be generated automatically from a requirements model using built-in templates. In addition a wide range of other documents can be produced using built-in or customized templates. The documentation facility in Enterprise Architect is highly configurable and many reports can be produced using the template system, but for more complex reports there is a facility called Virtual Documents that allows a publisher to model the structure of the document and to cherry pick content from anywhere in the repository, applying different templates to each section of the document. There is also a wide variety of options that can be applied at the template or document generation level, and the Scripting engine can be used to inject content into a document or to produce the entire report.

There is also a Custom Document facility that allows a modeler to define a dynamically created document by simply dragging content from the Browser in the form of elements, diagrams or Packages and applying a built-in or a user defined template to each item that specifies how the content will be rendered. This allows a document to be visualized and can include any handcrafted content or images in addition to the content injected from the Browser.

Project Glossary

A Project Glossary lists and defines the terms that are important for a project or program of work. The Project Glossary can be generated as an isolated document, or it can be included as a section in one or more other documents. It provides a single point of truth for the important project terms and their meanings; when new documentation is generated the terms will automatically be updated. The Glossary can be generated to a DOCX or PDF format, or to HTML that could be included in a project or organization level web site. The Glossary allows the modeler to categorize the terms into user-defined Types, and these can have styles applied when they are generated in documentation.

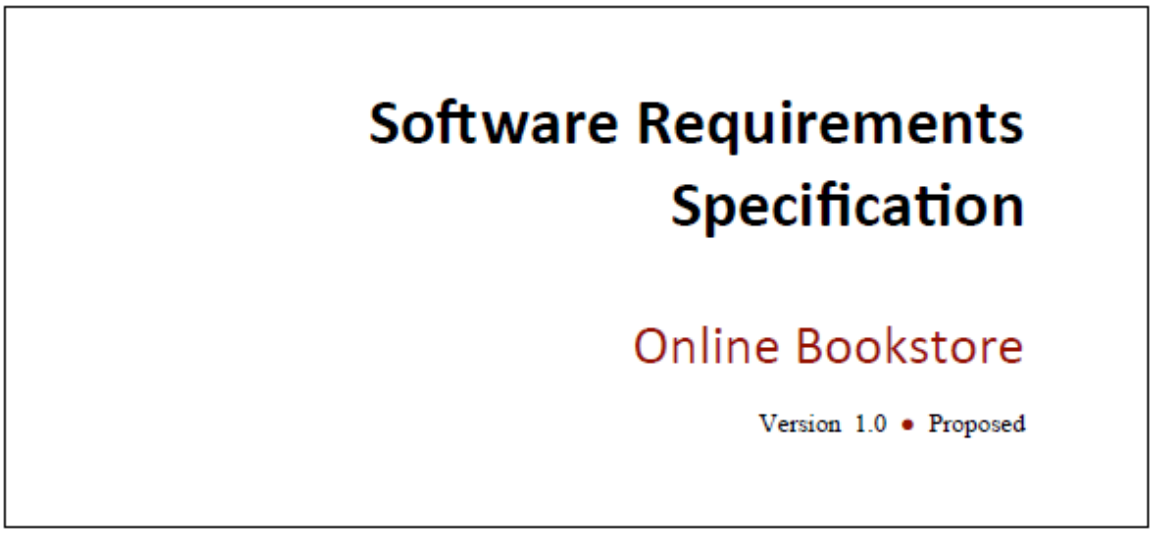


The Project Glossary can be viewed and managed from this ribbon location:

Design > Dictionary > Glossary > Glossary View

Software Requirement Specification

This document describes the Requirements of the system, its behavior under defined conditions, and the constraints that it must operate under; it will typically be read by a variety of stakeholders. There is a built-in Requirements template that can be used to generate the document, although the modeler is free to create a new template that could be either based on this or created from a blank template. When the document has content from a variety of locations in the Browser window, it would be most expedient to use the Virtual Documents facility, which allows the user to create a model of the document (similar to a Master document in a Word Processor) that has a number of sections called Model Documents. These can have content picked from anywhere in the Browser window.



Describing User Goals with Use Cases

Use Cases were originally devised by Ivar Jacobson, a Swedish electrical engineer who was also an important proponent in the development of the Unified Modeling Language (UML). Use Cases are used as a method for representing functional requirements from the users' perspective. They are said to be goal driven, because the Use Case defines the goal that the user is trying to achieve while interacting with the system. Enterprise Architect fully supports the development of Use Case diagrams, but also fully supports the modeling and management of Use Case text; it has a unique and highly productive tool for working with Use Cases, called the Scenario Builder.

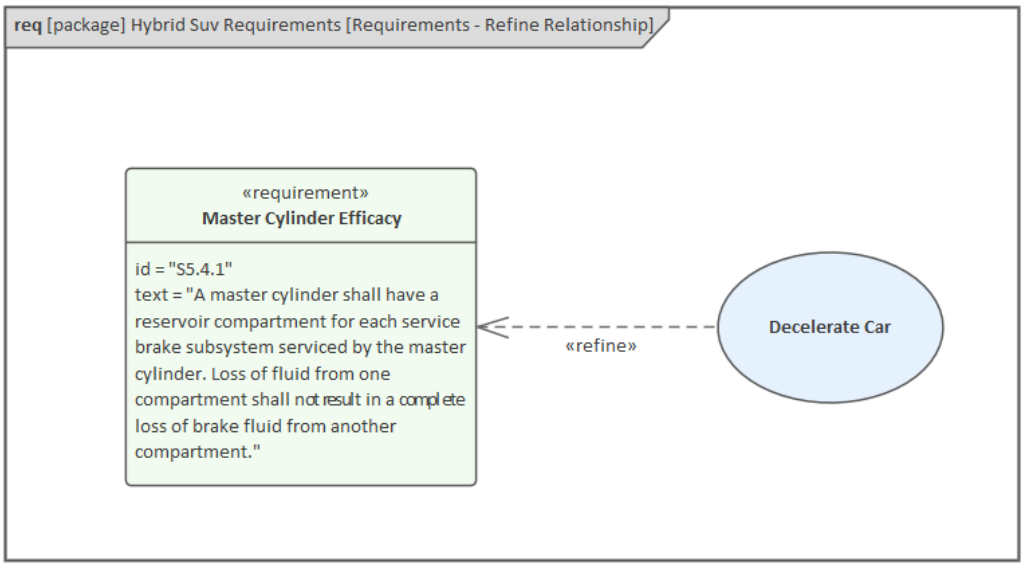
This innovative tool not only allows Use Cases to be modeled at any level of detail, but also automatically creates behavioral models that allow the detailed steps of a Use Case and the interaction between the Actor and the System to be visualized and related to other parts of the model.

The Use Case is a close cousin to the User Story, which is used in a number of Agile Software development techniques. The term originally coined in Swedish is more naturally translated as Usage Scenario, which provides a more compelling explanation of the method.

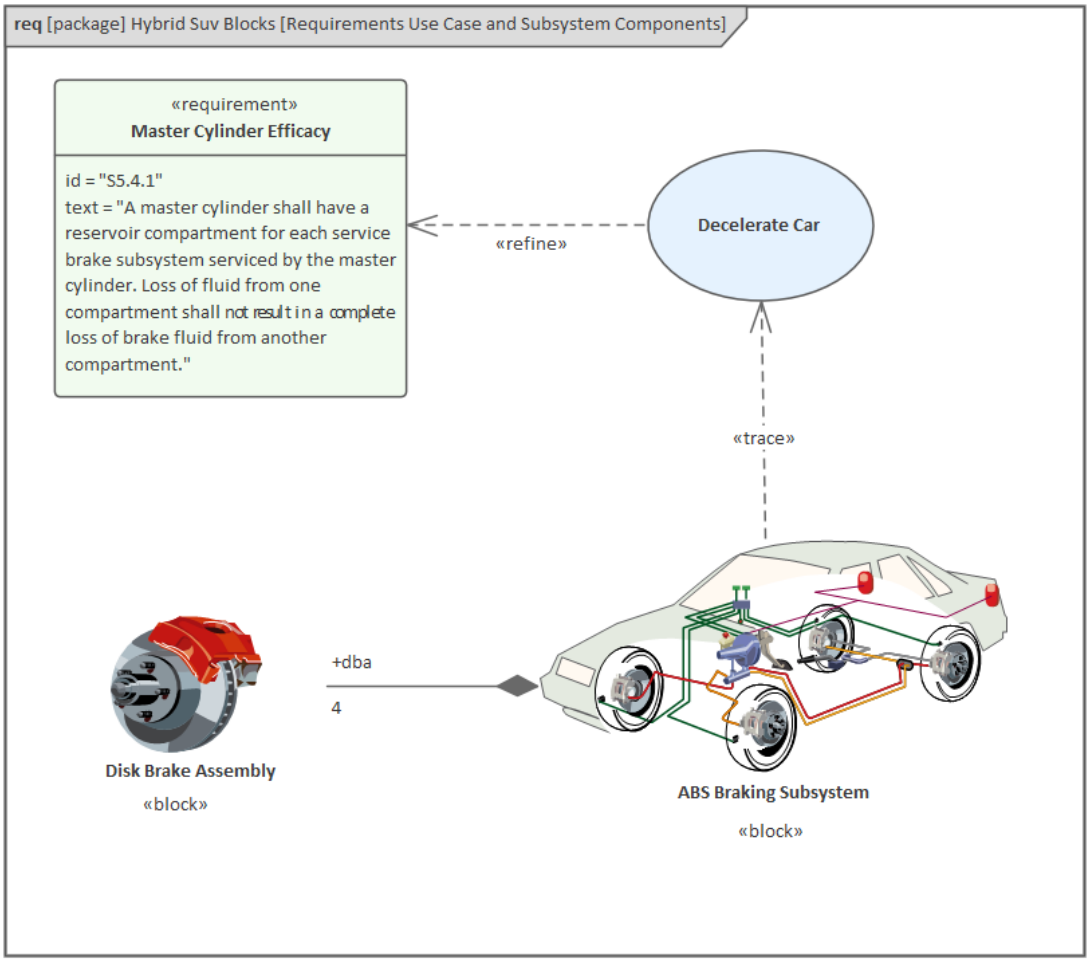
Requirements and Use Cases

The Use Case technique is fundamentally very simple and was originally devised to ensure that functional requirements were written from the perspective of the User. This standpoint helped to ensure that deployed systems would be fit for purpose and be accepted by the diverse community of users. There is however a vast amount of conflicting literature and an equally large number of styles for defining Use Cases. This has led to confusion and uncertainty and has tended to attenuate the value that can be derived from this effective and simple technique.

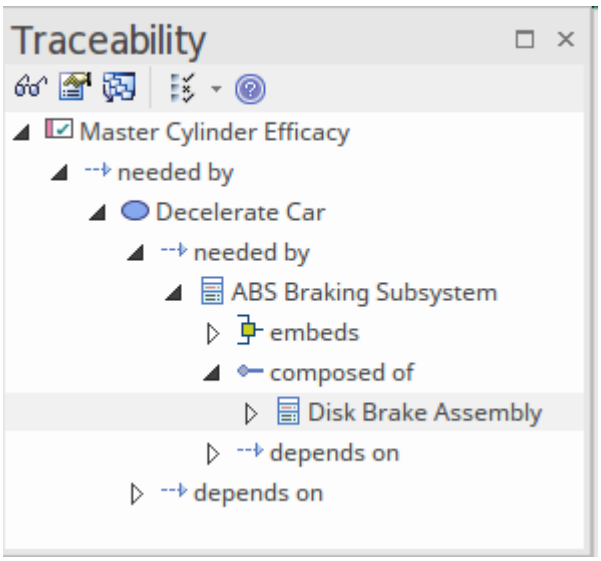
In software engineering, many methods prescribe the use of Use Cases as an alternative to Requirements development, because the Unified Modeling Language (UML) does not include a formal Requirement element. In contrast, most Model Based Systems Engineering methods using SysML combine the application of Use Cases and Requirements. This is a result of the fact that SysML defines both a Use Case and a Requirement element so these two elements can be related to each other and compliment the system specification to bring clarity a precision to the important discipline of requirements engineering and management.



In these two diagrams the modeler has used the <<refine>> relationship to indicate that the *Decelerate Car* Use Case refines or adds additional explanation to clarify the Requirement *Master Cylinder Efficacy*. This provides a mechanism to trace implementation level components that are connected to the Use Case back up to the Requirement and ultimately to the Stakeholder.



The Traceability window can also be used to view the connections between model elements at different levels of abstraction and to see the connection from a Block that forms part of a subsystem assembly back to the Requirement that specified the functionality.



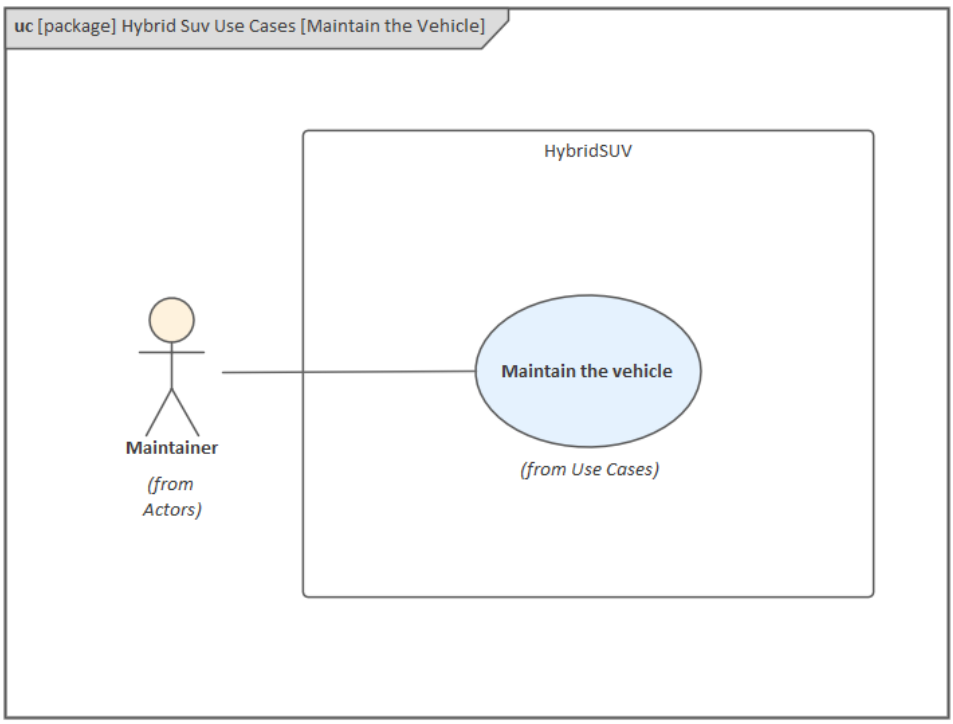
Use Cases are typically used to refine the high level requirements and to express the communication and interaction between the User and the System.

Introducing Use Case Diagrams

The Use Case diagram is a simple diagram that visually describes the users' goals with respect to the system or part of the system. This could be paraphrased to 'the value that the system provides to the Actors'. Use Case diagrams appear quite simplistic, with a small number of elements:

- Subject
- Actors
- Use Cases

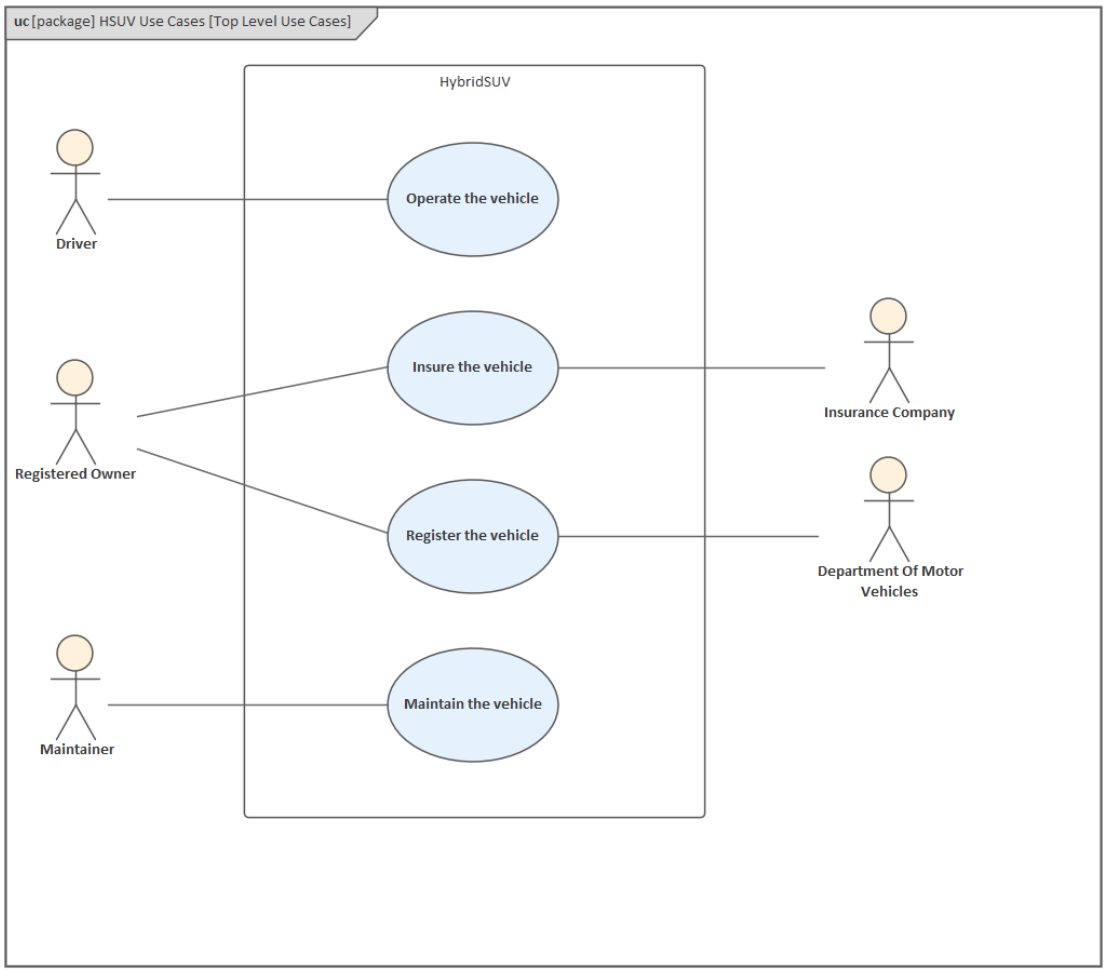
These are connected by a series of relationships.



The Subject (boundary) provides a context for the definition and represents a system or part of a system; the Actors by definition lie outside the Subject and the Use Cases within. The Communication Path relationship by definition crosses the perimeter of the Subject as it connects an Actor with a Use Case. Again, the number of relationships is quite limited, but each has specific meaning in the diagram.

- Communication path
- Extend (also with Condition)
- Include
- Generalization

As with all SysML elements, there is both a graphical and textual aspect to the elements, and in the description of Use Cases there is typically more emphasis on the textual or narrative aspect.



Any number of Use Case diagrams can be created to represent the users' interaction with the system or part of a system. It is important to understand that Use Cases are intended to describe the value the system provides for its users and they are not intended to be broken down by functional decomposition. This is unquestionably the most common mistake made by novice modelers, resulting in the attenuation of the profound benefits that can be gained by this technique.

The Use Case model can be embellished by a mechanism called 'structuring the Use Case Model', which factors out repeating text, classifies Actors and Use Cases, and specifies extension points. This mechanism will be discussed later in this chapter. For more information see the SysML Use Case Models Help topic.

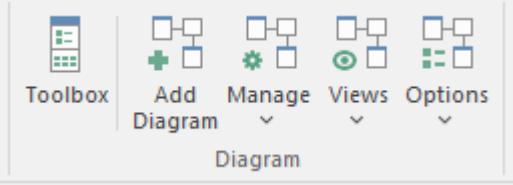
Creating a Use Case Diagram

A Use Case diagram can be created from a number of places in the User Interface by selecting:

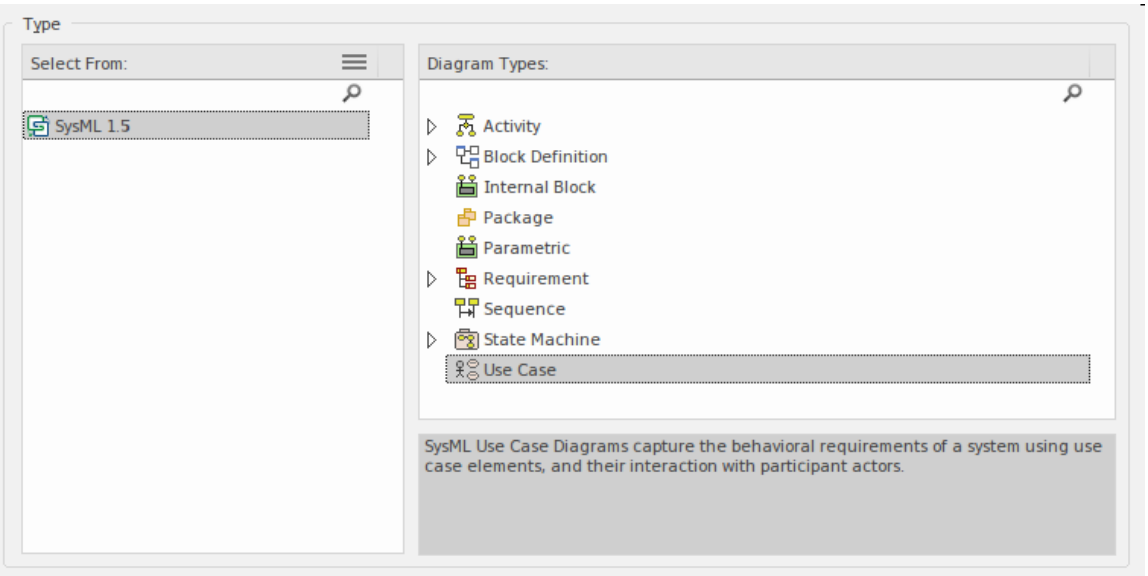
- Design ribbon - *Add Diagram* icon on the *Diagram* panel
- Browser window toolbar - *New Diagram* icon
- Browser window context menu - *New Diagram*

We will use the Design ribbon to create a Use Case diagram. Firstly, select the location in the Browser window where you want the Use Case diagram to be located. As with all diagrams, this can be either a Package or an element, but it is common to insert Use Case diagrams into a Package. Once the Package location has been selected in the Browser window, select:

Design > Diagram > Add Diagram



Selecting this option will open the *Diagram Buildler* tab page of the *Model Builder* dialog, where you select the diagram type and name the diagram; the name initially defaults to the name of the Package or element that contains the diagram. With the SysML perspective chosen and the version of SysML selected, a list of diagram types will be displayed allowing you to choose the Use Case diagram. Click on the *Create Diagram* button to create a new Use Case diagram in the location selected in the Browser window. The Diagram View will be opened, allowing you to start adding elements and connectors that describe the value that the system will provide to its users. Enterprise Architect will also display the 'Use Case' pages of the Diagram Toolbox that contain the elements and relationships defined by the SysML specification to be applicable for constructing Use Case diagrams. Any number of other Toolbox pages can be opened, if required, in addition to the *Common* elements and *Common Relationships* Toolbox pages that will always be available.



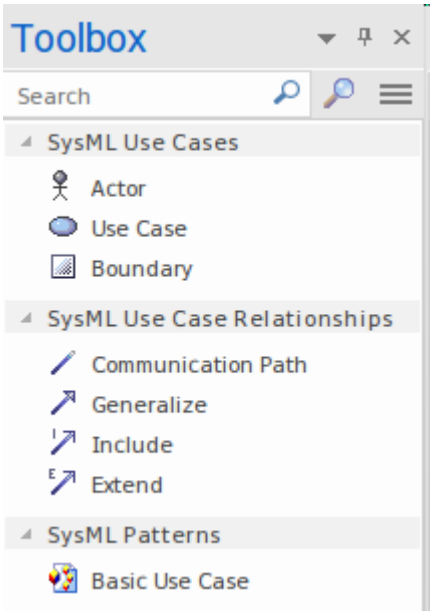
The most important elements and connectors that are used with the Use Case diagram are:

Elements

- Actor - used to describe a role a user plays with respect to the system
- Use Case - used to describe the value a system provides to its users
- Boundary - used to show the scope of the systems (typically one per diagram)

Connectors

- Communication Path - used to connector Actors to Use Cases
- Generalize - used between two Actors or between two Use Cases
- Include - used between to Use Case to reuse scenario steps
- Extend - used to embellish a Use Case with extra detail

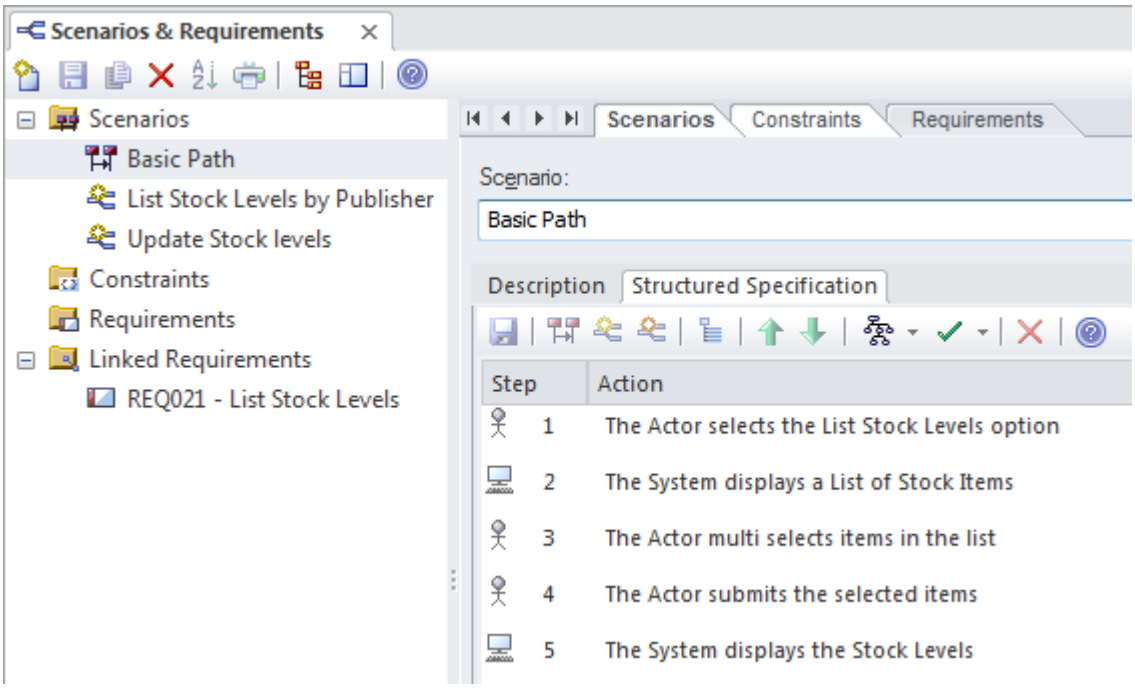


Elements can be added to the diagram by dragging-and-dropping them from the Toolbox onto the Diagram View. It is considered good practice to start with a Boundary element, which should be named appropriately to describe the system, sub-system or entity being modeled by the Use Case diagram. Leaving the name blank, or giving it a name that does not make it clear to the reader what system or part of a system is being modeled, can lead to misinterpretation of the diagram. With the Boundary added and appropriately sized in the diagram, Actors and Use Cases can be added - Actors positioned outside the Boundary and Use Cases inside. The next step is to add Communication Path relationships between Actors and Use Cases, thus defining the value that the Actors derive from the system.

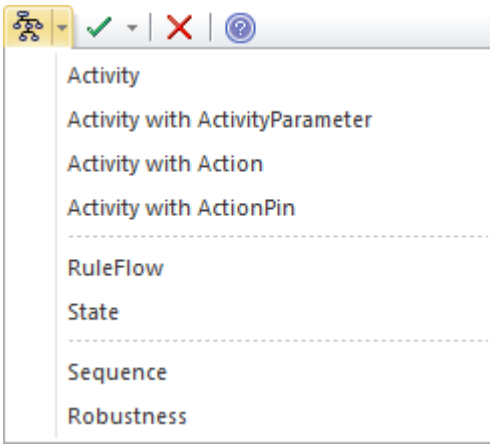
Once a basic diagram has been created, and as knowledge of the domain and the system's behaviors are further revealed, it is possible to structure or embellish the diagram using the additional relationships of Include, Extend and Generalize. The newcomer is cautioned against using these relationships too liberally, and any attempt to use functional decomposition will attenuate the value of the Use Case model, which is intentionally broad in its description to allow stakeholders to get a 10,000 meter view of the services provided by the system, sub-system or entity being modeled.

Meet the Scenario Builder

There is a wide range of tools for working with Use Cases, but none more important and useful than the Scenario Builder. This unique tool bridges the gap between what has traditionally been done within Word Processors or isolated tools that separate the Use Case diagram with its Actors and Use Cases, and the steps of the scenarios.

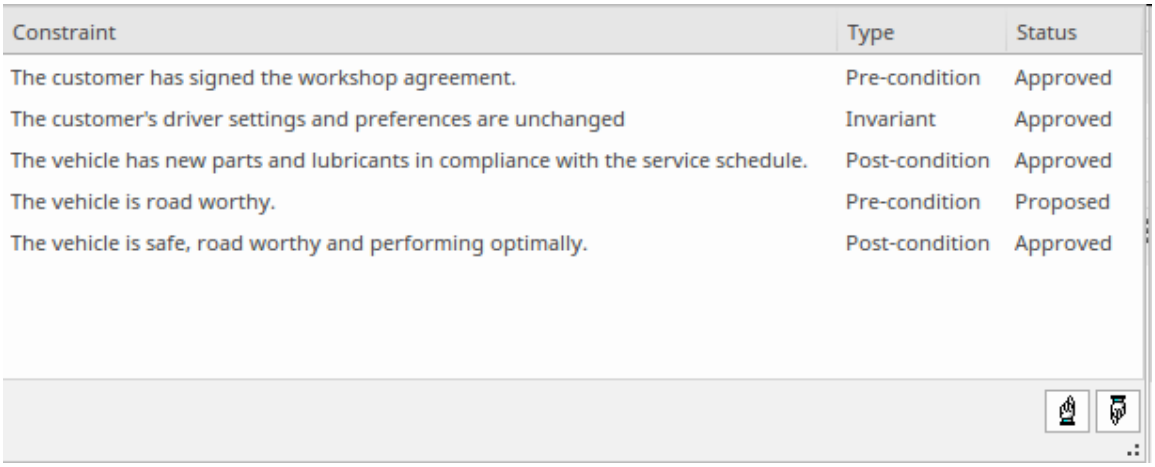


The Scenario Builder also provides a mechanism to automatically generate behavioral models directly from the Scenario steps, allowing elements of the architecture and design to be related to individual steps.



The tool provides a whole range of options that will be suitable for any Use Case or Requirements process, ranging from what is commonly termed a basic process where the Actor and Use Cases are named and the Use Case is given a description, to a partially-dressed process where the basic flow is completed. A fully-dressed process will typically detail all the steps in the basic flow and define and detail the steps of Alternate and Exception Scenarios. For more information see the Scenario Builder topic.

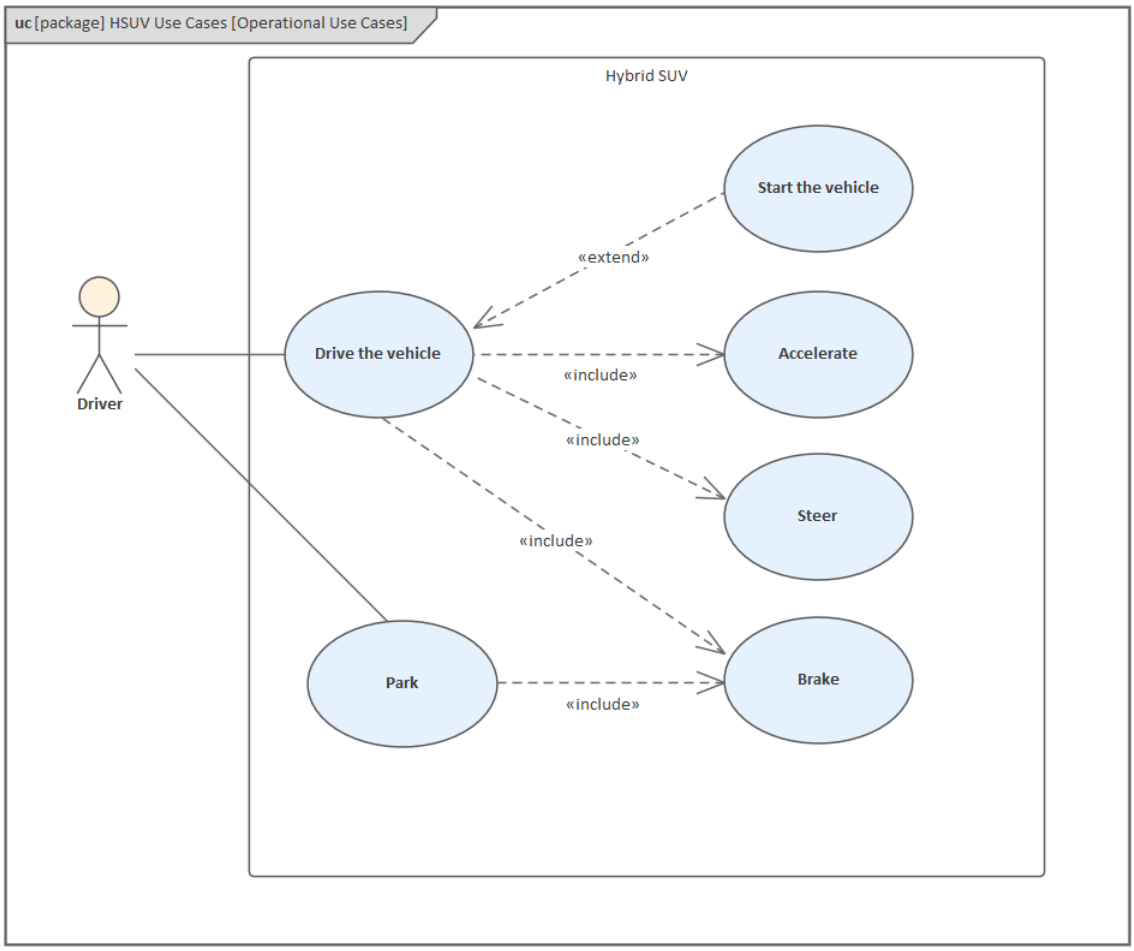
In addition any number of constraints can be added, such as pre-conditions and post-conditions and invariants.



Constraint	Type	Status
The customer has signed the workshop agreement.	Pre-condition	Approved
The customer's driver settings and preferences are unchanged	Invariant	Approved
The vehicle has new parts and lubricants in compliance with the service schedule.	Post-condition	Approved
The vehicle is road worthy.	Pre-condition	Proposed
The vehicle is safe, road worthy and performing optimally.	Post-condition	Approved

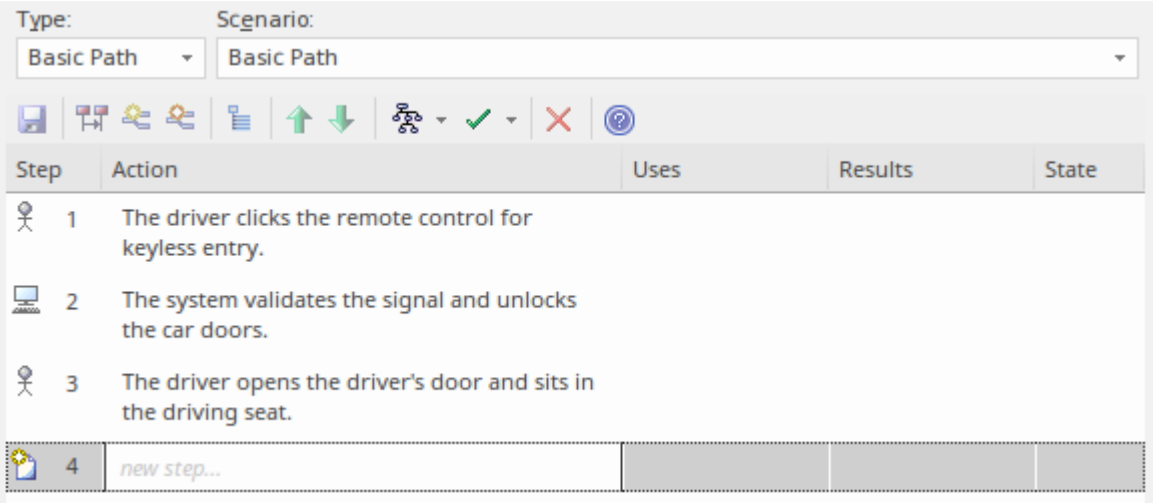
Structuring a Use Case Model

While the Use Case model provides a high level of visualization, and a systems engineer is cautioned against applying functional decomposition, SysML does provide a number of mechanisms that can help to structure a Use Case model to ensure that discrete pieces of functionality can be reused. These mechanisms consist of the <<include>> Dependency, <<extend>> Dependency and Generalization relationship.



Generating Behavior Diagrams

Enterprise Architect has a helpful productivity tool that allows behavior diagrams to be automatically generated from Use Case specifications defined in the Scenario Builder. This provides a way of visualizing these otherwise textual descriptions. It also allows relationships to be drawn between steps in a Use Case description and other modeling elements.



Use Case Report

The creation of Use Case documentation has traditionally been a manual process and with the documents in many projects running into hundreds of pages their production consumes valuable project resources. These hand-crafted documents become difficult to maintain and remain isolated from other parts of the project such as Requirements, Business Rules and solution Components. Enterprise Architect has a multi-featured tool called the Scenario Builder that allows the modeler to specify Use Cases and Scenarios inside the model and these can be automatically generated to high quality documentation using built-in templates. There are two built-in templates that can be used for generating a Use Case report: one documents the Use Case at a summary level and the other at a detailed level.

Example content from a Use Case Report

The detailed Use Case report will list all the details of the Use Case and the detailed steps, including Basic Paths, Alternate and Exception Scenarios. Other information, including Internal Requirements, Pre- and Post-Conditions and other Constraints will also be included in the report. If a Behavioral diagram such as an Activity diagram has been automatically created, this diagram will also be displayed in the report.

Alternate. List Stock Levels by Publisher

The List Stock Levels by Publisher allows a user to obtain stock level information for a selected publisher. The Stock Control Manager and Storeroom Worker need this information to plan logistics and to ensure that stock remains at adequate levels to service incoming requests. There is also the need to predict the date that the stock items will fall below an acceptable level

SCENARIOS

based on purchase cycles and promotional periods.

1. User selects "List Stock Levels by Publisher"
   Uses:

2. System returns a list of publishers to select from
   Uses:

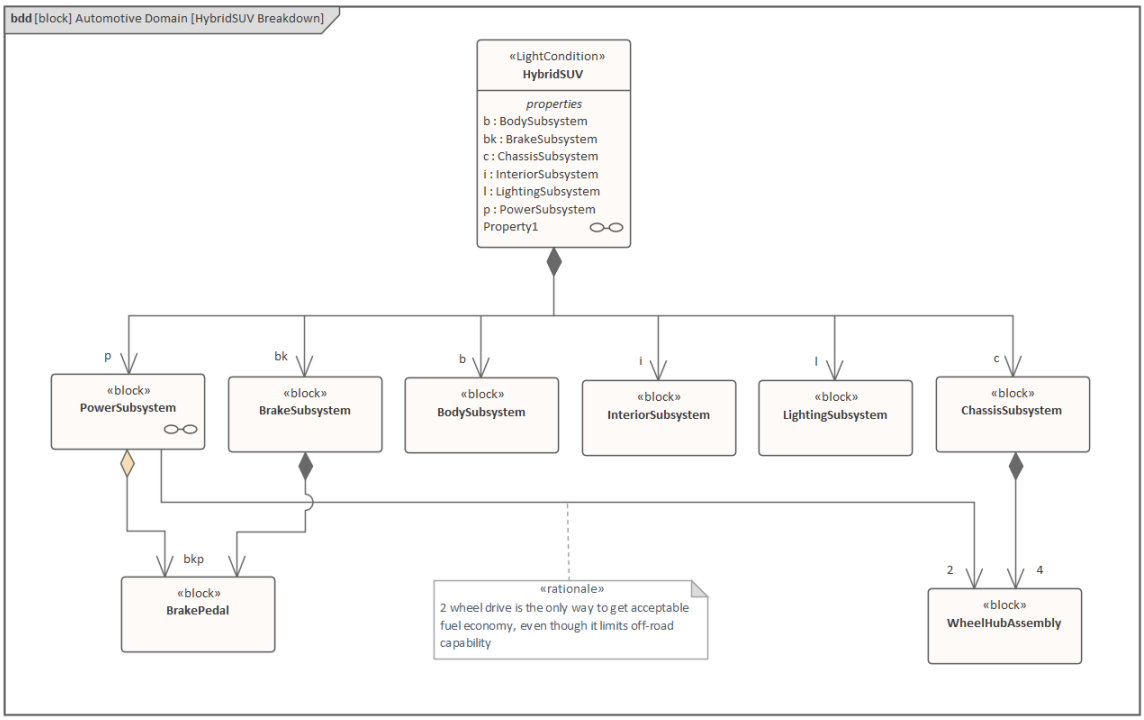
3. User Selects a publisher
   Uses:

4. System returns a listing of titles and quantity in stock for the publisher
   Uses:

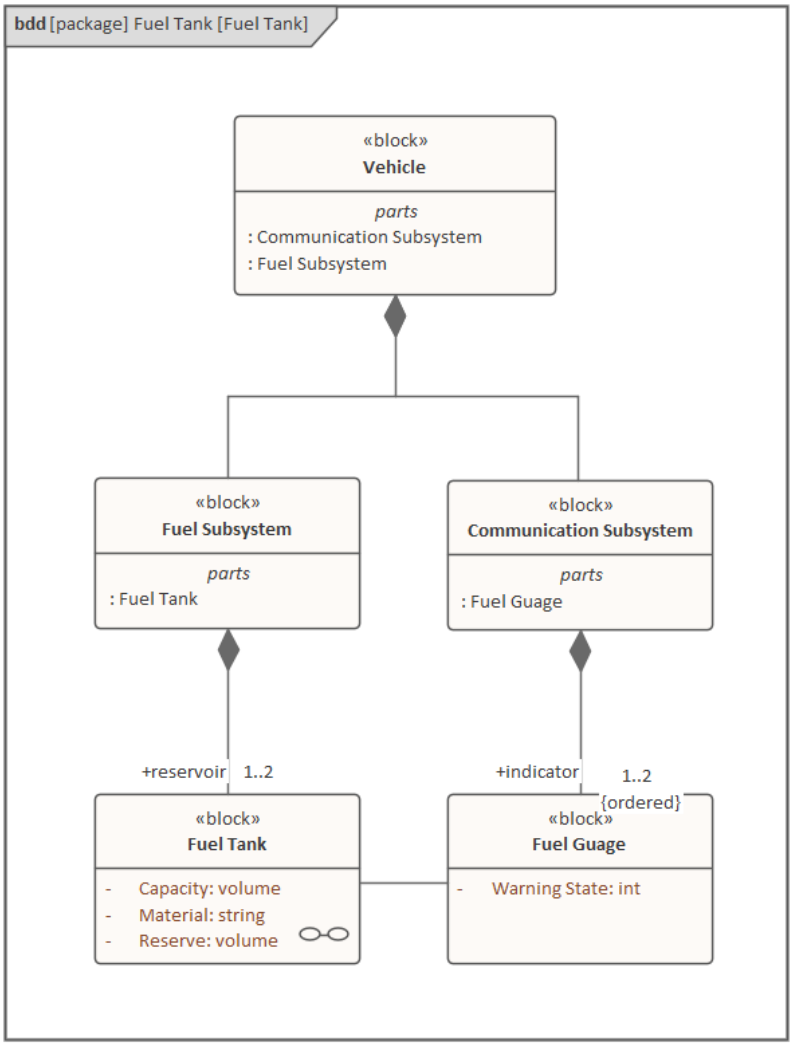
Using Blocks to Model Structure and Constraints

The language constructs and expressions in SysML, as with our natural languages, can be divided into structural and behavioral types. In languages such as English, German or Japanese, nouns describe structure and verbs describe behavior. Sentences typically contain a combination of nouns and verbs that bring to light some aspect of the speaker's world. The SysML has a similar division, with elements that describe Structure and other elements that describe Behavior. In SysML the structural things (nouns) are described using a Block. When engineers create diagrams they will often have a mixture of behavior or structure elements and they will describe a particular aspect of a system - bringing to light some aspect of the system being modeled.

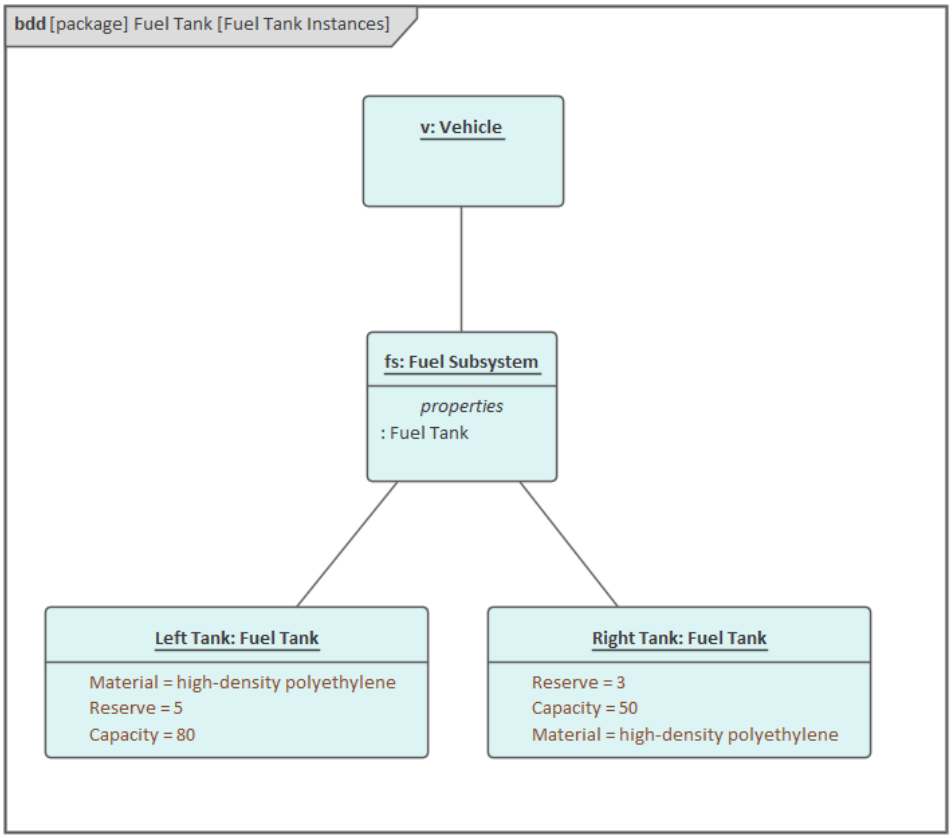
The Block is the fundamental unit of system structure; it can be used to describe an entire system, a subsystem, a component, an item that flows through a system, a constraint, or entities that reside outside a system. In a similar way to our natural languages, a Block can represent something abstract, logical or physical. This is an important concept, and writers and readers of the SysML must be clear as to the intention of the representation. For example, in a logical architecture there are typically Blocks representing conceptual ideas or designs that at the time of detailed design and construction might be realized by physical and tangible components. A systems architect might define a Block called *Collision Detection Subsystem* that is an expression of a logical system component that could at the detailed design phase, be in part, realized by a set of radar and laser transmitters, detectors and cameras.



A number of our natural languages have a grammatical term called *classifiers,* which group the things (nouns) of a lexicon into classes of things that share common characteristics and behavior. This same principle applies to Blocks, which are essentially a type of classifier that groups a collection of instances that share the same structural and behavioral features. Instances of a Block can be modeled in a generic way or they can be given precise values, such as the volume of petrol contained in a fuel tank at a particular point in a journey or at the time of an accident.



In the Fuel Tank diagram the car is modeled as a classifier (Block) level, where the model is describing a generic vehicle and representing the fact that a vehicle could have one, or a maximum of two, fuel tanks. This Fuel Tank Instances diagram, however, describes a particular vehicle that has two fuel tanks that have different capacities and reserve volumes.

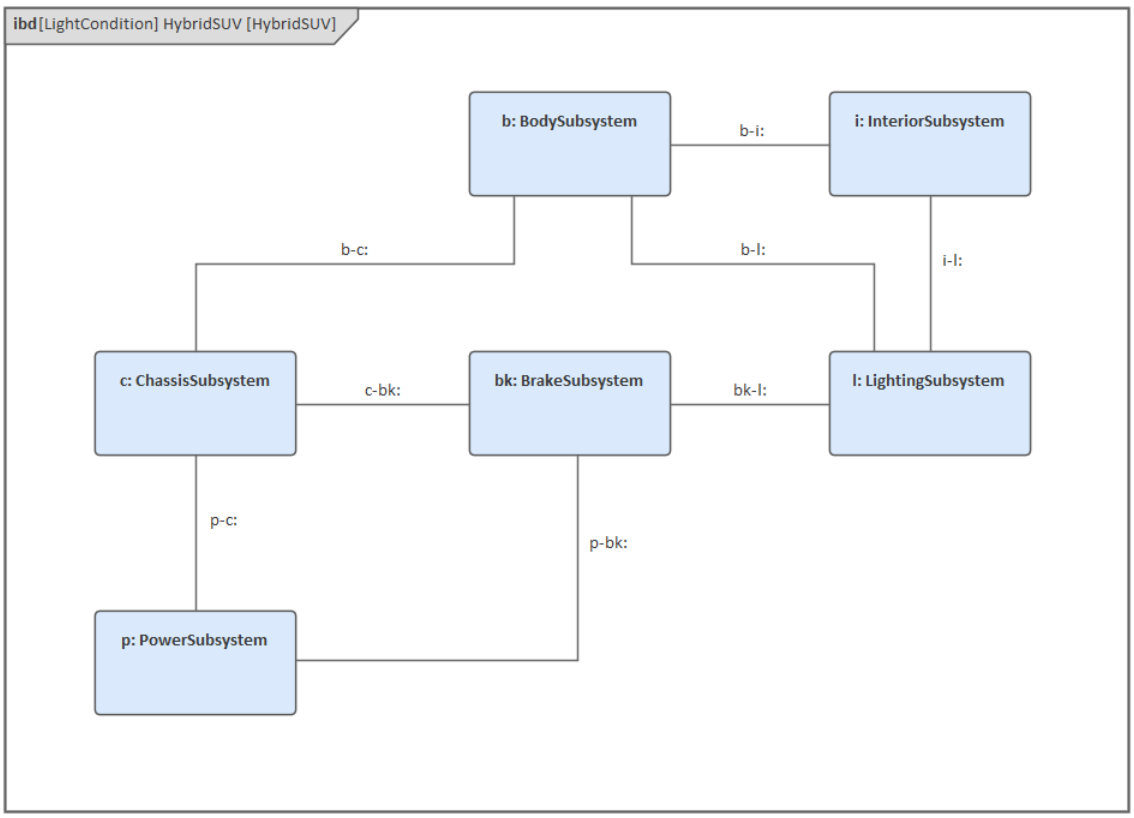


A Block defines a collection (or set) of features that are used to describe a system, subsystem, component or other element of interest. These features can include both structural and behavioral features, such as properties, operations and receptions, to represent the state of the system and the behavior that the system is capable of exhibiting.

Enterprise Architect has a set of tools that help the systems engineer to work with Blocks and to visualize the structure and behavior of these all-important elements in a system's definition. These facilities include the:

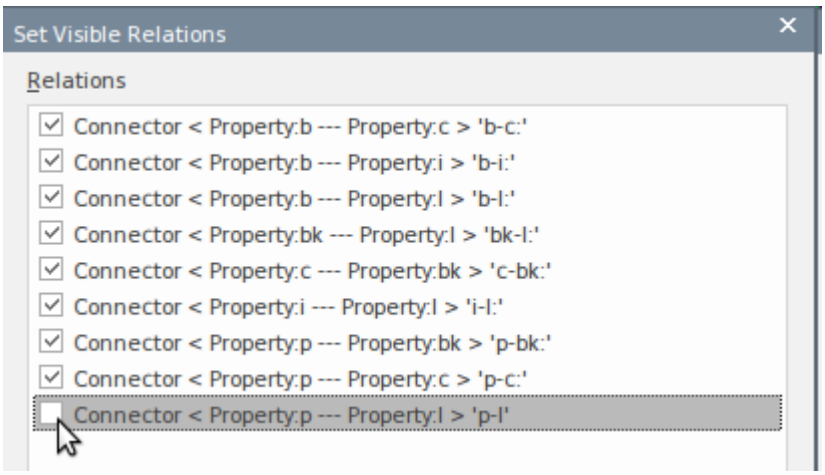
- Block Definition Diagram, which describes the Blocks, their features, interaction points and structural relationships
- Internal Block Diagram, which captures the internal structure of a Block in terms of properties and connectors between properties

This Internal Block Diagram shows how a number of sub-systems cooperate to create the structure of the vehicle. For example the Lighting Subsystem has a connection with the Interior Subsystem which in turn has a connection to the Body Subsystem.



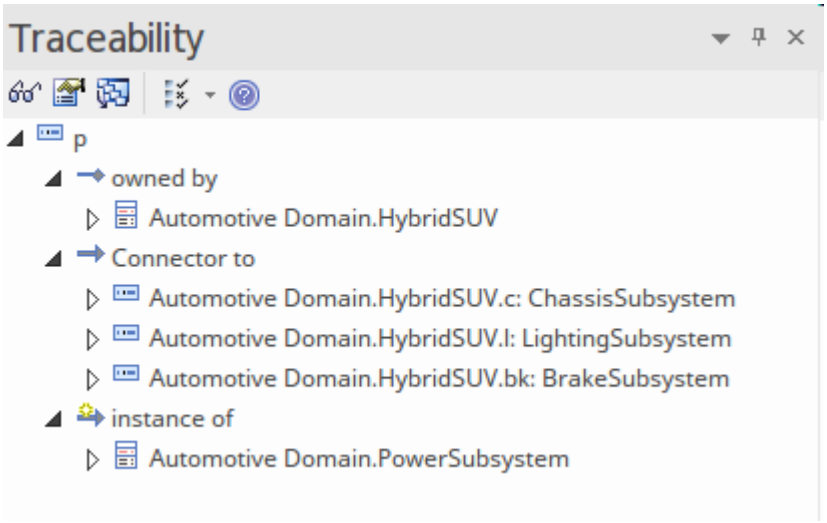
Some relationships have been suppressed in the diagram; for example, the Power Subsystem would typically have a connection to the Lighting Subsystem. This point is important, as newcomers to SysML and Enterprise Architect often think that every defined relationship should be displayed in a diagram. While this statement appears to be true it is important to remember that a modeler, like a cartoonist creating a caricature, will often leave details out of illustrations to focus the viewer's attention on other subjectively more important elements and connectors.

This screen capture shows how an engineer can set the visible relationships for a diagram.



If a connector is not checked in this dialog it will not be displayed in the current diagram. It might, however, be visible in other diagrams where the connected elements are displayed. This can be set from the 'Layout > Diagram > Appearance > Visibility > Set Visible Relationships' ribbon option.

Regardless of which connectors are displayed in a diagram, a modeler can always view all of an element's connectors by selecting the element in the diagram and viewing the Traceability window. In this screen capture the Power Subsystem has been selected, and even though the connector between the Power Subsystem and the Lighting Subsystem has been set to 'not Visible' in the diagram, the relationship can be seen in the Traceability window.



The features of a Block are either structural or behavioral.

The structural features are of three kinds:

- ***Parts*** - that describe the composition of a Block; for example, that a vehicle's chassis is composed of two axles and four wheel assemblies
- ***References*** - that describe the Block's relationship with other Blocks (including itself); for example, that a metropolitan train has a relationship to a station and an overhead wiring system
- ***Values*** - that describe quantifiable aspects of a Block; for example, such things as dimensions, temperature and luminosity

The Behavioral features include:

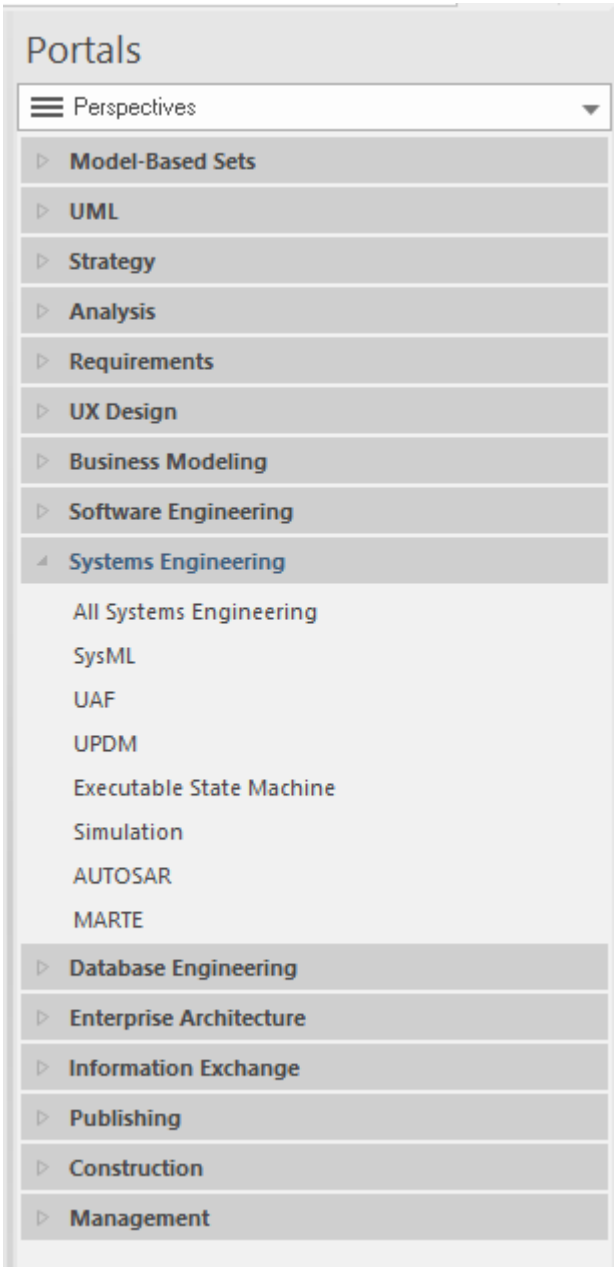
- ***Operations*** - typically representing synchronous requests
- ***Receptions*** - representing asynchronous requests from a signal

Getting Started with Blocks

The designers of Enterprise Architect have created great flexibility for the Systems Engineers and other disciplines, recognizing that team members often perform a variety of roles and need to effectively change hats multiple times within a project, or even within a day. Perspectives and Workspaces provide a convenient and efficient way for a Systems Engineer to effectively change roles without the need to launch another tool.

Setting the Perspective and Workspace

Systems Engineers who have been working with the tool for some time will have selected a Perspective from the Systems Engineering Perspective Set; typically this might be the SysML Perspective, giving them access to all the patterns and Toolbox pages to create any one of the SysML diagrams, including the Block Definition and Internal Block diagrams.



As explained in the introductory topic, workspaces can be set to facilitate modeling of a particular type; in the case of Block Definition diagrams, any one of the workspaces contained in the Core Workspace Set will be useful for modeling, including:

- Basic Diagramming
- Core Modeling
- Default

An Engineer who is working at a project level might also find some of the construction workspaces useful, including:

- Roadmaps
- Kanban
- Document Publishing
- Reviews and Discussions

Creating Block Definition or Internal Block Diagrams

There are two diagrams that you will typically create when working with Blocks:

- Block Definition diagram (BDD) - used to show the structural relationships between Blocks, including hierarchies of both Parts and type, and reference connections to other Blocks
- Internal Block diagram (IBD) - used to show how the Part properties are connected directly, or through interaction points such as Ports and interfaces

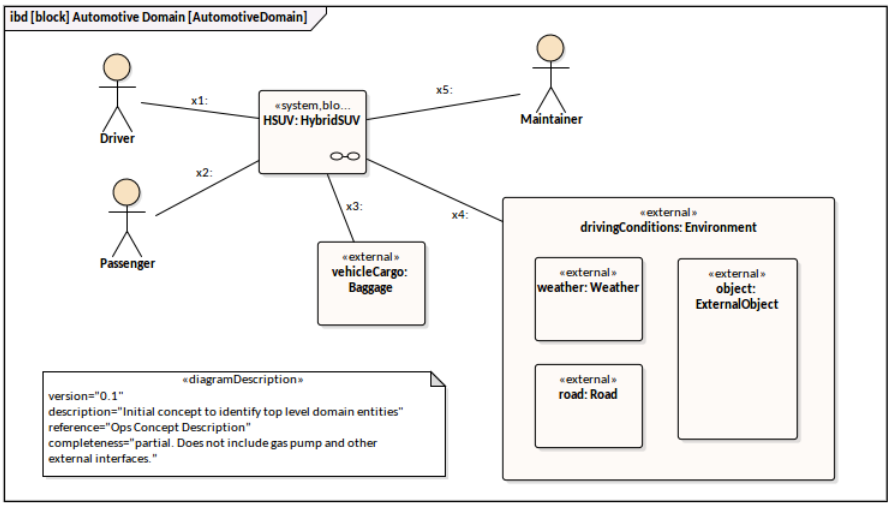
We will discuss the Block Definition diagram in this chapter of the Guide; in the next chapter we will discuss the Internal Block diagram, which will demonstrate how Blocks can be used in a given context.

Internal Block Diagram - Setting Context

One of the most important diagrams to create early in an initiative is a Context diagram, which describes the product or service being modeled in the context of its environment or domain. This helps a model viewer get a clear picture and understanding of how the product sits within one or more of the environments it will need to operate in. It also gives an early indication of what is in scope and out of scope in the project. Elements in the diagram have been marked 'External' (using a stereotype) indicating that they form part of the product or service environment or context.

Images have been used to soften the diagram, making it more appealing to a wide range of stakeholders, including business and non-technical audiences.

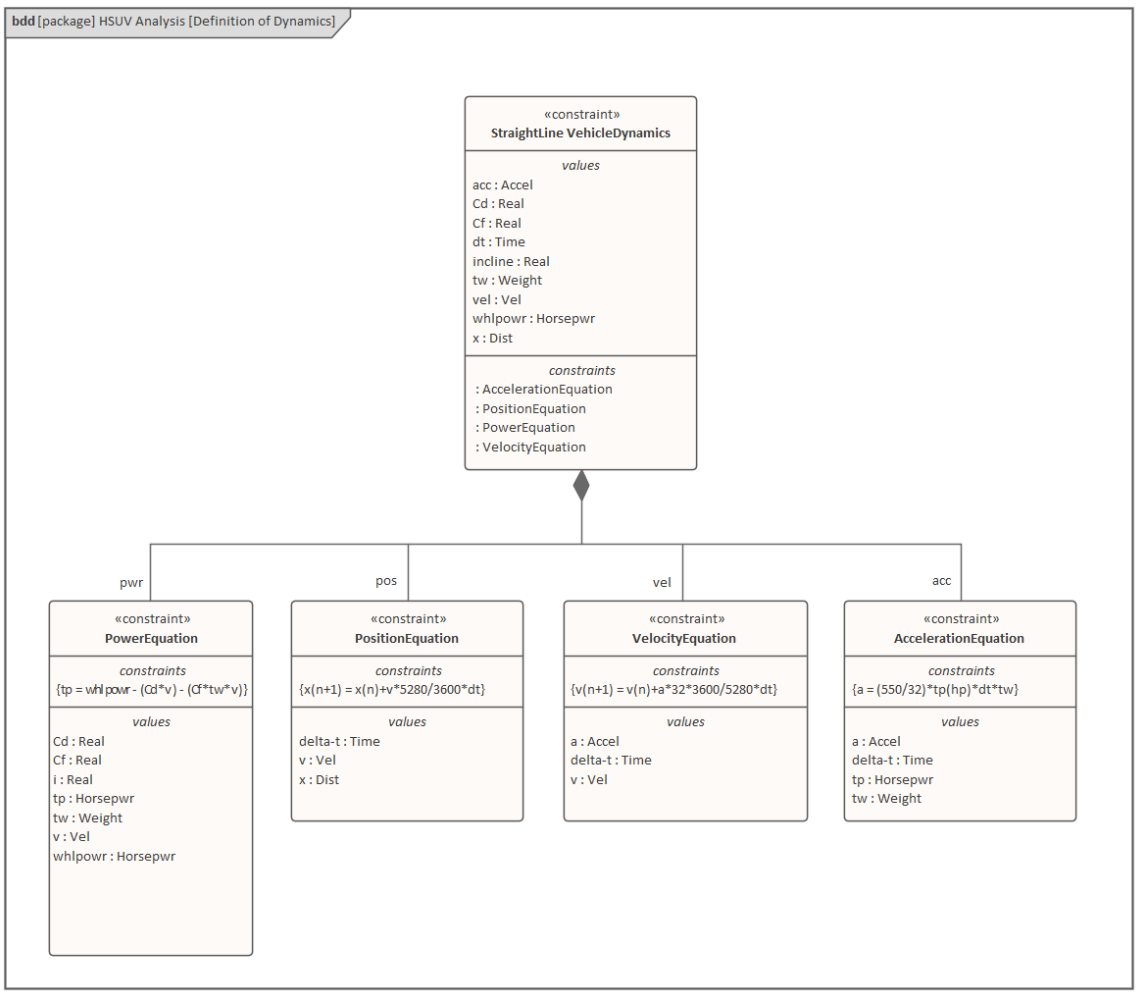
This Internal Block diagram shows the Hybrid SUV in the context of a typical city environment. It is envisaged that at least one other diagram could be created, showing an off-road environment where roads would be replaced by dirt tracks and rivers, and buildings by mountains and trees.


ibd [block] Automotive Domain [AutomotiveDomain]
Driver
x1:
«system,blo...
HSUV: HybridSUV
x5:
Maintainer
x2:
Passenger
x3:
x4:
«external»
vehicleCargo:
Baggage
«external»
drivingConditions: Environment
«external»
weather: Weather
«external»
object:
ExternalObject
«external»
road: Road
«diagramDescription»
version="0.1"
description="Initial concept to identify top level domain entities"
reference="Ops Concept Description"
completeness="partial. Does not include gas pump and other
external interfaces."

Modelling Constraints as Blocks

Engineering Analysis requires the Engineer to perform a variety of functions that include the construction of performance and reliability models, trade-off analysis, alternatives analysis and trade studies. These analyses often require the use of mathematical expressions and equations that are used to constrain the elements of an analysis. SysML provides a language mechanism in the form of a ConstraintBlock that is used to model the equations graphically. This is a useful mechanism that allows the expression to be articulated along with its parameters and their types. The modeled equations can then be reused in a number of different contexts, allowing an Engineer to define the formula for Newton's second law of motion {F=m*a} or Carnot's definition, resulting in a fundamental theorem of thermodynamics {p= W/t =(mg)h/t}. One of the helpful results of modeling these equations graphically is that they can be related to other model elements such a stakeholder's Requirements, mission goals and lower level elements such as Blocks and implementation artifacts.

Enterprise Architect allows these ConstraintBlocks to be modeled and then reused as Constraint Properties on Parametric diagrams. The Constraint definitions can be grouped into libraries, and not only used in the current initiative but reused across multiple projects and initiatives. In a later topic we will see how the constraints can be built up into a network of equations and used on Parametric diagrams to evaluate alternatives, and to conduct trade-off and alternative analysis. The tool's precision and technical excellence will ensure that equations defined in this way can be created, maintained and used with rigor.



Introducing Block Definition Diagrams

Blocks provide a unifying concept to describe the structure of an element or system, including:

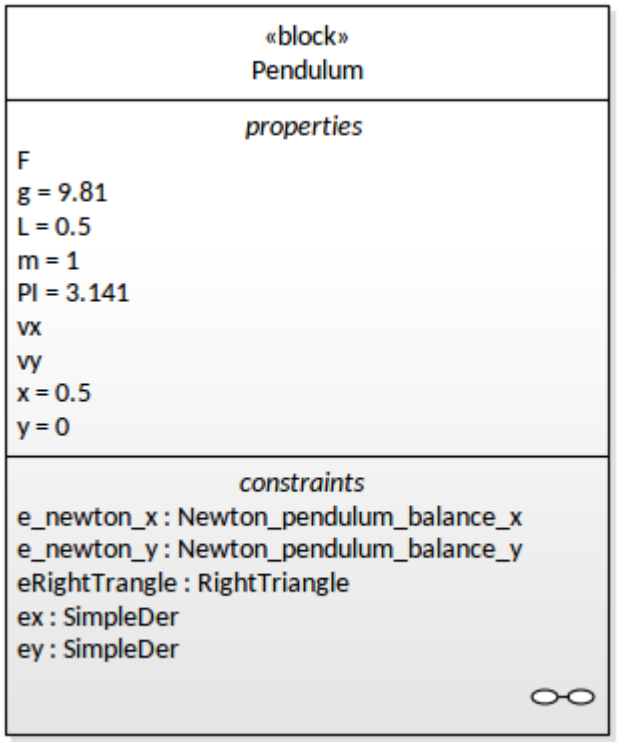
- Systems
- Hardware
- Software
- Data
- Procedures
- Facilities
- People

Blocks can have multiple standard compartments that can be made visible on diagrams to describe the Block's characteristics, including:

- Properties (parts, references, values, ports)
- Operations
- Constraints
- Allocations from/to other model elements (such as Activities)
- Requirements that the Block satisfies
- User defined compartments

Any of the compartments can be suppressed. A separator line is not drawn for a missing compartment. If a compartment is suppressed, no inference can be drawn about the presence or absence of elements in it. In this diagram a Pendulum has been modeled and a number of compartments have been made visible in preparation to create a parametric simulation.

-



Additional compartments can be supplied as a tool extension to show other predefined or user-defined model properties (for example, to show business rules, responsibilities, variations, events handled, raised, and so on).

Each Block must have a non-null name that is unique within its namespace. The scope of a name is its containing Package and other Packages that can see the containing Package.

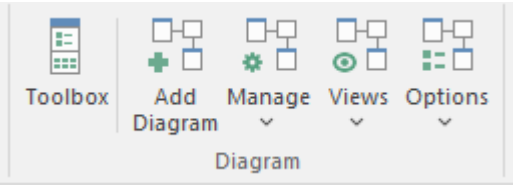
Creating a Block Definition Diagram

A Block Definition diagram can be created from a number of places in the User Interface by using any of these options:

- Design ribbon - *Add Diagram* icon on the *Diagram* Panel
- Browser window Toolbar - *New Diagram* icon
- Browser window Context Menu - *New Diagram*

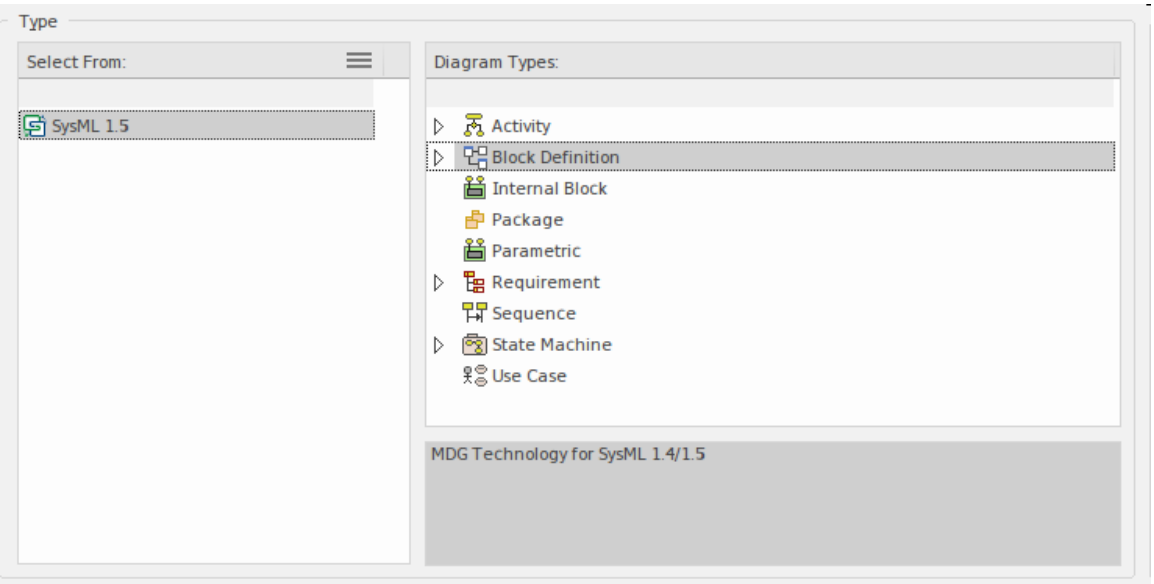
We will use the Design ribbon to create a Block Definition diagram. Firstly, select the location in the Browser window where you want the Block Definition diagram to be located. As with all diagrams, this can be under either a Package or an element, but it is common to insert Block Definition diagrams into a Package. Once the location has been selected in the Browser window, select the ribbon option:

Design > Diagram > Add Diagram



Selecting this option will open the 'Diagram Builder' tab page of the 'Model Builder' dialog, where you can choose the diagram type and specify a name for the diagram. The name initially defaults to the name of the Package or element that contains the diagram. With the SysML perspective chosen and the version of SysML selected, a list of diagram types will be displayed; select the Block Definition diagram and click on the 'Create Diagram' button. A new Block Definition diagram will be created in the location selected in the Browser window. The Diagram View will be opened allowing you to start adding elements and connectors that describe the Blocks and other important structural elements such as Ports,

Interfaces and Value Types. Enterprise Architect will also display the 'Block Definition' pages of the Diagram Toolbox, which contain the elements and relationships defined by the SysML specification as applicable for constructing Block Definition diagrams. Any number of other Toolbox pages can be opened if required, in addition to the 'Common' elements and 'Common Relationships' pages that will always be available.



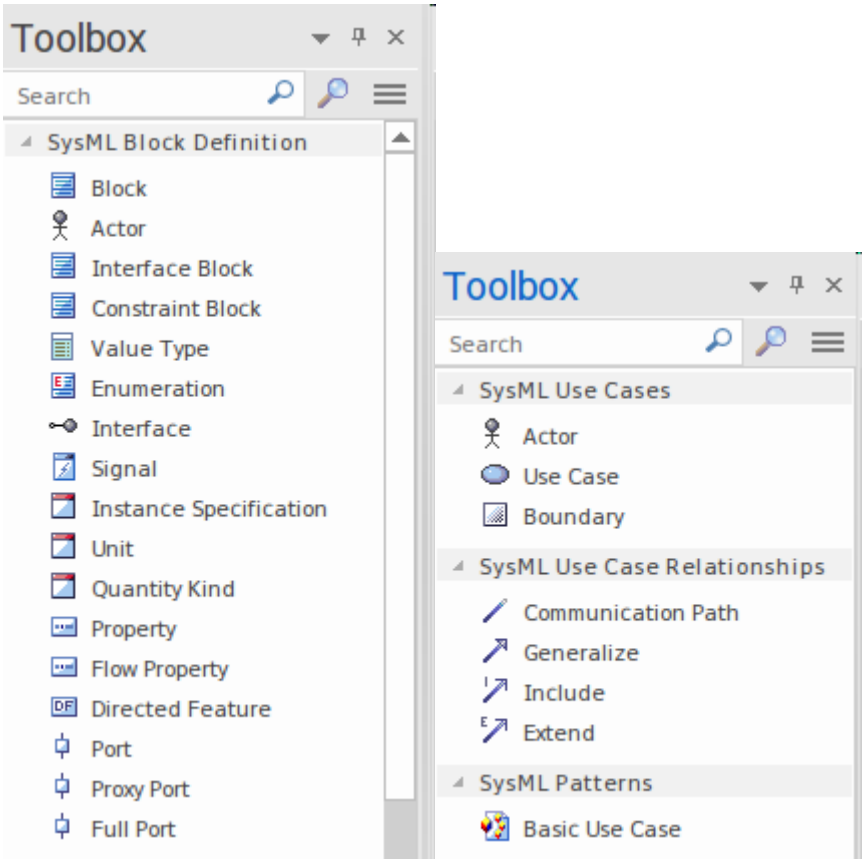
The most import elements and connectors that are used with the Block Definition diagram are:

Elements

- Block
- Constraint Block
- Value Type
- Property
- Unit
- Quantity Kind
- Proxy Port
- Full Port

Connectors

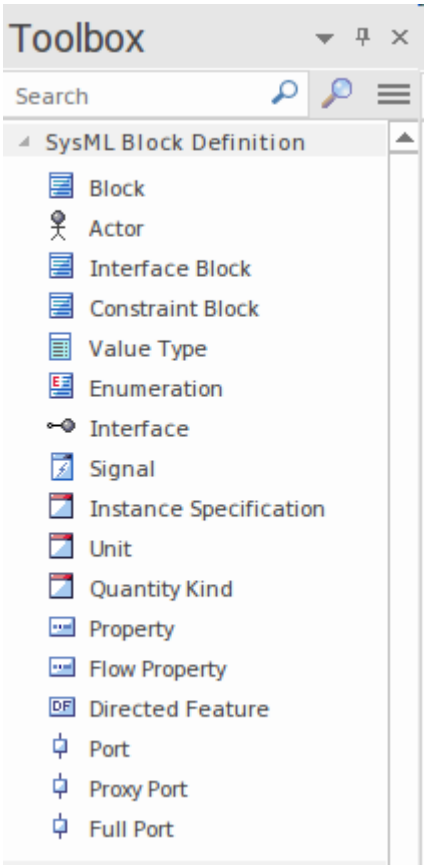
- Reference Association
- Part Association
- Shared Association
- Generalization
- Dependency
- Item Flow



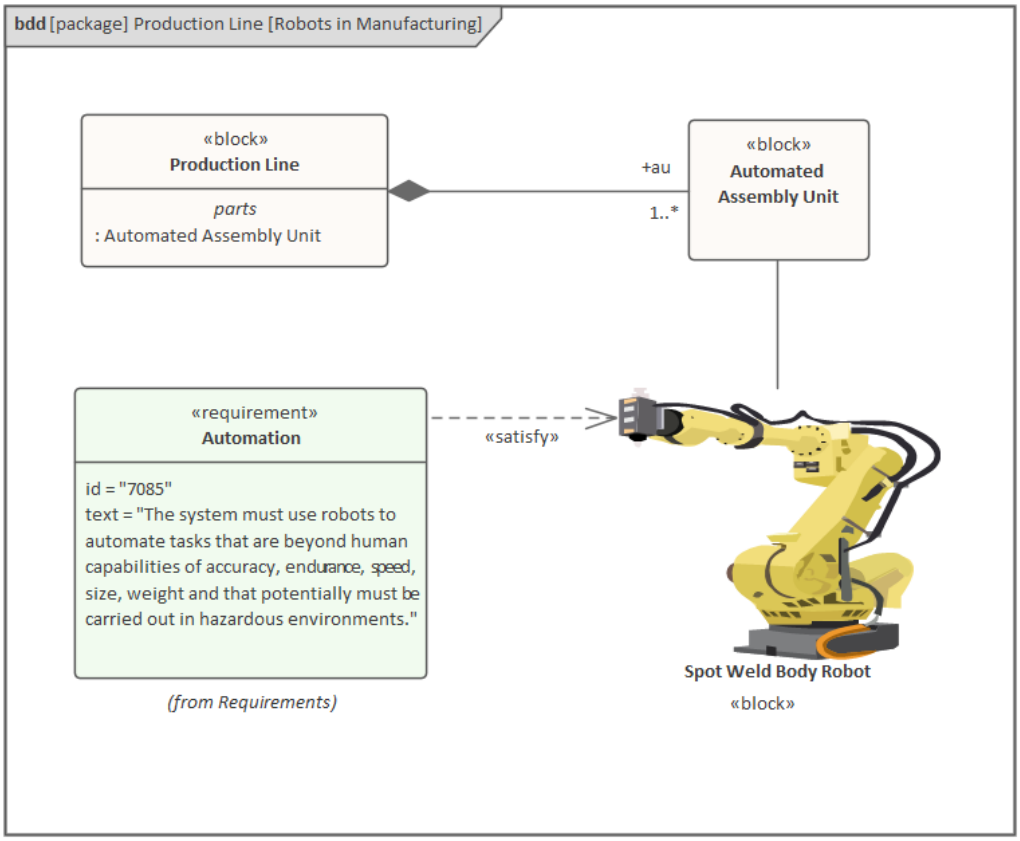
Elements can be added to the diagram by drag-and-dropping them from the Toolbox onto the Diagram View. For more information see the Block Definition Diagrams (BDDs) Help topic.

Creating a Block Element

Block elements, as with any other type of element, can be created using the 'Add Element' option on a Package context menu, or by using the 'SysML Block Definition' page of the Diagram Toolbox to place a Block on a Block Definition diagram (BDD).



It is common for Blocks to appear on multiple BDDs, where each diagram is designed to address the concerns of a particular stakeholder or stakeholder group. Enterprise Architect has a wide range of display options both at the level of individual Blocks (or any element) or at the level of the diagram. These can be used to decide, for example, which compartments to display or even which features to display for individual elements, There is also a wide range of generic element and diagram settings to style both the element and the diagram. For example, it is possible to set the element colors including fill, borders and text, or to change the appearance of an element by applying a graphical image that better conveys the Block's function. In this example, a modeler has decided to use an alternative image for a Spot Welding Robot to convey more clearly the automation taking place on the production line.

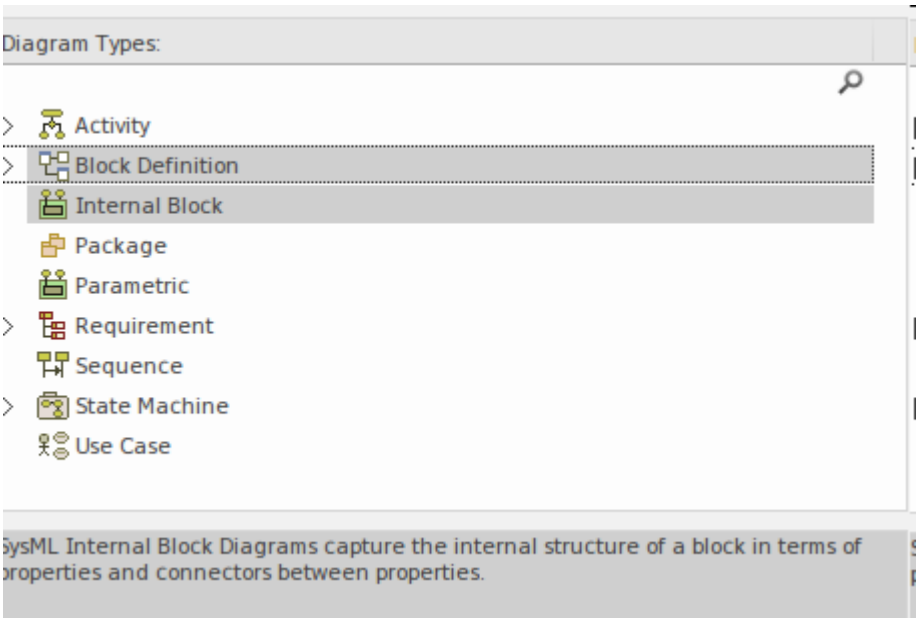

bdd [package] Production Line [Robots in Manufacturing]
«block»
Production Line
parts
: Automated Assembly Unit
+au
1..*
«block»
Automated
Assembly Unit
«requirement»
Automation
id = "7085"
text = "The system must use robots to automate tasks that are beyond human capabilities of accuracy, endurance, speed, size, weight and that potentially must be carried out in hazardous environments."
(from Requirements)
«satisfy»
Spot Weld Body Robot
«block»

The Fundamental Structural Building Blocks

Blocks are the fundamental and discrete modular units of system description. A Block defines a collection of features that are used to define an aspect of a system or a system itself. The features are of two fundamental types: structural (what a Block consists of) and behavioral (what it does) features. A Block's relationships with other Blocks (including itself) and with elements of other types, help to describe the structure of a system, subsystem or component.

System modelers use Block Definition Diagrams (BDDs) to define the structure of Blocks, and Internal Block Diagrams (IBDs) to describe their usage.

These diagrams can be created from the 'New Diagram' dialog, accessible from the Browser window toolbar.



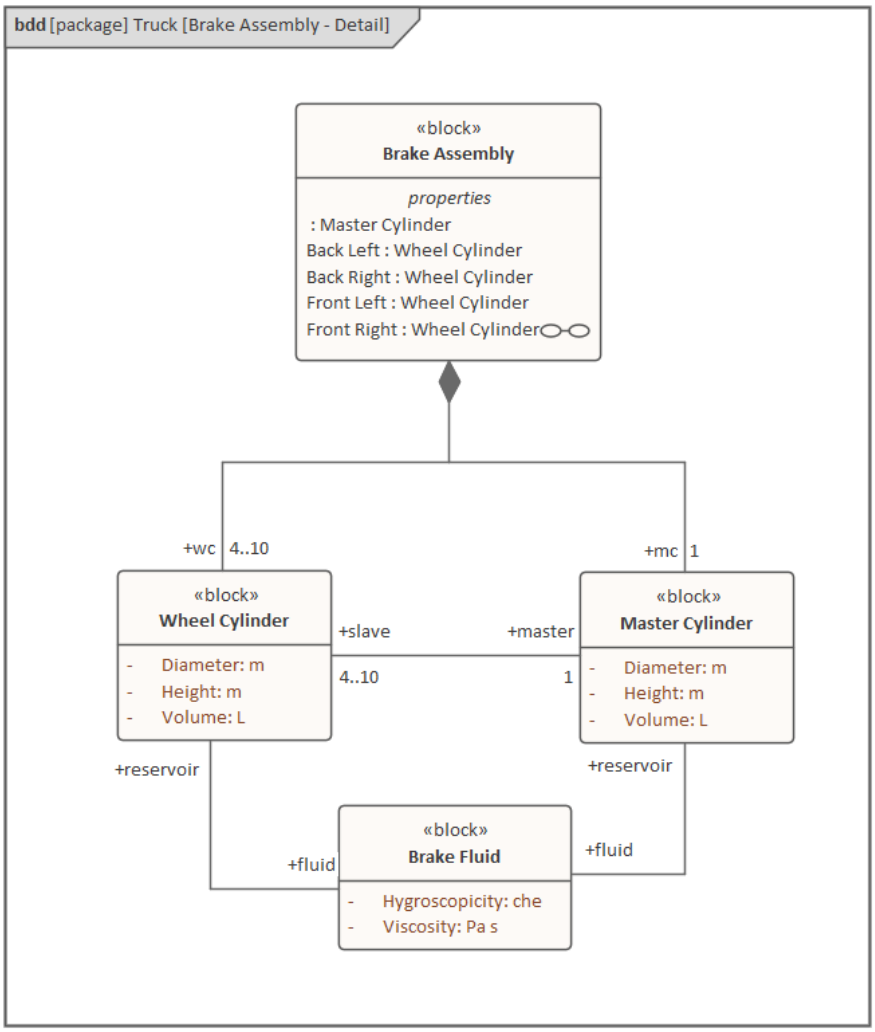
Modeling Structural Features

Blocks typically are defined using a series of structural features. These are the properties of the Block and define the nature of the Block. For example, a train engine (Rolling Stock) will have properties such as Engine Class, Identity Number, Number of Wheel Assemblies, Motive Force, Motors and a range of other properties. An important point to remember is that the Block is a classifier that describes a set of Engines. The engine at the front of the train set that you board for your summer holiday is an instance of an engine and it will have a particular Class, for example OSE class 660, and an identifier of SM-09873, and 8 wheel assemblies.

Enterprise Architect supports three basic kinds of structural feature and each is important for modeling different aspects of the structure of a Block. We will look at each of them in these sections.

- Parts - *a block is composed of parts*
- References - *refer to features of other blocks*
- Values - *describe quantities*

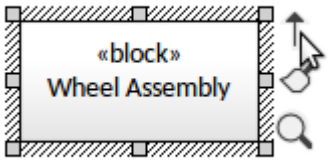
This diagram shows all three types of structural feature.



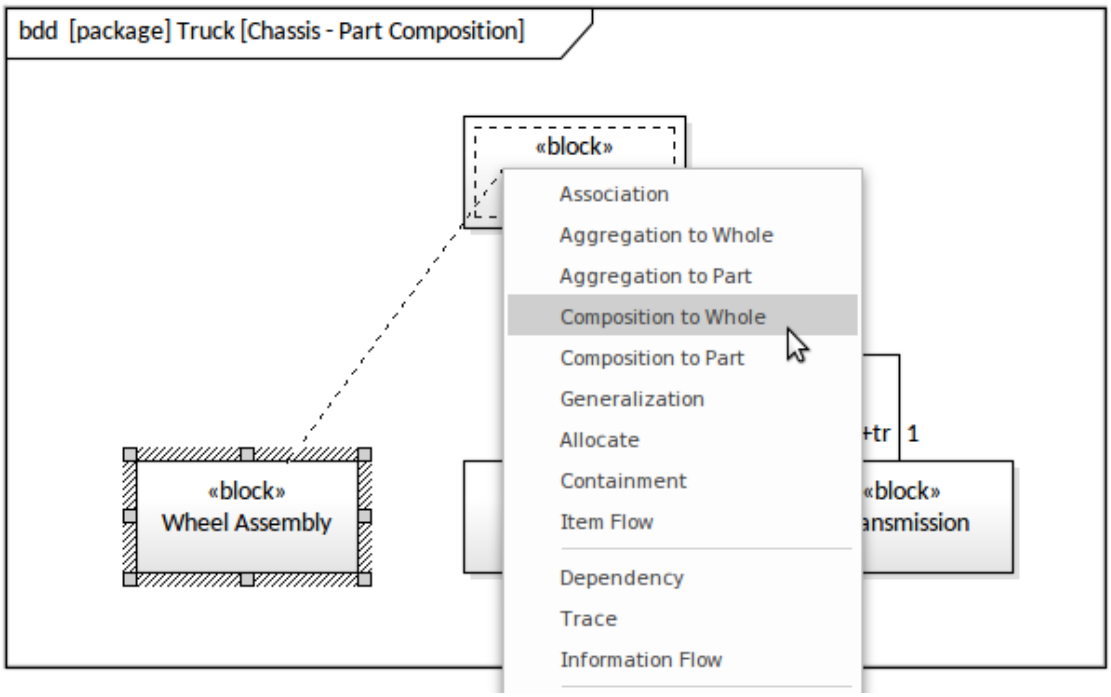
The Braking System is made up of a number of parts, two of which have been shown on this diagram. The Part Association has been used to indicate that the *Master Cylinde*r and *Wheel Cylinder*s are fundamental constituents of the braking system. A Reference Association has been used to show both a relationship between the two types of cylinders and also between the cylinders and the *Brake Fluid.* Values that have been entered as attributes are displayed with their accompanying Value Types; for example, Volume has a Value Type of L, which is the symbol for the Dimension of Volume whose SI Unit is Litre.

Blocks Composed of Parts

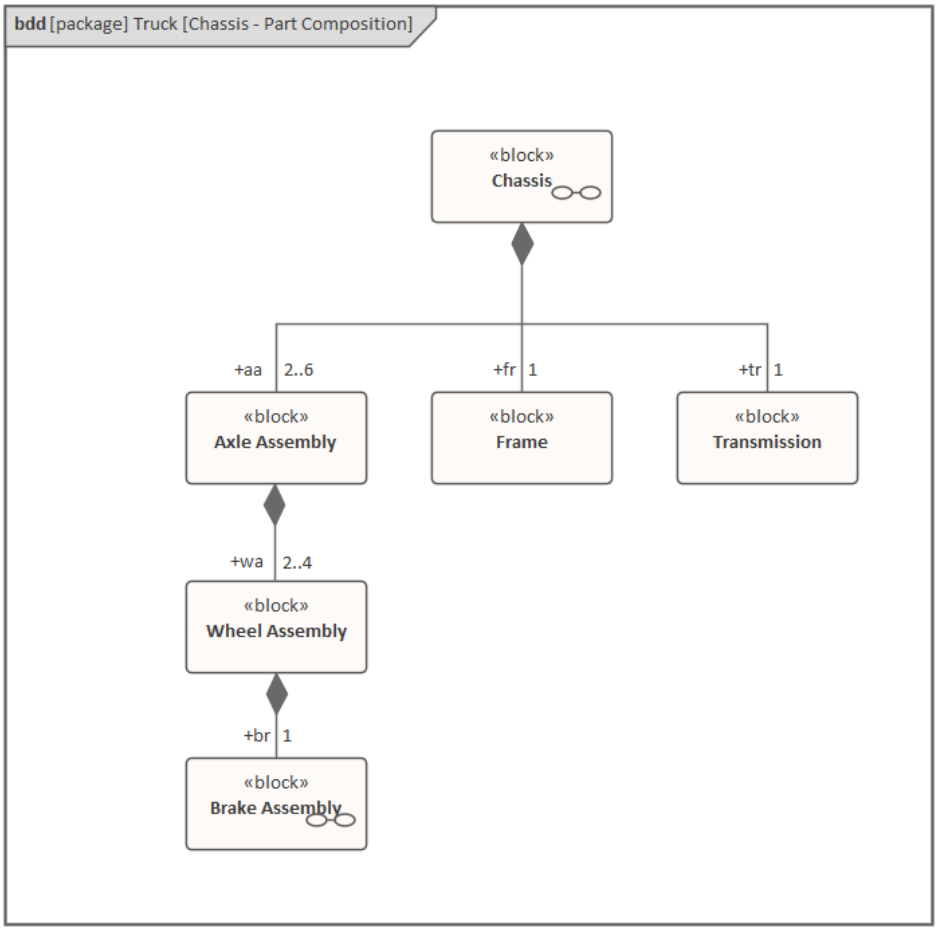
A Part is a structural Feature of a Block and forms one of the strongest relationships between a Block and its properties. It is important to understand that an Instance of a Block might have multiple instances of a Part; for example, a truck might have multiple wheel assemblies and - depending on the size and type of truck - this could be as low as 2 or as high as 10. These possible configurations can be specified in the definition of the Block and its Parts, which are formally known as multiplicities - the lower number is referred to as the lower bound and the higher number as the upper bound. A Part will typically be typed by another Block, thus in the example the type of the Part will be another Block named 'Wheel Assembly', which would typically itself comprise an axle and two wheel assemblies. Thus each Part will be defined in the Block with a name, a type and a multiplicity. The tool allows the *Part Composition* relationship to be created in a number of ways, but perhaps the most immediate way is to drag both the *Chassis* Block (the whole) and the *Wheel Assembly Block* (the Part) onto the diagram and then use the Quick Linker to drag from the Part (*Wheel Assembly*) to the whole (*Chassis*).



Dragging from the source object to the target will display a menu of possible connectors, and the engineer would choose the *Composition to Whole* connector. The result will be a relationship with the diamond marker at the *Chassis* end of the line, indicating it is the whole and the element at the *Wheel Assembly* end is the Part.

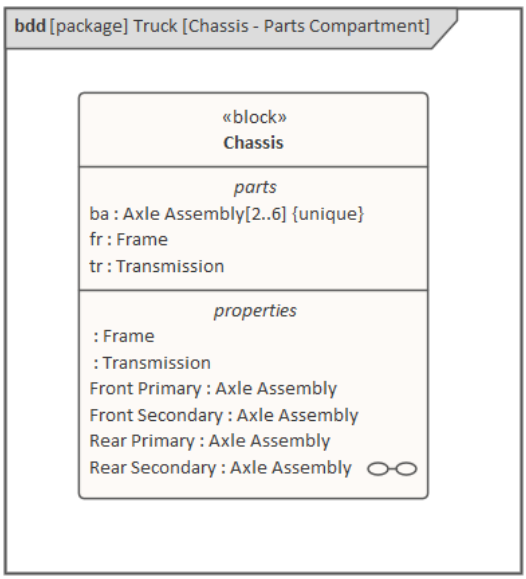


The connector properties will allow you to set the source role and multiplicities which, as discussed, specify the name and the possible number of Parts for each Instance of a *Chassis*.



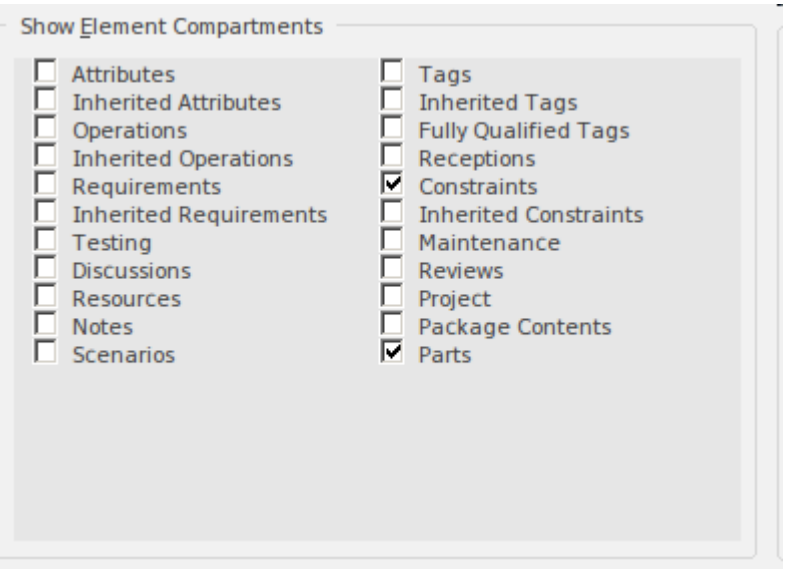
In the diagram the modeler has expressly defined the Parts by using the Part Association, available from the SysML Block Definition toolbox.

In this diagram the modeler has used the Owning Block's Part compartment to display the Parts owned by the Chassis Block.

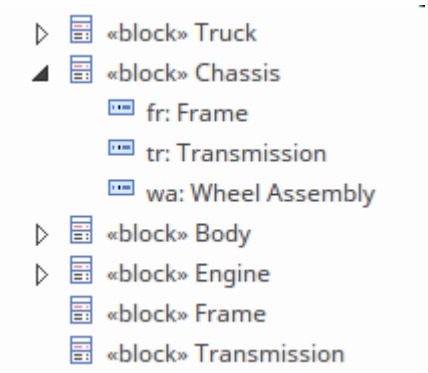


The Parts compartment will display by default, but its visibility can be controlled at a diagram level using the diagram Properties, or at an individual element level using the element's 'Compartment Visibility' option on the element's context menu. Setting the visibility at the diagram level will result in all elements in the diagram complying with the specified visibility - displayed or not displayed as specified - whereas setting it at the element level will only affect the selected

element.



The repository elements will be updated regardless of whether they are edited in the diagram or the Browser or any other window. In the example, the engineer created the Parts in the diagram by dragging a Part Association from the Toolbox; in response to this Enterprise Architect creates three new Parts, which are placed under the Chassis node in the Browser as indicated in this screen shot.



The Part Association is the strongest type of Association relationship - the strength continuum from weakest to strongest being:

1. Reference Association
2. Shared Association
3. Part Association

We will explore the other relationships in later sections of this guide.

References to Other Blocks

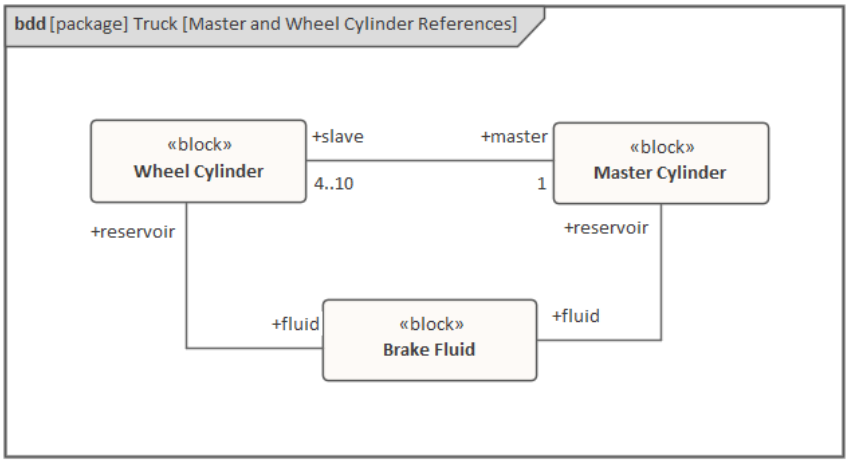
As stated earlier the Part Association is the strongest type of relationship in the SysML and implies a sense of responsibility on the part of the whole:

- It is responsible for the lifetime of its parts from which it is comprised
- A part can only participate in a part composition with a single block

The second condition means that the multiplicity at the whole end of a Part composition is always 1..1 which can be

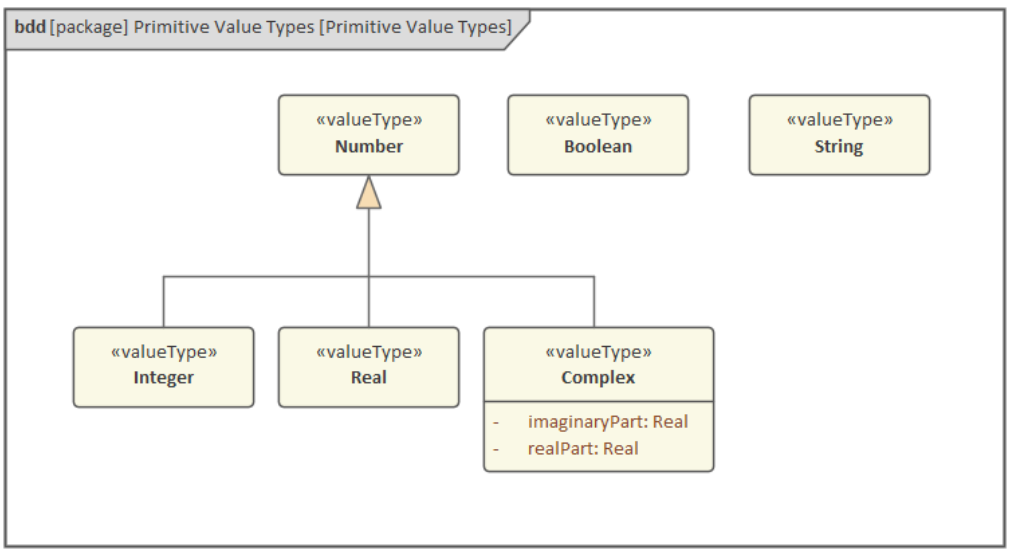
abbreviated as 1.

There is however another relationship, the Reference Association (or Reference for short) that can be used to specify relationships between Blocks independent of composition or the notion of one block being a part of another. This provides a very useful mechanism for creating relationships between blocks that are part of different part hierarchies or between any two blocks that are related to each other. For example the Master Cylinder and Wheel Cylinders both have a relationship to Brake Fluid which is used to fill their reservoirs. The Wheel Cylinder could in turn have a relationship to a mechanic that periodically checks the cylinder for leaks that would compromise the efficacy of the braking system.



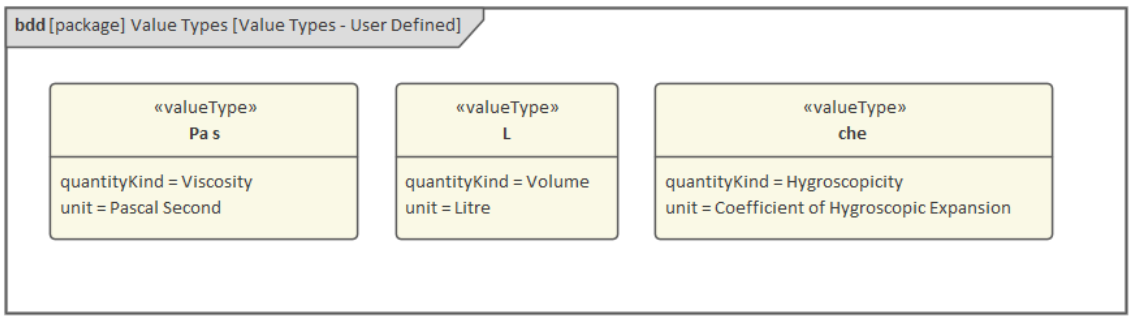
Values used to Describe Quantities

Blocks can have properties with some type of quantifiable value; for example, an Engine has a Power Output, a Reservoir has a Volume, an Automobile has a Color, a Railway carriage has a number of Bogies. The types can be a primitive type defined as Number, Integer, Real, Complex, Boolean or String, as illustrated in this diagram.

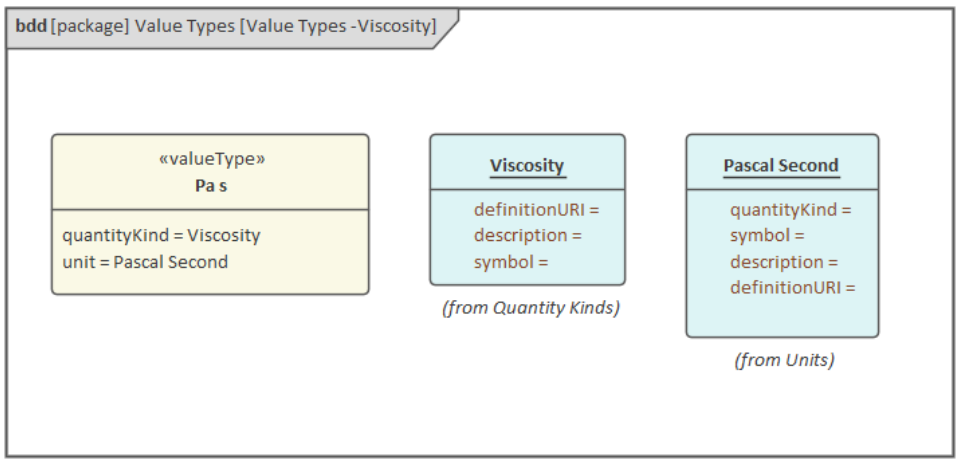


An engineer, team or community of practice can also define any number of Value Types that can be simple or structured. These can be based on any number of Systems of Units, such as the International System of Units (SI). An automotive engineer designing a braking system might find themselves using a number of Standard SI Value Types and a number of

Derived Types, as well as other Values not defined as part of that standard. This diagram illustrates how these Values can be defined, using the Value Type element available in the SysML Block Definition Toolbox.



The Value Type has two defined properties - the quantityKind and the Unit. These can also be modeled in Enterprise Architect and give rigor to the application of the Value Type. An engineer will know that the type is based on a quantity (dimension) and a defined Unit. This diagram shows these elements for the (Viscosity) Value Type.



Modeling Behavioral Features

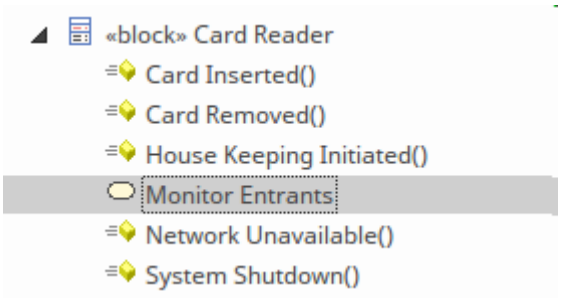
When a system is in operation, Instances of the Blocks that have been defined as part of the architecture and detailed design are instantiated. At this time if a Block has a Classifier Behavior defined this behavior will typically begin and will continue operating until the Block is destroyed. Thus in the example of our Car Park System, when the system has been activated the Card Reader will begin operating and its prime behavior will come into effect. In addition to this a Block (even though fundamentally structural in nature) has behavioral features that will be called upon to carry out work. In summary, there are two fundamental definitions of behavior that are defined within the context of a Block, namely:

- Classifier Behavior - *the native behavior that is initiated when a Block is instantiated*
- Behavior Features - *these are the Operations and Receptions (and their related Signals)*

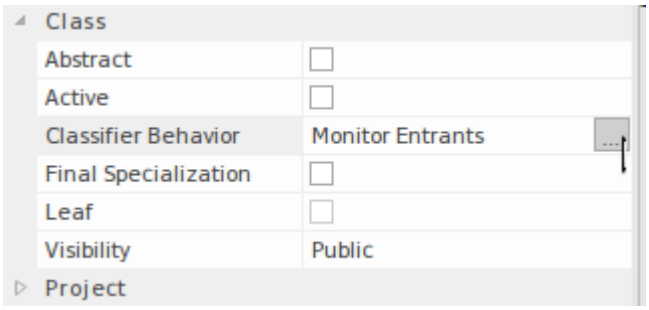
We will look at these different behaviors in the next sections of the Guide, but it is important to understand that they will work in unison, coordinated by system interactions that will ensure that the operations are called in sequence and that the Signals are received and acted upon by the Receptions.

A Blocks Classifier Behavior

A Block has the potential to do work, but by itself it is a somewhat latent entity and needs to be commanded into action by some type of call to its operations or by the receipt of a signal, state change or other behavioral trigger. A Block has a concept of its native or classifier behavior, as it is formally called. This diagram shows a Block in the Browser window that has a nested Activity that will be defined as the Classifier Behavior for the Block.

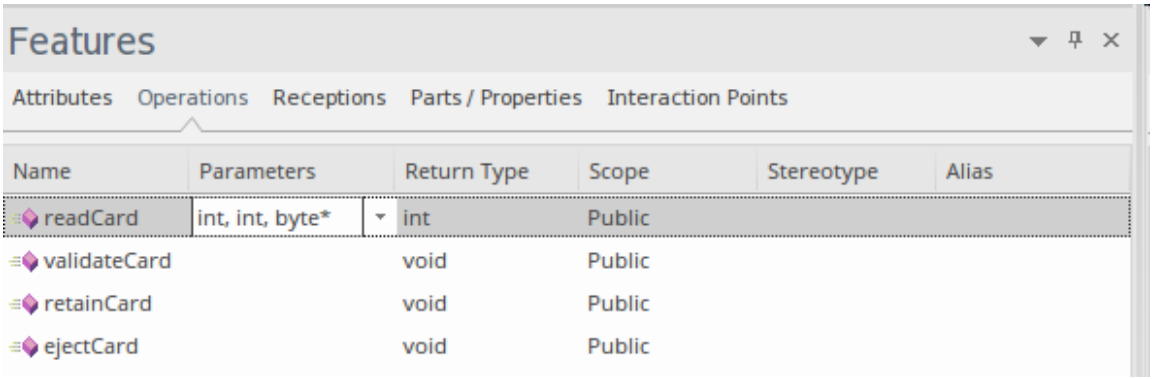


To select this behavior for the classifier behavior, open the Properties window and change the Classifier Behavior property by selecting the [...] icon and locating the appropriate Behavior (Activity) as indicated in this illustration.



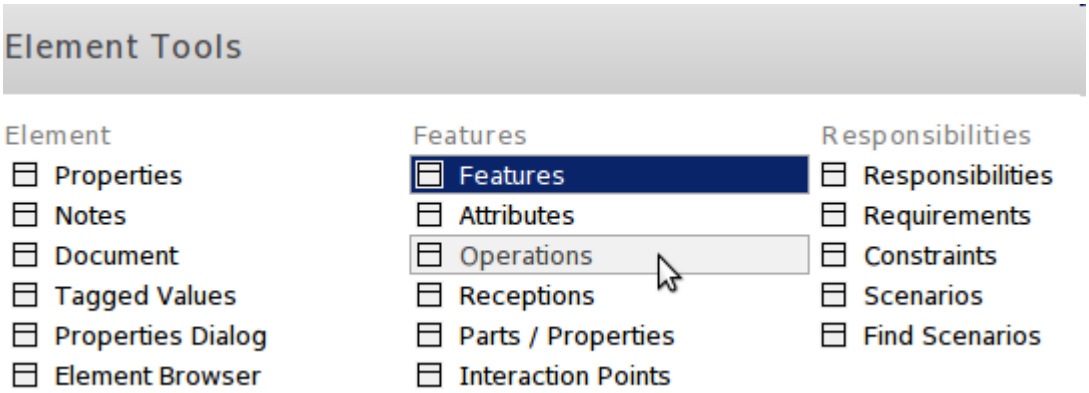
Operations as Behavioral Features

Blocks can define operations essentially as the 'muscles' of the Block, as it is the operations that do most of the work required of the system. In Enterprise Architect an engineer can access the operations from a number of points in the user interface, but all of these points will open the Features window, which lists the operations on the 'Operations' tab as shown here:

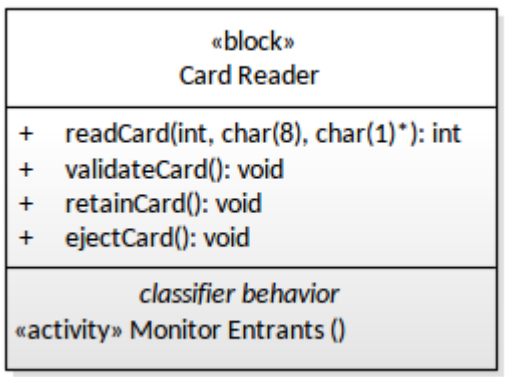


The Features window is useful as a summary of all the structural and behavioral features, including Parts and Interaction Points owned by the Block. The easiest way to create an operation is to select the Block in a diagram or in the Browser window and click on the ribbon item:

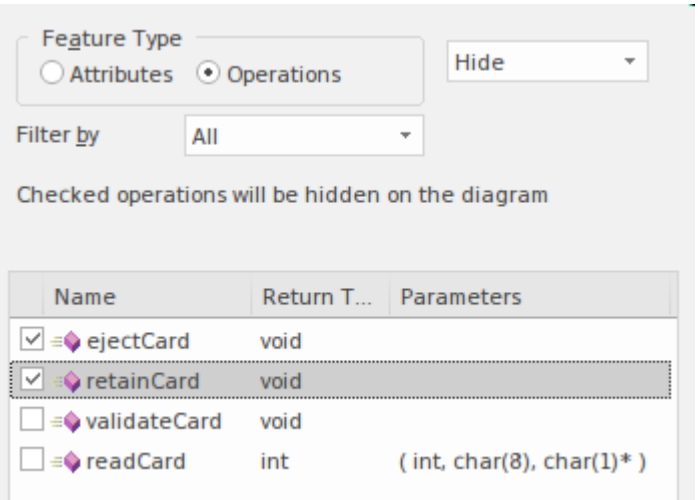
Design > Element > Editor > Features > Operations



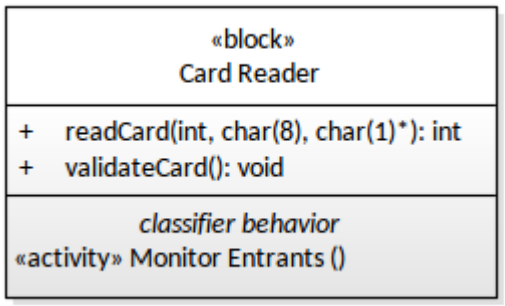
Operations are simply created by selecting the 'Operations' tab and adding the name and other details in a row of the window. Any number of operations can be created, and each operation can define any number of parameters, which specify the inputs and outputs to the operation. Their importance will be discussed later in this section when we describe the relationship between Activity Parameters and Action Pins. Operations can also be displayed in a diagram, either on their own or with other features, each type of which is displayed in a separate compartment of the parent element.



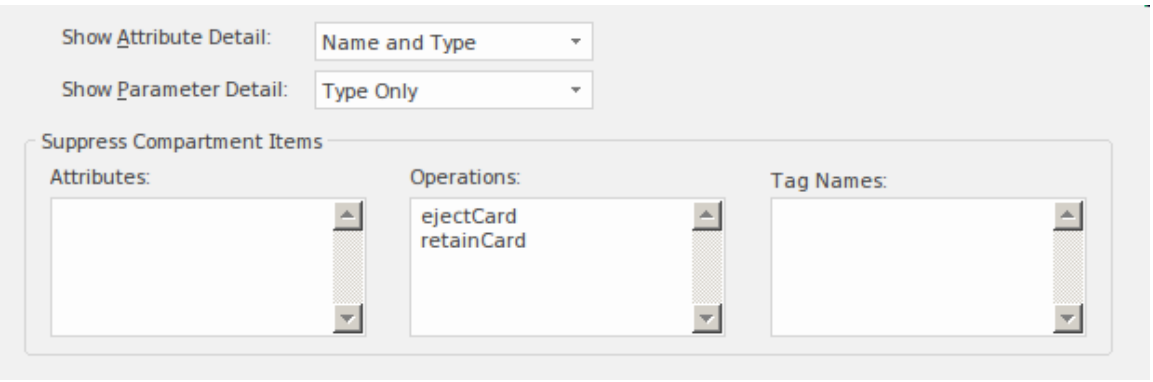
There is a wide range of options that govern how the operations are displayed, including the ability not to display the entire compartment or to only display particular operations by suppressing others from display.



This will result in selected operations being hidden on the diagram, which is a very useful presentation device as it helps an engineer create a diagram focused on a particular aspect of the Block, suppressing or hiding irrelevant and distracting content. This diagram fragment shows the result of suppressing operations:



The same can be done for attributes at an element level, and a similar function is available to suppress particular operations, attributes and Tagged Values at a diagram level. An engineer might use the diagram-level function when there is a particular operation that appears on multiple Blocks and they want to suppress it for every element in the diagram.



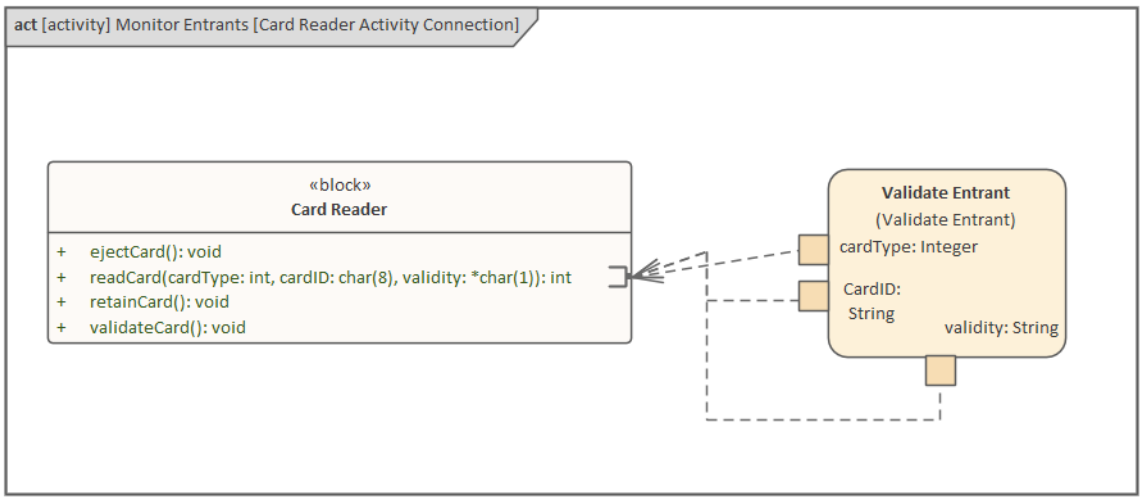
Operations can be invoked in two modes, either synchronously or asynchronously, and can be initiated in a number of different ways depending on the type of behavior that is orchestrating the systems behavior, including:

- A Call Operation Action (invocation of an Activity)
- A Message as part of an Interaction (Sequence diagram)
- A StateMachine

This means that the operation can be visualized in a range of SysML diagrams and will appear differently in different contexts. For example, in a Sequence diagram where messages are sent between instances of Blocks or other classifiers, the operation will appear as an annotation to one of the Block's incoming messages to show that the operation will be initialized as a result of the message. Enterprise Architect allows an engineer to access the Block's operation list directly from this diagram and will also allow operations to be created directly from the diagram.

Usage Type	Diagram Type	Diagram
Classifier	Sequence	Card Reader and Control Unit
Classifier	Sequence	Interactions
Classifier	Sequence	Interactions Basic
Link	SysML Block Definition	Verify Entrant
Link	SysML Block Definition	Boom gate Dependencies
Link	Activity	Card Reader and Controller

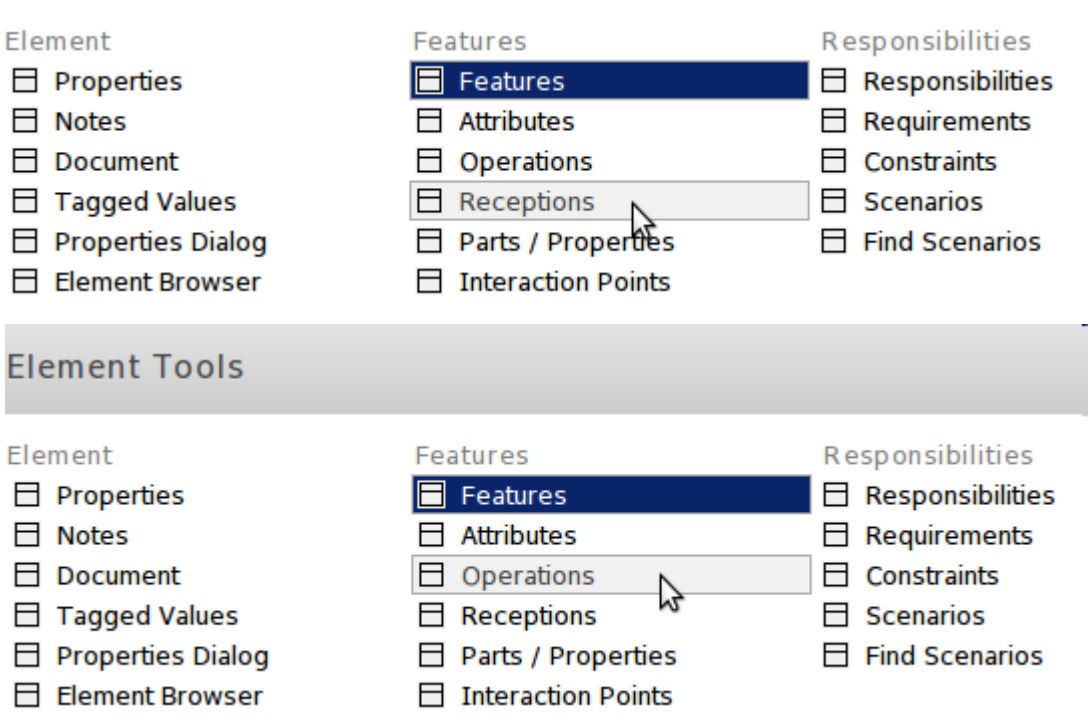
In the case of the Call Operation Action, the element's Pins must be aligned by type and name to the called operation's parameters; Enterprise Architect helps you visualize this mapping on a diagram, using the 'Link to Feature' facility.



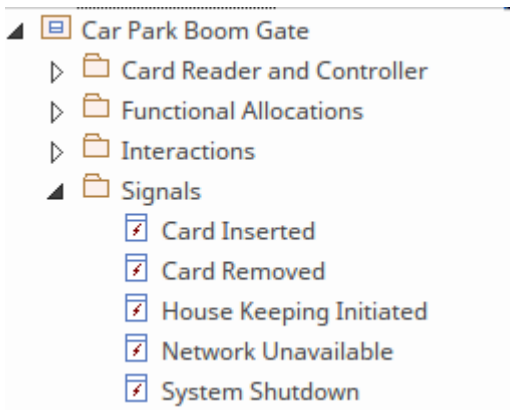
Receptions as Behavioral Features

Receptions are another behavioral feature of a Block but, in contrast to an operation, Receptions can only be called asynchronously. Receptions also work differently to operations in that an Operation Call specifically identifies an operation to be invoked, whereas the receipt of an instance of a Signal is deemed to be a request for any Reception of the receiving object that references that Signal or any direct or indirect generalization of it. In this way there is a level of indirection between the calling element and the Reception. A Reception has parameters corresponding to the attributes of the Signal referenced by the Reception, and these are considered as 'in' parameters of the Reception.

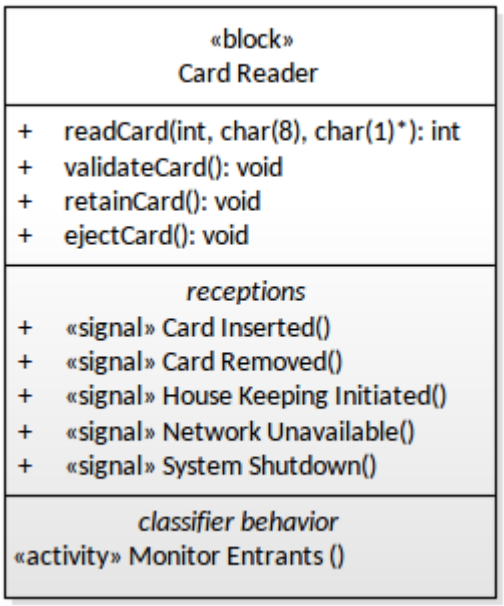
The easiest way to create a Reception is to click on the Block in a diagram or in the Browser window and select the ribbon item 'Design > Element > Editor > Receptions'.



To create a new Reception you must first have created the appropriate Signal to relate the Reception to. When you create the Reception you will be prompted to locate the appropriate Signal in the Browser window as shown here:



Receptions, like operations, can be displayed in a specialized compartment in a Block on a diagram. It is possible to customize the display and suppress all Receptions or configure which particular Receptions are displayed. In this screen capture the engineer has decided to make all Receptions visible, but each diagram and each Block within a diagram can be configured differently.

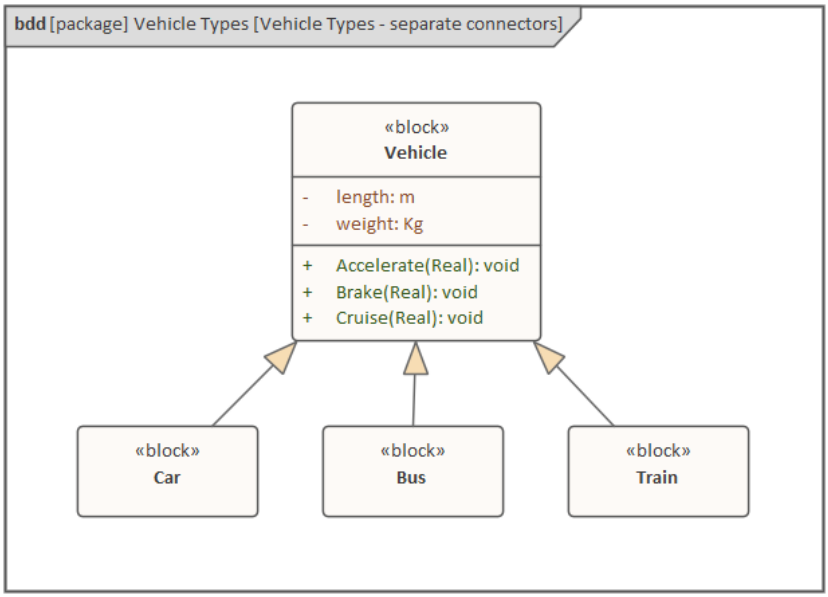

«block»
Card Reader
+ readCard(int, char(8), char(1)*): int
+ validateCard(): void
+ retainCard(): void
+ ejectCard(): void
receptions
+ «signal» Card Inserted()
+ «signal» Card Removed()
+ «signal» House Keeping Initiated()
+ «signal» Network Unavailable()
+ «signal» System Shutdown()
classifier behavior
«activity» Monitor Entrants ()

Other Block Relationships

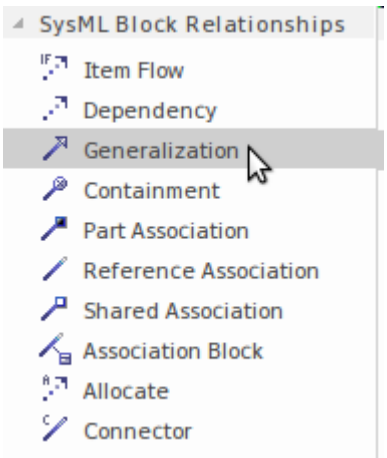
Blocks are the key structural elements in the SysML and can participate in a variety of relationships, some of which have been discussed in earlier sections of the Guide while we were discussing Associations. There are a number of other relationships that can be used when defining Blocks.

Generalization a Relationship of Family

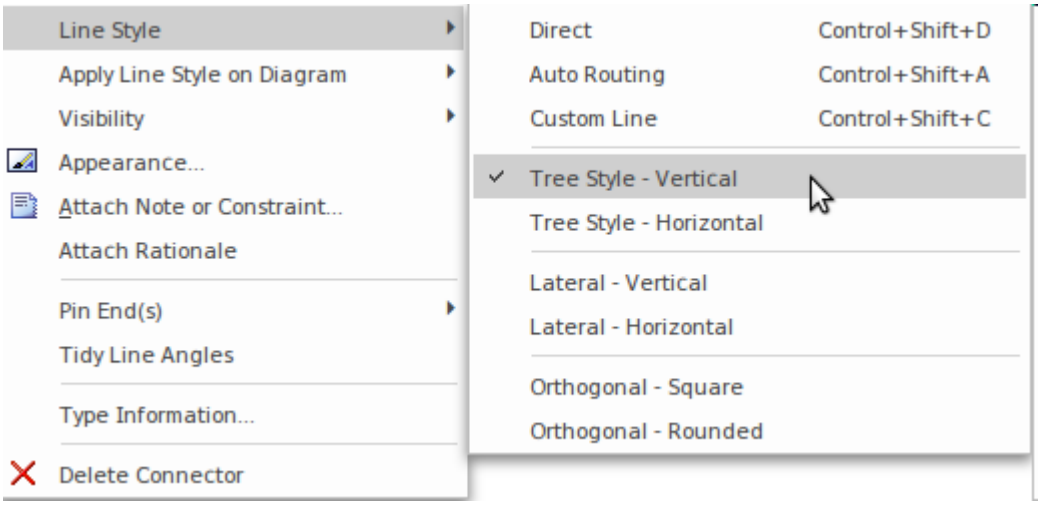
In an earlier section we spoke of the Part Association being the strongest type of Association relationship, but there is another relationship - the Generalization - which is also very strong and essentially is used to model the fact that Blocks (and other Classifiers) belong to the same family. The word 'classifier' comes from our natural languages, such as Chinese and Thai, that have an abstract way of classifying or grouping classes of nouns that have similar characteristics; for example, a belt and a road are long thin things, whereas a berry and a ball are round things. So too with the SysML, the Generalization relationship is used to classify things and the structure can be an arbitrary depth. In many ways it is more natural for engineers to read the relationship in reverse and say something is a specialized version of something.



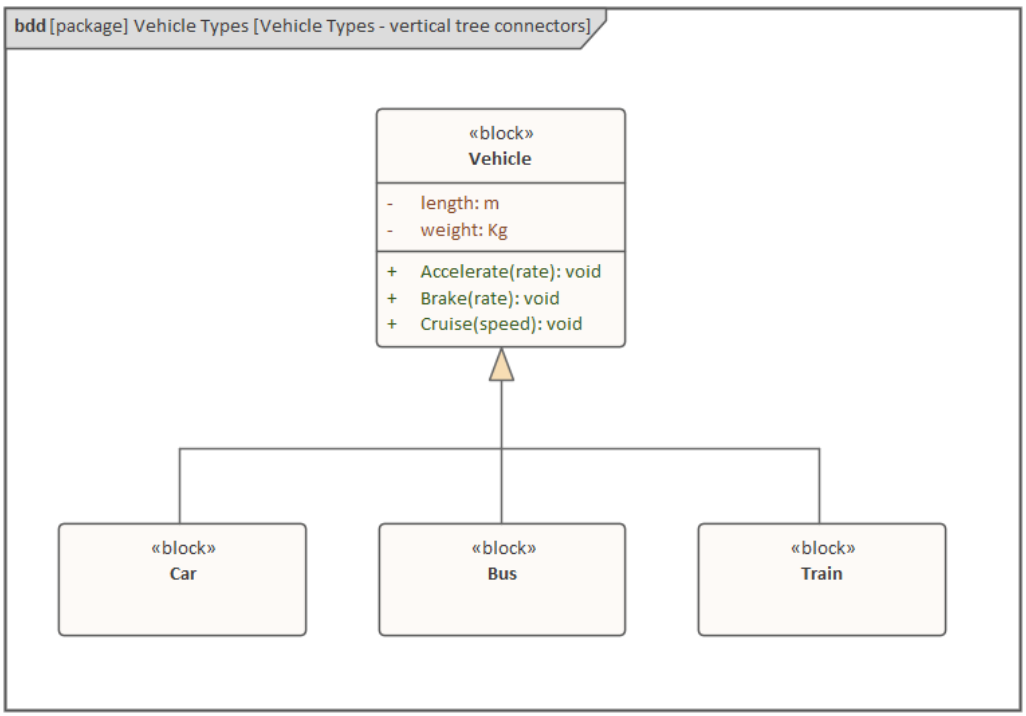
Enterprise Architect allows an engineer to create these classification hierarchies for Blocks, Value Types, Signals, Interfaces, Activities and more. A diagram typically contains a single family.



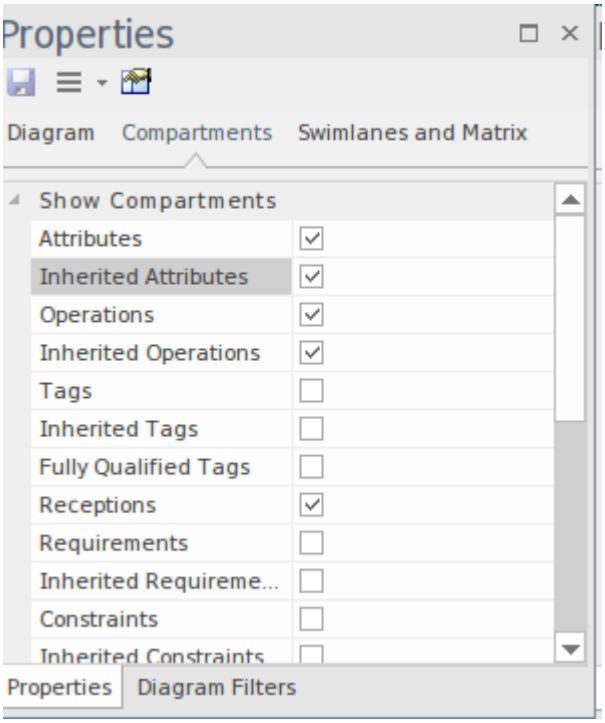
The relationship can be drawn by first selecting the 'Generalization' icon in the Toolbox and then dragging-and-dropping from the more specialized element to the more generalized element. Alternatively this can be done using the Quick Linker.



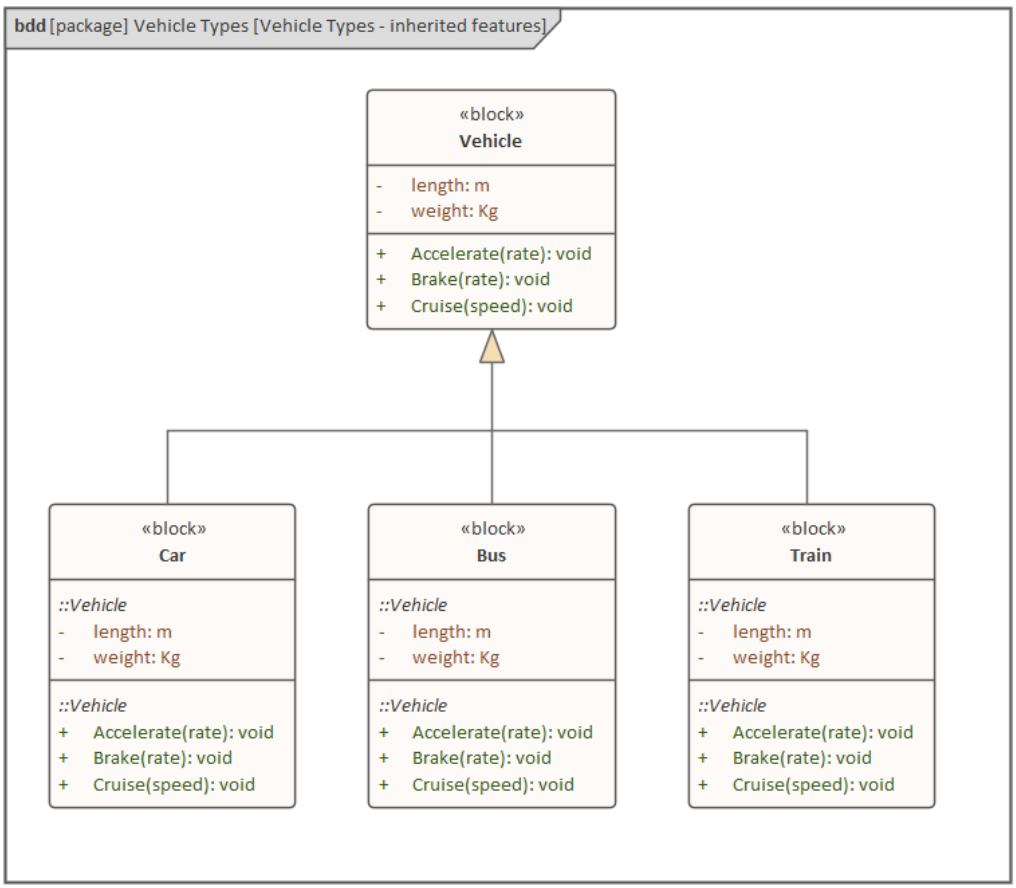
When a Block participates in a generalization hierarchy and has a number of specializations, the connectors emanating from the Block can become untidy. Enterprise Architect provides a mechanism to change the line style to any one of a number of styles, but probably the most useful style is the Vertically-oriented Tree style, which groups the heads of the relationship together and allows their tails to be aligned in parallel.



One of the useful language mechanisms that results from Generalization is for the specialized elements to inherit the structural and behavioral features from the generalized element. So far in the example diagrams the engineer has chosen not to display these inherited features, but they can be set to be displayed using the compartments sections of the element's Property sheet.



The result will be that the specialized Blocks will display the attributes and operations that have been inherited from the parent Block. These will be shown grouped by the name of the parent Block. This mechanism is used extensively in software engineering but also is useful for the systems engineer where the specialized Block automatically inherits the features of its parent by virtue of being a 'member of the family'. Just as in a human family a specialized Block (child) can override the structural or behavioral features inherited from a parent.



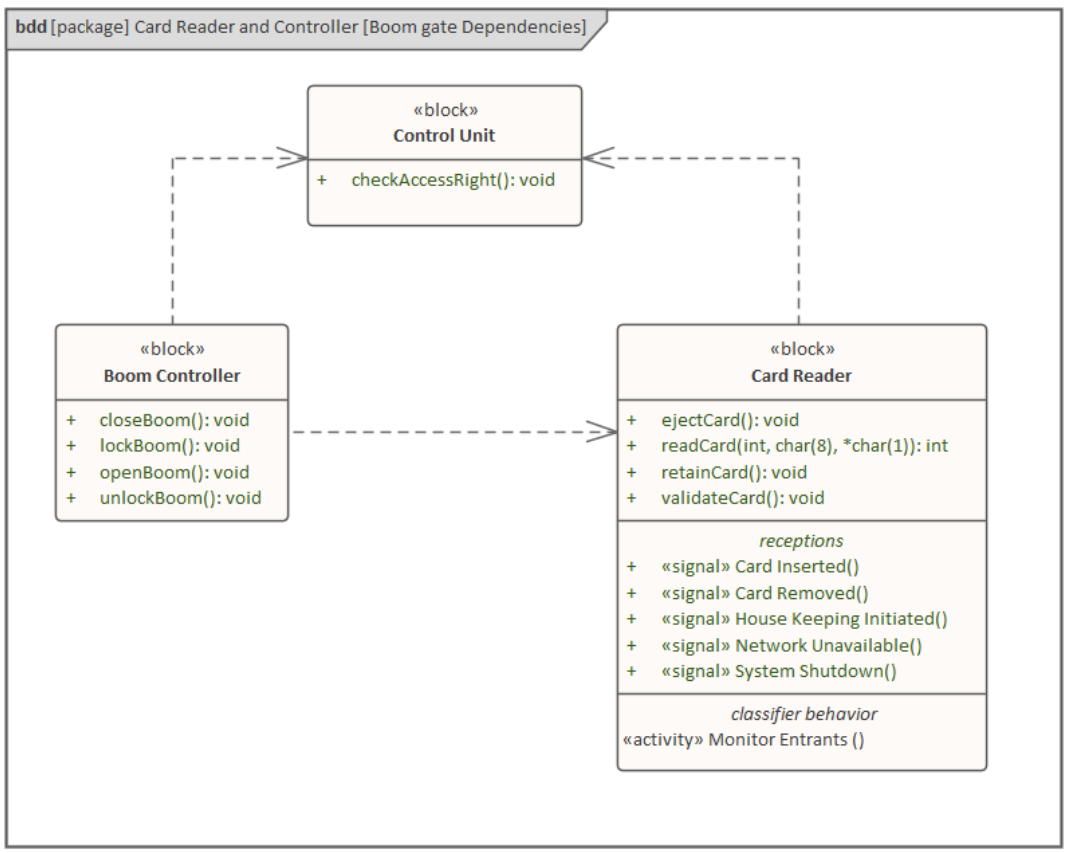
Blocks belong to families base upon certain criteria, and this can be modeled using the Generalization Set, which is a

mechanism used to define the basis for membership of a family.

Dependency

The Dependency is a useful but semantically weak relationship. It is the 'pawn' of the engineers' toolkit of relationships, often used early in the modeling process when the details of the relationships between system elements have not been analyzed or are simply not known. It models the fact that the element (Client) at the tail end of the relationship relies in some way on the element (Supplier) at the arrow-head end of the relationship. Novice modelers can be forgiven for drawing this relationship in the reverse direction, since anecdotally material is often thought to pass in the direction from the supplier to the client. Once the semantics of the relationship are understood and it is realized that the relationship does not say anything about the direction of flow, the mistake will not be made.

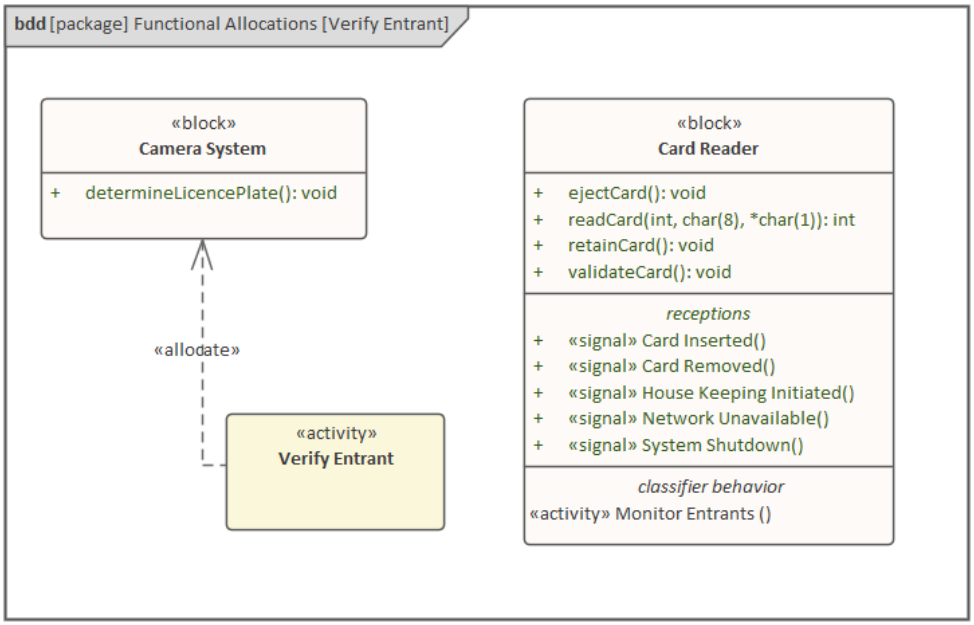
There are a number of types of dependency, all of which are supported by Enterprise Architect. The connector can be created by selecting the 'Dependency' icon in the 'SysML Block Relationships' page of the Toolbox and then clicking on the client (tail end) element and dragging the cursor across to the supplier (arrow-head end) element. The connector can also be created using the Quick Linker arrow at the top right corner of a selected diagram element. Once the relationship has been created, a stereotype can be chosen from the connector's Properties window to make the dependency more specific. This screen capture shows all the available stereotypes, some of which are used between different types of element other than Blocks; for example, Packages and Requirements.



Allocating between Blocks and Activities

The Allocation relationship can be used in a variety of circumstances but it is particular useful for expressing a fundamental relationship between the two most canonical Behavioral and Structural elements, namely the Activity and the Block. This is similar to our natural languages, where a verb is meaningless without the presence of a noun that

carries out the action described by the verb. This type of allocation is referred to as Functional Allocation, and the engineer bridges the divide between these two aspects of a system by finding a Block that can carry out the behavior described by an Activity.



In this diagram the engineer has created two functional allocation relationships that describe how the work specified in the Activity *Verify Entrant* will be carried out. One relationship targets the Camera System that is used to capture the vehicle's licence plate in order to determine if the particular vehicle has been authorized for entry. The other relationship targets the Card Reader Block that is used to determine that the card owner has a relationship with the Parking Station.

Modeling Interaction Points

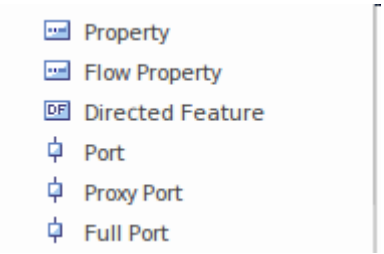
Blocks - and the Parts that are typed by Blocks - reside in an environment and will interact with this environment and the other elements it contains. In general terms the SysML provides a language construct called Interaction Points, which are locations on the boundary of an element that act as entry and exit points for communication with the owning element. Ports are a type of Interaction Point and provide a mechanism for the Block to expose its behaviors - either those that it owns innately or those that are provided by its Parts. The Port is represented by a small rectangle (usually a square) mounted on the boundary of a Block or Part. SysML currently supports two types of Port that are intended to eventually replace the earlier concepts of Flow Port and Standard Port:

- *Proxy Port* - Acts as a relay to expose the behavioral features provided by the owning Block and is typed by an Interface that describes these services
- *Full Port* - Acts as a Part and is typed by a Block, which means that it can itself contain Parts

Enterprise Architect has full support for both these types of Port, and has backward compatibility to the earlier Standard and Flow Ports (which are still available for use but will be deprecated in later versions of the standard).

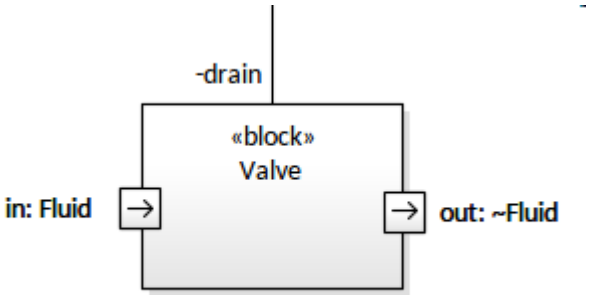
Once a Block Definition diagram has been created and a Block has been placed on the diagram, Ports can be created by either:

- Drag-and-dropping the appropriate Port from the Toolbox onto the Block - this diagram shows the section of the Block Definition Toolbox that lists the Ports



- Selecting the 'New Child Element' option from the Block's context menu and select the appropriate type of Port, as shown:

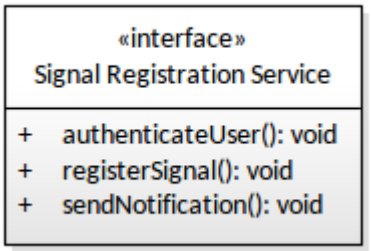
Whichever method you choose, the Port will then be automatically placed on the boundary of the Block and can be moved into the required position and named. Note also that the Port can be added from the 'Element' panel of the 'Design' ribbon. This screen image is of a section of diagram showing two Ports with direction indicators inside the Port element. The Ports have been named 'in' and 'out' respectively, and have been typed by 'Fluid', which indicates the type of the item arriving at the Port.



Ports can contain Interfaces and also nested Ports; Enterprise Architect provides a useful feature that allows the engineer to customize the Port size, effectively changing the small squares into small rectangles.

Interfaces and Ports

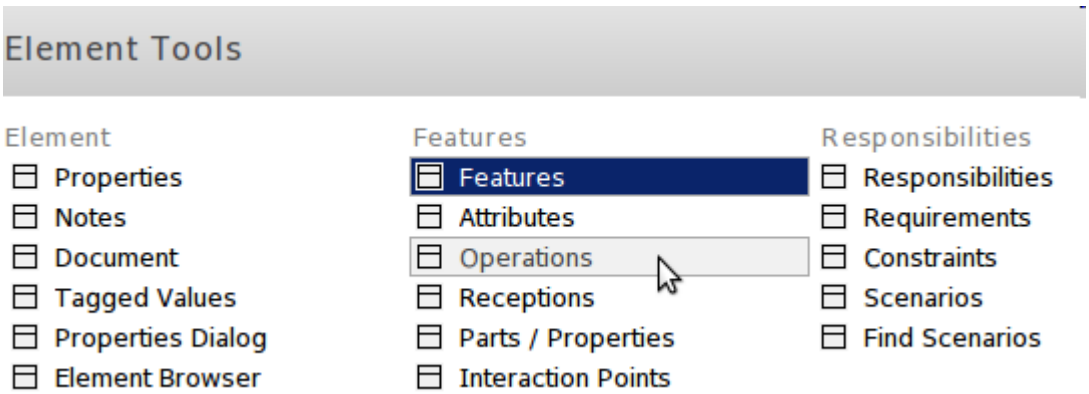
An Interface is a useful way of encapsulating a group of services provided by a Block, providing a simple way of exposing those services to clients. The Interface has the same appearance as a Block and can have defined operations and Receptions, but no attributes (Properties).



An Interface can be created by simply dragging the 'Interface' icon from the Toolbox page. Behavioral features can be added by selecting the appropriate Feature from the Interface's context menu.

Another easy way to create an interface's operations is to select the Block in a diagram or in the Browser window and click on the ribbon item:

Design > Element > Editor > Features > Operations



Operations are simply created by selecting the 'Operations' tab and adding the name and other details in a row of the window. Any number of operations can be created, and each operation can define any number of parameters that specify

the inputs and outputs to the operation. Receptions - the other behavioral feature - can be added in a similar way using the 'Receptions' tab. Any of these items can be reordered using the <Ctrl>+ up- and down-arrow keyboard keys.

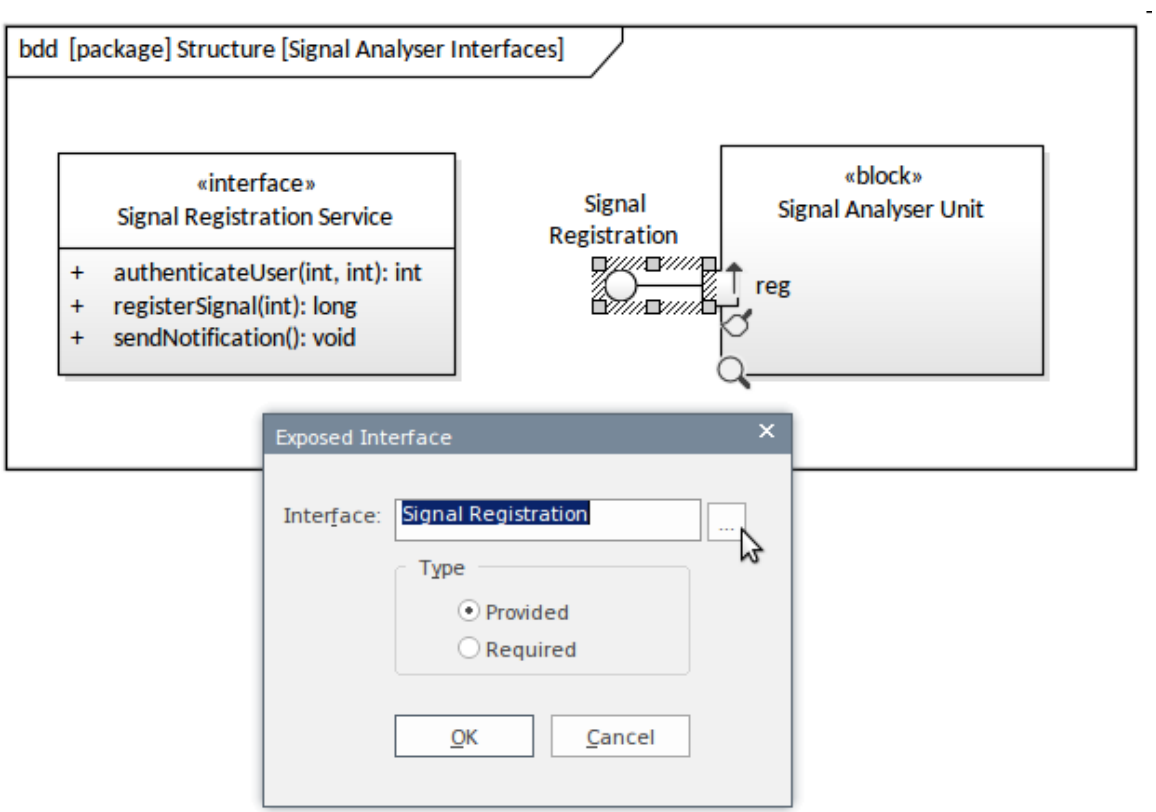
Interfaces can be added to Ports, which are a common mechanism in the Unified Modeling Language and provide a way of publishing the services that are available at a Port. The interfaces are of two fundamental types:

- *Provided* - available for use
- *Required* - required for use

With a Port selected on the boundary of a Block in a diagram, an Interface can be added as for any other structural element - from the Port's context menu items:

- New Child Element | Provided
- New Child Element | Required

Interfaces added in this way must be typed by an Interface proper (an element stereotyped as <<interface>>). This can be achieved by first selecting the Interface in the diagram and then selecting the Properties window from the Interface's context menu. You can then name the interface and use the [...] icon to navigate or search for the Interface element. This diagram demonstrates the step for a Signal Registration Interface.

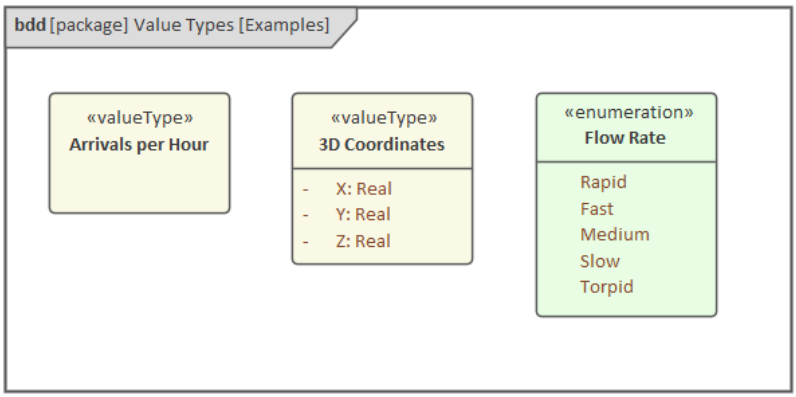


Modeling Quantity using Value Types

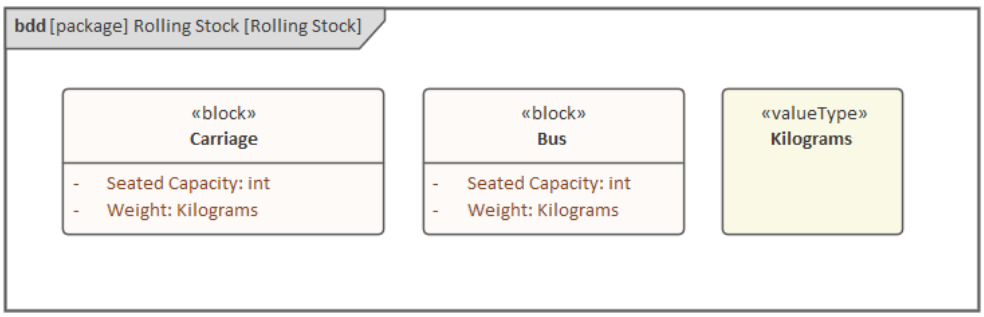
A Block can have simple properties called Value Properties, which define attributes of a Block that represent a scalar or vector quantity. Values are used to express information about a Block and provide a slot for an actual value to be entered in an instance of a Block. For example, a Tank can have a properties of diameter, height and volume defined. The value types allow an engineer to create a universal way of quantifying a property; for example:

- A centrifuge has a maximum speed specified in revolutions per minute (rpm)
- A train carriage has a weight specified in kilograms (kg)
- A tank has a volume specified in liters (l)
- A light source has a luminous intensity specified in Candela (cd)
- A dialysis machine has a blood flow rate specified in milliliters per minute (ml/min)

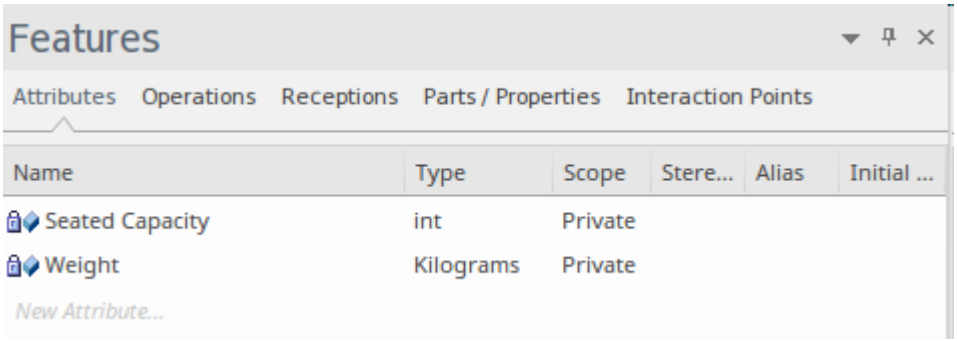
This diagram shows a number of different Value Types that can be defined in Enterprise Architect and then applied to any number of attributes defined in Blocks.



The intent of the Value Type is to allow an engineer, team or industry to define standard types that can be reused to characterize the value properties defined for any number of Blocks. For example, the value type of 'Kilogram' could be applied to a value property specifying the weight of a train or the weight of a bus or the seating capacity of either.



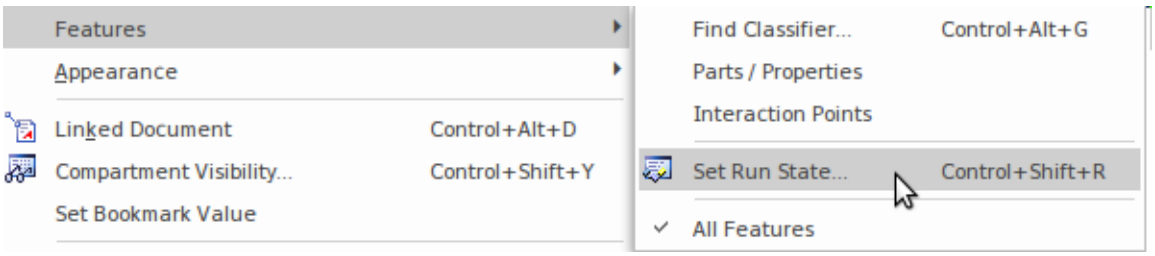
These value types, as the name suggests, have a value that describes the quantity of the property; they can be defined using the 'Attribute' tab of the Features window, as shown here.



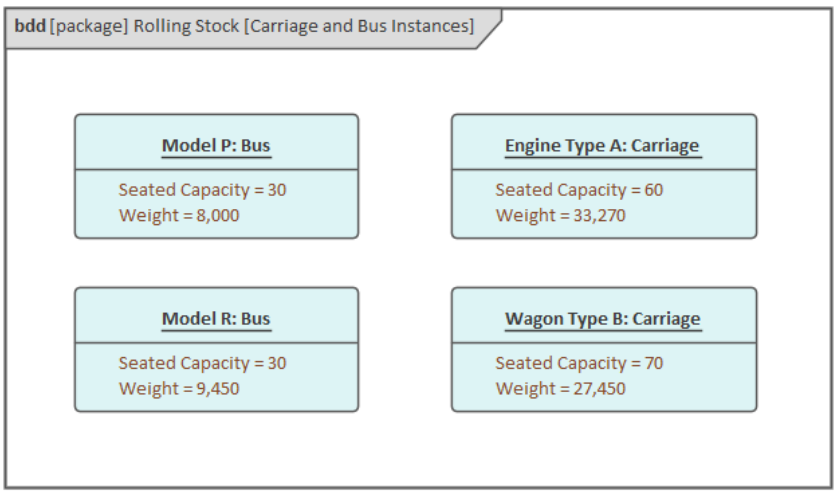
For example, two Blocks representing a Train and a Bus could have a property of 'Weight' defined that is typed by the value type 'Kilogram'. If an attribute such as 'Seating Capacity' has a simple type such as 'Integer' this can be directly selected in the 'Type' drop down, but if 'Type' is based on a Value Type this can be selected using the 'Select type..' option from the drop-down.

Instances of Blocks that have an attribute (Value Property) defined in Enterprise Architect can have an actual value specified for the attribute. For example, each instance of the Bus and Train with, say, a particular model number could have a different weight defined. Other properties such as 'Seating Capacity' could have a primitive type of 'Integer' defined, and these also could be set for particular instances of 'Carriage'.

Enterprise Architect allows an engineer to set the values for each of the defined attributes by using the 'Set Run State' option from the 'Features' sub-menu for a Block instance.



The attribute values (slots) can be displayed on a diagram, allowing an engineer to create compelling examples or catalogues of Block instances, as shown here.



Using Properties and Parts to Model Block Usage

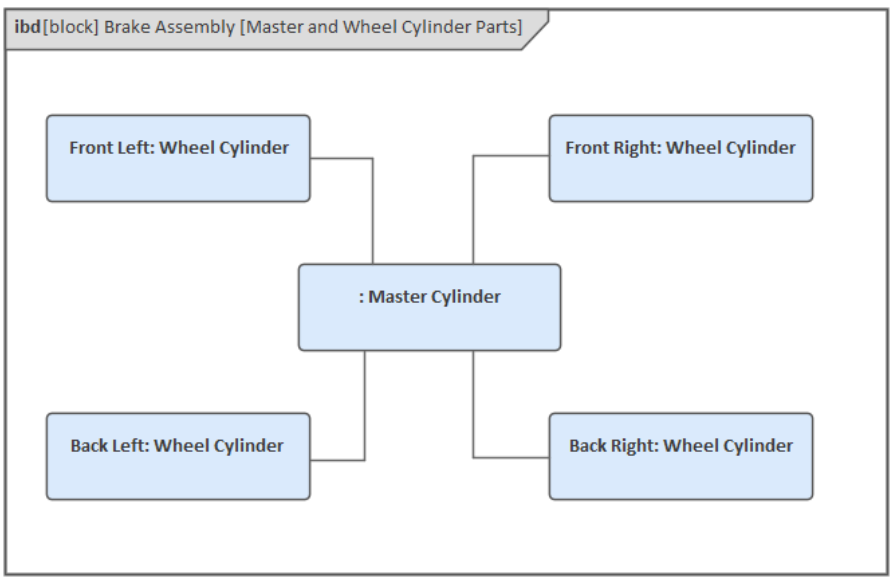
Blocks are classifiers and describe the characteristics of a set of elements that represent the way the Block is used in a context. When the Block has attributes (value properties) defined these are given specific values in the instances of the Block. Effectively, each Block instance has an identity and typically would have different values assigned that define the Block's state. Enterprise Architect allows these values to be specified using a *Set Run State* option available from the context menu.

Internal Block diagrams often show how a Block's parts are connected together in a usage context. Enterprise Architect allows Blocks to be dragged from the Browser window onto a diagram and dropped as Part Properties. These are effectively Parts and represent instances of the Block classifier. An engineer has the opportunity to name these in the context of the diagram. For example, this diagram represents a Brake Assembly that has been modeled, which is indicated in the diagram frame in this format:

ibd [block] Brake Assembly [Master and Wheel Cylinder Parts]

- *ibd* - signifies that it is an Internal Block diagram
- *block* - signifies the owing element type
- *Brake Assembly* - is the name of the element
- *Master and Wheel Cylinder Part*s - is the name of the diagram

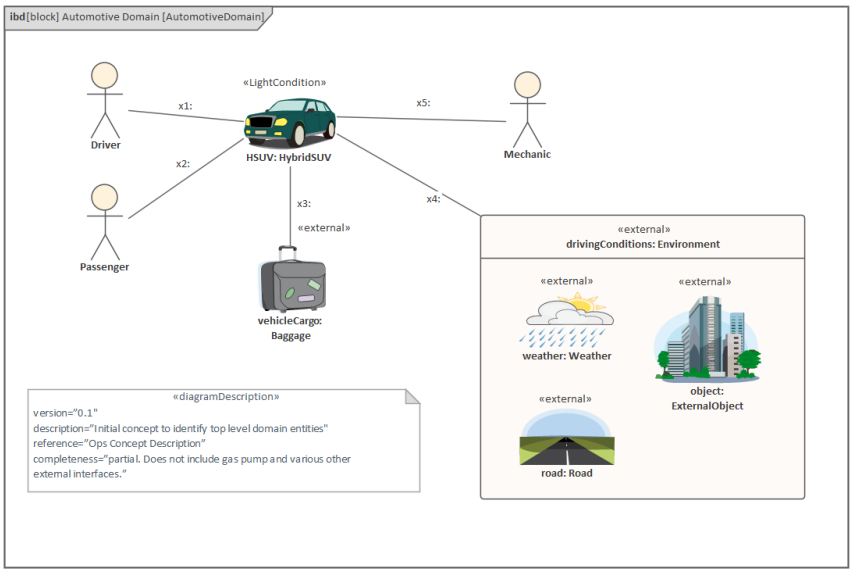
The engineer has named each of the wheel cylinder parts (Front Left, Front Right, Back Left, Back Right) as these need to be identified with respect to their location in the vehicle, but has decided not to name the master cylinder as no further qualification is required.



Introducing Internal Block Diagrams

An Internal Block diagram provides a way of visualizing the internal structure of a Block, including its Properties and Parts and the way that these Parts relate to each other. The diagram is not required to display all the Parts that a given Block is composed of and it is common for an engineer to create a diagram that focuses on a particular aspect of a system or subsystem.

The frame of an Internal Block diagram represents the owning Block, so it will be named as such and the elements that appear on the diagram will be Parts that are instances of the Blocks that the owning Block is composed of. This Internal Block diagram shows an instance of a vehicle in a given context; it uses a number of images in place of the conventional SysML language symbols as a way of adding appeal and making the diagram more compelling to a non-engineering audience. For more information see the Internal Block Diagrams Help topic.



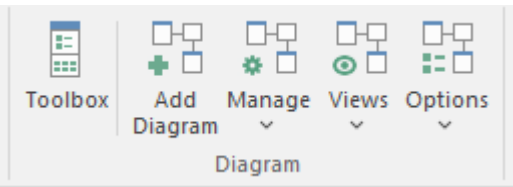
Creating an Internal Block Diagram

An Internal Block diagram can be created from a number of places in the User Interface, such as:

- Design ribbon - *Add Diagram* Icon on the *Diagram* Panel
- Browser window toolbar - *New Diagram* icon
- Browser window context menu - *New Diagram*

We will use the Design ribbon to create an Internal Block diagram. Firstly, select the location in the Browser window where you want the diagram to be located. In contrast to most other SysML diagrams the Internal Block diagram is typically inserted under its owning Block. Once the location has been selected in the Browser window, select the ribbon item:

Design > Diagram > Add Diagram

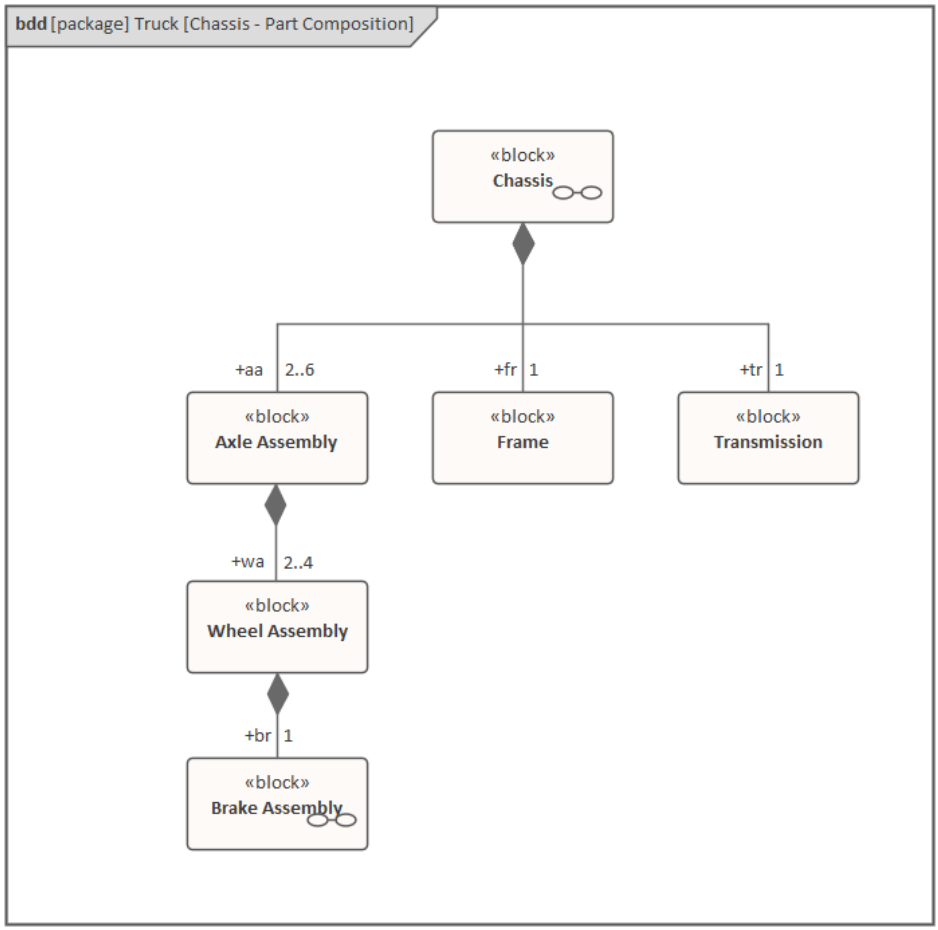


Selecting this option will open the *Model Builder* dialog and display the *Diagram Builder* tab page, allowing you to name the diagram; the diagram name defaults to the name of the Block that contains the diagram. With the SysML perspective chosen and the version of SysML selected, a list of diagram types is displayed from which you select the Internal Block diagram. Click on the *Create Diagram* button to create a new Internal Block diagram in the location

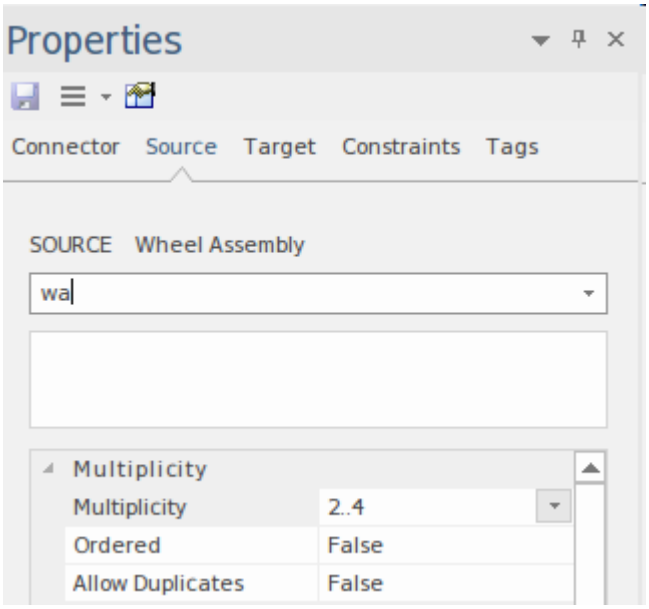
selected in the Browser window. The diagram canvas will be opened, allowing you to start adding elements and connectors that describe the internal structure of the Block. Enterprise Architect will also display the 'Internal Block' page of the Toolbox, which contains the elements and relationships defined by the SysML specification to be applicable for constructing this diagram type. Any number of other Toolbox pages can be opened if required, in addition to the 'Common Elements' and 'Common Relationships' pages that are always available.

Modeling and Connecting Parts

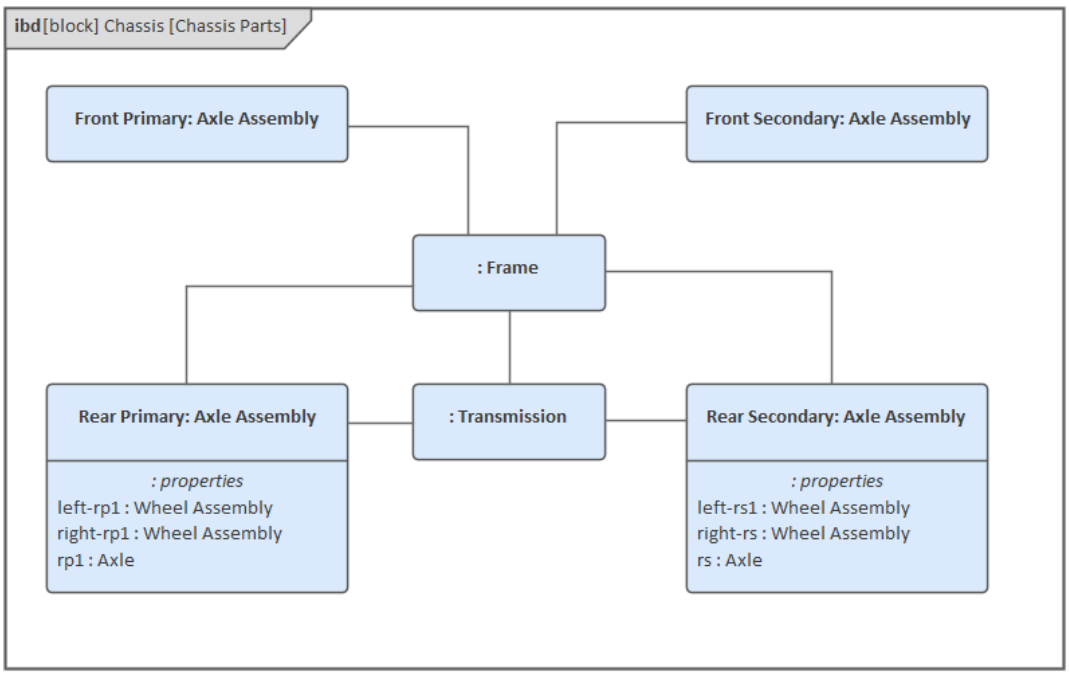
When a Block is composed of a number of other Blocks, it is typically the case that when the system is instantiated the contained Blocks will perform much of the work that is required of the owning Block. The Internal Block diagram provides a language mechanism to visualize how the parts interact, to show the structure of the Blocks in context and to provide the overall behavior specified by the owning Block. In an earlier topic we viewed a Block Definition diagram of the chassis of a truck, describing the Blocks that make up the chassis based on Blocks.



The diagram includes multiplicities at the part ends of the Association showing how many of a particular part can be included in a single instance of the owning Block. These numbers represent the cardinality expressed as an upper and lower bound, which can be defined in the Properties window for the connector. This screen capture shows a portion of the Properties window used to define multiplicities and other properties of the Association End, all of which add rich semantics to the association.

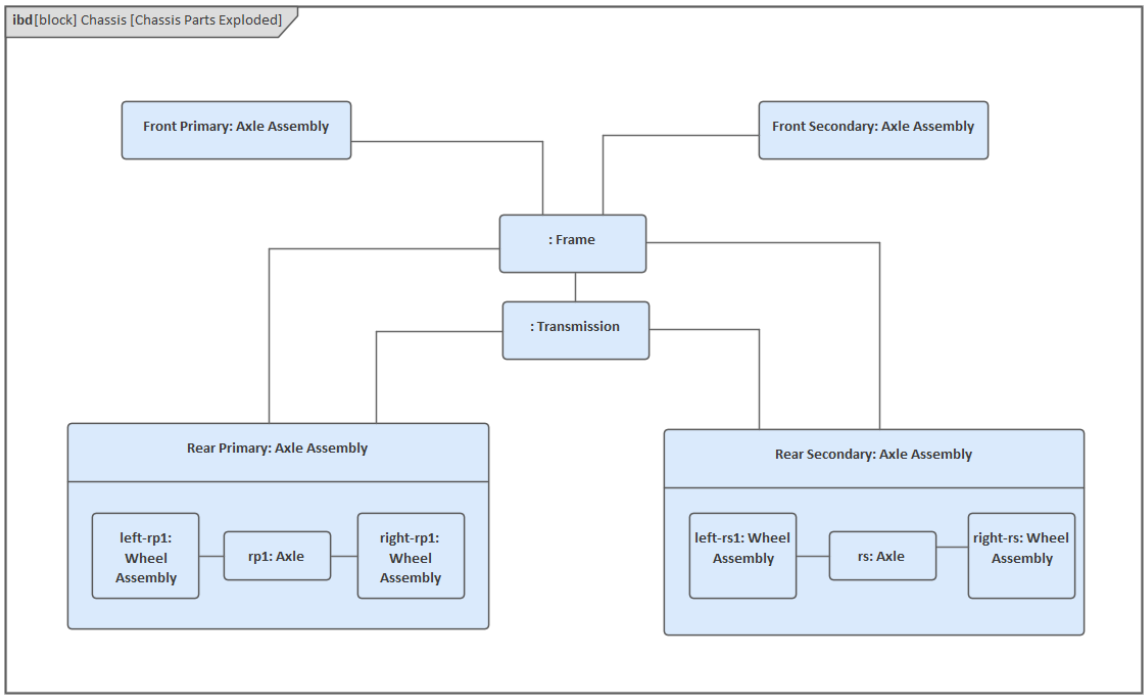


This Internal Block diagram of the chassis shows the parts that make up the chassis in an actual context. The number of axles is defined for the particular instance of the truck.



Enterprise Architect allows an engineer to create diagrams with parts nested to any level, which helps demonstrate the structure of a Block and the way the parts would be connected in a real world context.

This diagram shows parts nested on two levels, but any number of levels are possible and can be created on a diagram. This type of expression can lead to quite large diagrams, and Enterprise Architect supports paper size up to A0, allowing large diagrams to be created and printed.



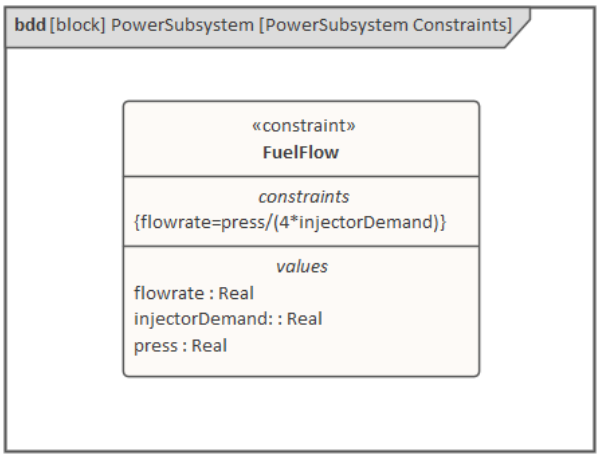
In the diagram the Rear Primary and Secondary Axle Assemblies have been shown in detail, where each Axle is composed of a right and left wheel assembly, which themselves could be shown as a nested structure comprising the Brake Assembly, which in turn could show the Wheel Cylinder Assemblies.

In the case where a part is added to the diagram but the modeler for some reason needs to change the Block classifier that types the Block, this can be done from the 'Parts' context menu that provides an option for the Property Type to be changed. This can be convenient where an elaborate diagram has been created and it is easier to leave the visual element in place and just update the Block it is based on.

Modeling Parametric Equations

Engineers are charged with finding solutions to problems and opportunities, and use models as a way of visualizing simplifications of both the system under consideration and the world context that the system will need to operate in. Systems engineering models created in Enterprise Architect provide a valuable tool for analysis, design, architecture, testing and visualization. This includes being able to predict how a system will behave in a given context, balancing competing requirements and design considerations in the form of stakeholder negotiations and trade-off analysis. Parametric diagrams are an advanced tool that can assist the engineer to address these concerns in a model and pre-emptively represent how a system is likely to behave.

In an earlier topic we learnt how equations can be modeled using the Block Definition diagram, with the Part Association relationship articulating the variables (parameters) of the equation. This provides an essential mechanism for re-use. As a refresher of how we use a Constraint Block to model equations refer to this diagram, which uses a Constraint (which is a stereotyped Block) to model a vehicle's fuel. The Fuel Flow Rate is based on an equation that has fuel pressure (press) and fuel demand as variables (parameters).



Any equation, or system of equations can be modeled using the constraint.

This constraint can potentially be re-used in a number of different contexts. It is on the Parametric diagram that we see how it is used. For more information, see the Parametric Diagrams Help topic.

Introducing Parametric Diagrams

A Parametric diagram provides a way of visualizing equations and their parameters in a particular context in the form of constraint properties. Each of these properties represents a usage of a ConstraintBlock that has typically been defined on a Block Definition diagram.

The frame of a Parametric diagram represents the owning ConstraintBlock, so it will be named as such and the elements that appear on the diagram will be constraint properties, which are instances of the ConstraintBlocks that the owning Block is composed of, thus showing the composition of the constraint.

Creating Parametric Diagrams

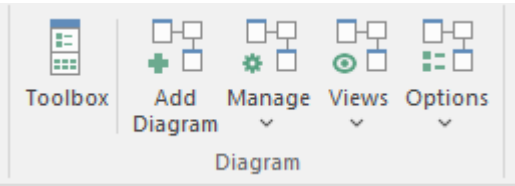
A Parametric diagram can be created from a number of places in the User Interface, using any of these options:

- Design ribbon - *Add Diagram* icon on the *Diagram* panel
- Browser window toolbar - *New Diagram* icon
- Browser window context menu - *New Diagram*

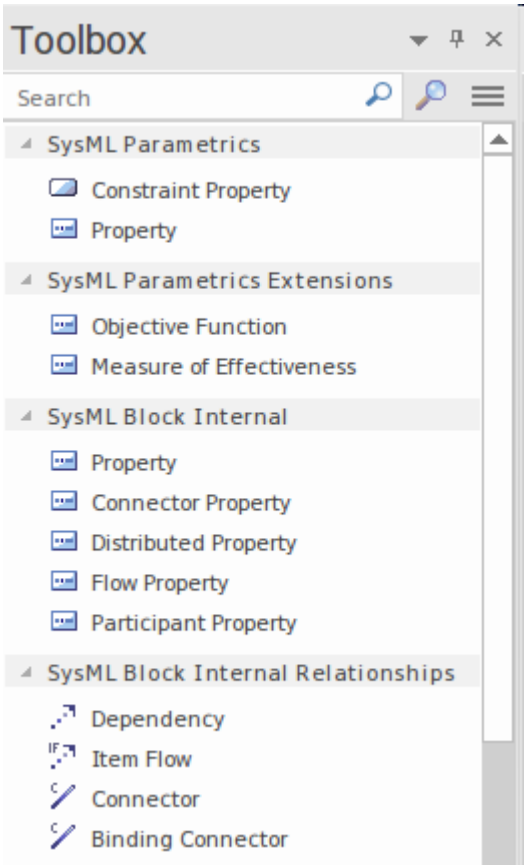
We will use the 'Design' ribbon option to create a Parametric diagram. Firstly you select the Package in the Browser window where you want the diagram to be located.

In contrast to most other SysML diagrams, the Parametric diagram is typically inserted under its owning Constraint Block. Once the location has been selected in the Browser window, select:

Design > Diagram > Add Diagram



Selecting this option opens the 'New Diagram' dialog, allowing you to name the diagram; the name will initially default to the name of the Constraint Block that owns the diagram. With the SysML Perspective chosen and the version of SysML selected, a list of diagrams will be displayed allowing you to select the Parametric diagram. Click on the OK button to create a new Parametric diagram in the location selected in the Browser window. The Diagram View will be opened, allowing you to start adding elements and connectors that describe the equations and the parameters. Enterprise Architect will also display the 'Parametric' pages of the Diagram Toolbox, which contain the elements and relationships defined by the SysML specification to be applicable for constructing this diagram type. Any number of other Toolbox pages can be opened if required, in addition to the 'Common' elements and 'Common Relationships' pages that are displayed by default and that allow diagram notes, legends and other common elements to be added.



The most important elements and connectors used with the Parametric diagram are:

Elements

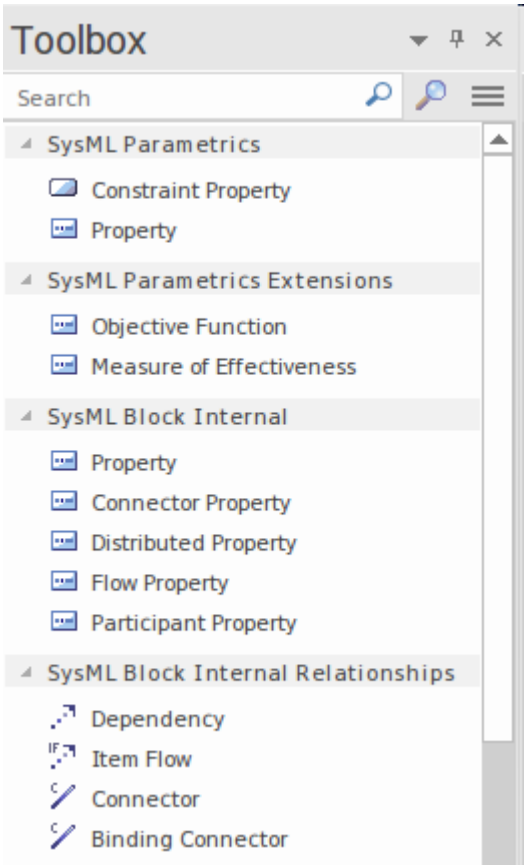
- ConstraintProperty
- Property

Element Extensions

- Objective Function
- Measure of Effectiveness

Connectors

- Dependency
- Item Flow
- Connector
- Binding Connector

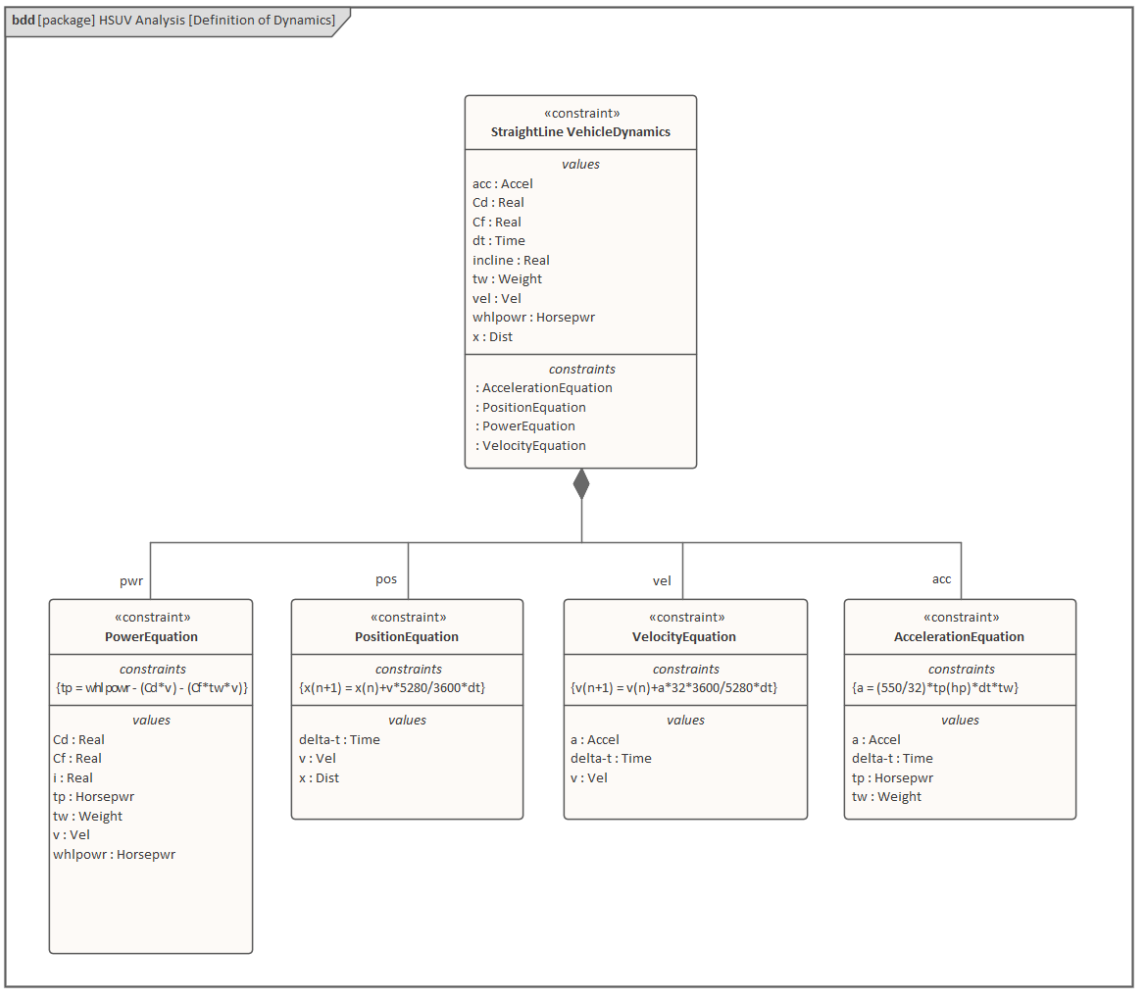


Elements can be added to the diagram by dragging-and-dropping them from the Toolbox page onto the diagram canvas.

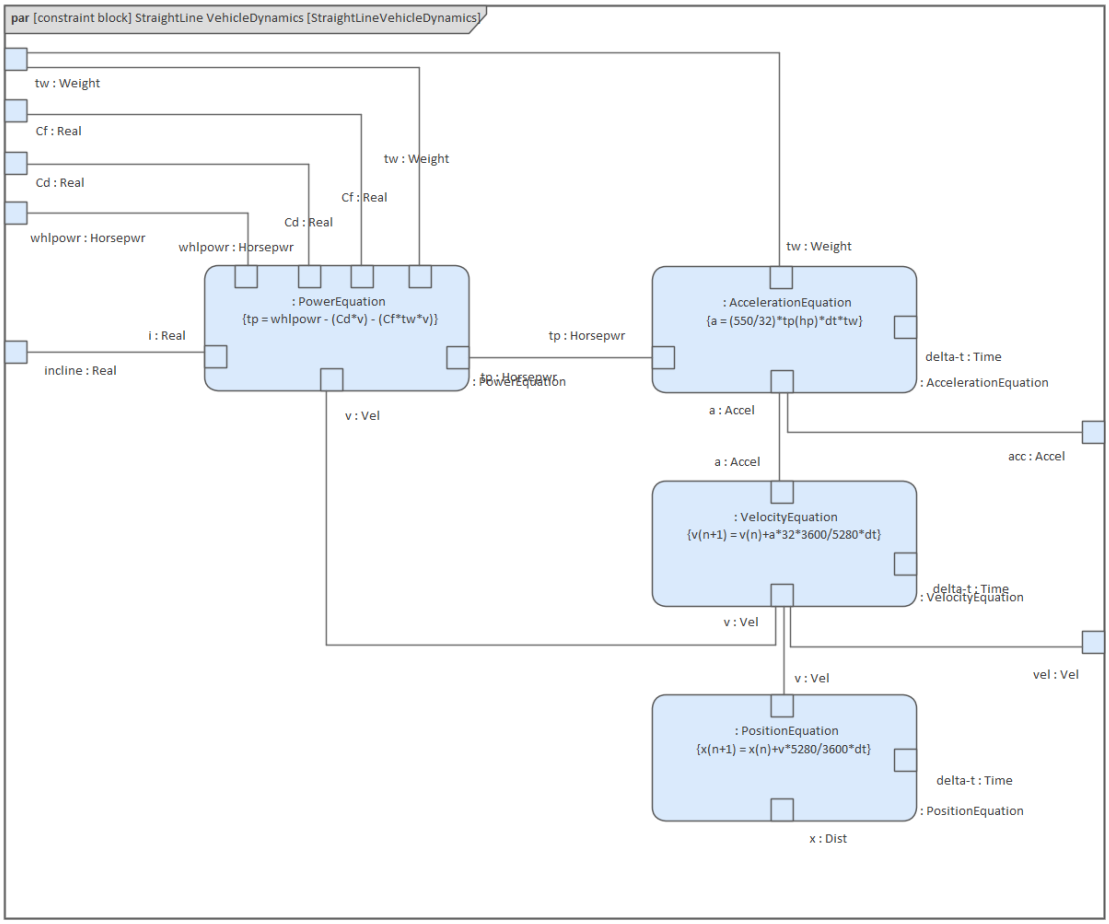
Systems of Equations using Part Associations

Engineering problems and systems typically require detailed analysis to determine how a particular proposed solution will perform. The analysis can involve any number of equations that are often related to each other to determine a particular value. Enterprise Architect allows an engineer to construct systems of equations using a Block Definition diagram and then to use these equations in multiple Parametric diagrams to describe proposed solutions.

This Block Definition diagram describes parameters of the straight line dynamics of a vehicle - the HSUV - and includes a number of equations that are represented on the diagram as Constraints, which are a type of Block.

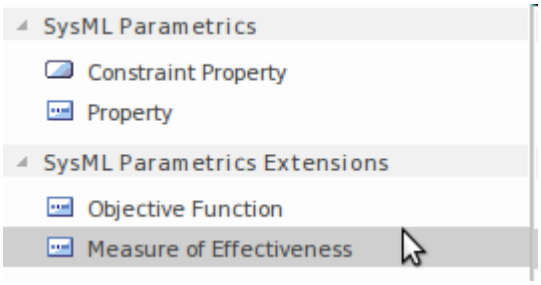


The next diagram, a Parametric diagram, shows how the ConstraintBlocks are used in a particular context, being represented on the diagram as ConstraintProperties. We can visualize how the total power parameter is calculated, with a connection between the Power Equation and the equivalent parameter on the Acceleration Equation. Connections can be seen between the Position Equation and the Velocity Equation, which is ultimately connected back to the Acceleration Equation.


par [constraint block] StraightLine VehicleDynamics [StraightLineVehicleDynamics]
tw : Weight
Cf : Real
Cd : Real
whlpowr : Horsepwr
tw : Weight
Cf : Real
Cd : Real
whlpowr : Horsepwr
: PowerEquation
{tp = whlpowr - (Cd*v) - (Cf*tw*v)}
i : Real
incline : Real
tp : Horsepwr
: PowerEquation
v : Vel
tw : Weight
: AccelerationEquation
{a = (550/32)*tp(hp)*dt*tw}
delta-t : Time
: AccelerationEquation
a : Accel
acc : Accel
a : Accel
: VelocityEquation
{v(n+1) = v(n)+a*32*3600/5280*dt}
delta-t : Time
: VelocityEquation
v : Vel
vel : Vel
v : Vel
: PositionEquation
{x(n+1) = x(n)+v*5280/3600*dt}
delta-t : Time
: PositionEquation
x : Dist

Measures of Effectiveness using Parametrics

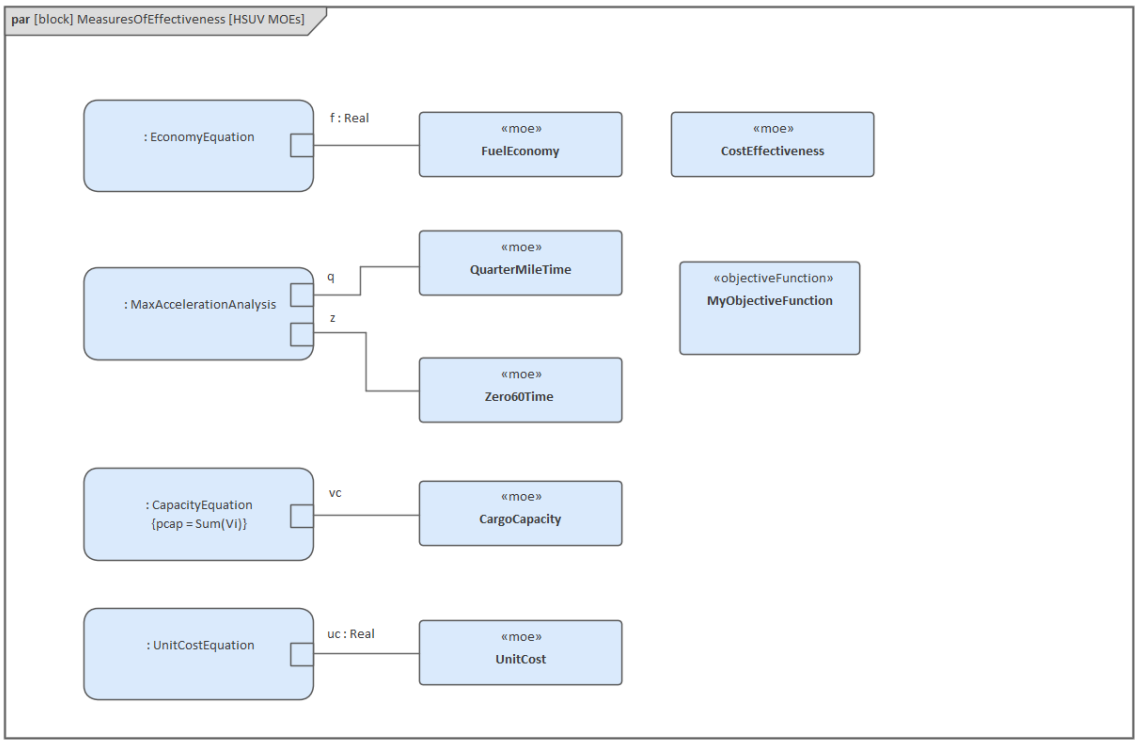
Measures of Effectiveness (MOEs) are an important engineering mechanism used to quantify the achievement of mission objectives or specified desired outcomes. They can be modeled using the Parametric diagram, and the Measure of Effectiveness element is available from the 'Parametrics' page of the Diagram Toolbox, from which it can be dragged onto a diagram and related to parameters of equations represented as ConstraintProperties.



The Measures of Effectiveness can be reused to evaluate any number of design alternatives and allow these designs to be systematically compared and evaluated. This diagram shows the overall cost effectiveness of a Hybrid SUV (HSUV) for a proposed solution entitled 'Alternative One' (alt1) based on a number of MOEs including:

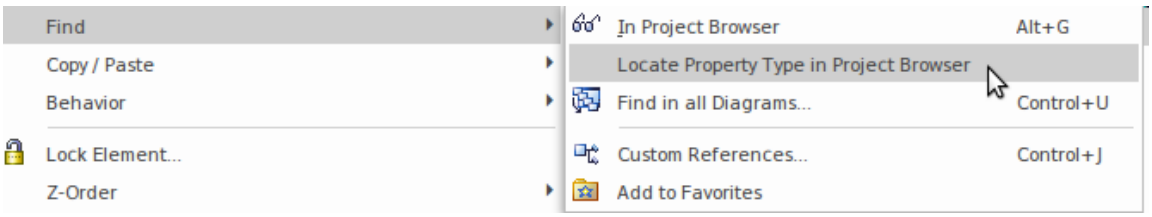
- FuelEconomy - *Expression of fuel economy*
- QuarterMile - *Time taken to travel 0.25 miles*
- Zero60Time - *Time taken to accelerate to 60/mph*
- CargoCapacity - *The volume of the cargo spaces*
- UnitCost - *Cost of the vehicle*

Each one of these would have its own Parametric model that would be able to determine the effective value and contribute to the overall equation, which is a series of weighted sums {CE = Sum (W i*P i)}. Enterprise Architect allows any number of alternatives to be defined, and the engineer can reuse the MOE elements and the ConstraintBlocks used to define the contributing equations.



Enterprise Architect has a useful search feature that allows the type that a property is based on to be located in the Browser window. This function is particularly useful for finding the owning Block or constraint of a Block or

ConstraintProperty found on Parametric and Internal Block diagrams.



It is possible to setup a slide show using the 'Model Views' tab of the Focus window. For more information see the Diagram Slide Show Help topic.

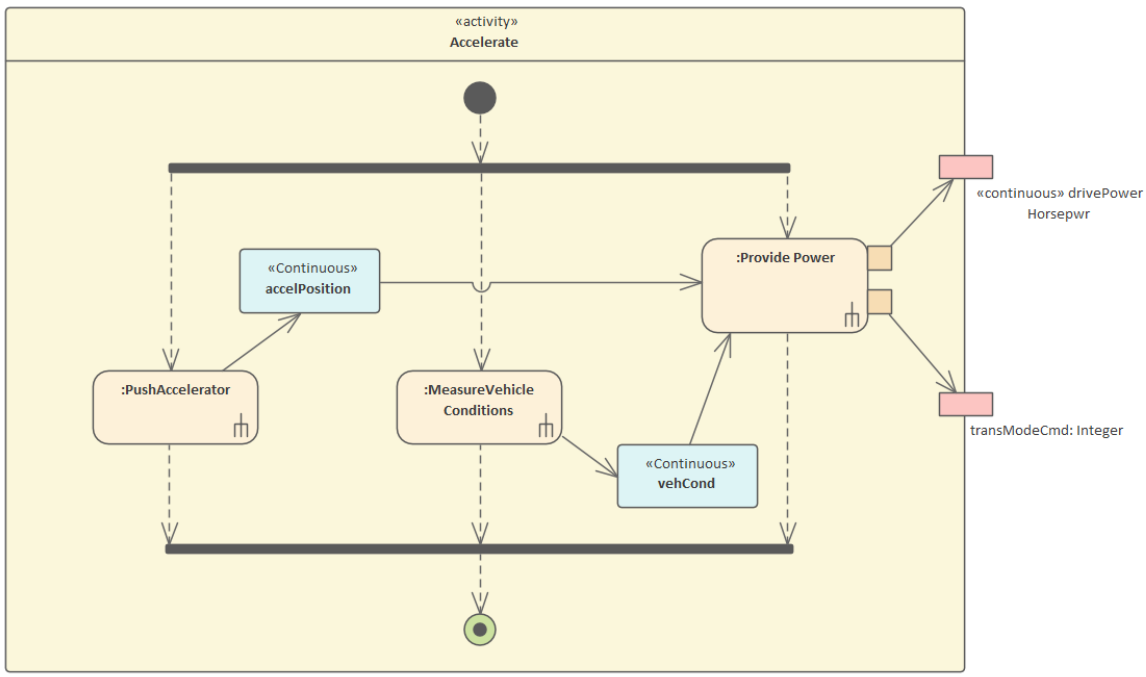
Coordinating Behavior with Activities

As discussed in an earlier topic, the Systems Modeling Language (SysML) has two fundamental aspects that are analogous to two important grammatical categories in the natural languages humans use to communicate, namely *nouns* and *verbs*. In the SysML these are *Structural* and *Behavioral* Constructs; Structural Constructs being analogous to nouns in our natural languages, and Behavioral Constructs being analogous to verbs.

We referred to the structural aspects of the Language in previous topics, when we discussed both Packages and Blocks. We will now turn our attention to the main Behavioral diagram, namely the Activity diagram. There are a number of other behavior diagrams, and indeed behavior is visible in structural diagrams in the form of operations and also in the Behavior that is assigned directly to a Block.

While the newcomer to SysML, viewing the Activity diagrams for the first time, might be reminded of the flow chart, they will soon learn that the Activity diagram has syntax and semantics that go far beyond the flow chart. The Activity diagram is formally based on a branch of mathematics called *Petri Nets* and it uses a system of tokens to indicate both the sequence of actions and also the items that flow through the system. The items that flow can be information items, physical items or even control signals. We will reference this token system as a way to illuminate the working of the Activity diagram.

This diagram, describing a vehicle's acceleration, shows many of the elements that are commonly seen on an Activity diagram. You will see in the subsequent topics that it is a very expressive diagram that, if crafted carefully, can rigorously convey a lot of information.



In fact the syntax of the Activity diagram is one of the richest of any of the SysML diagrams, and when you add to this the rich and effective mechanisms and tools that Enterprise Architect includes to work with these diagrams, the opportunities for a modeler to express themselves makes these one of the most versatile but also challenging parts of system representation.

The SysML Activity diagram is based on the UML diagram of the same name, but additional semantics have been added in two areas:

- *Continuous Flow*, allowing restrictions on the rate at which entities flow along edges in an Activity, and mechanisms to ensure that the most recent information is available to Actions
- *Probability*, introduced into Activities to include the likelihood that a value will be available to an edge or output on a parameter set

While the diagram might be said to be based on verb serialization mechanisms (strings of verbs connected together with nouns) in our natural language, as mentioned earlier it has its formal origins in a branch of mathematics called *Petri Nets*

and token flow. It is imperative that a modeler understands the token flow aspect of the language, and can learn to visualize these invisible items that flow through Object Flows, are detained at buffers, and are controlled by other language mechanisms that direct how items flow from Actions. Without this understanding it is difficult to interpret an Activity diagram, including how the sequence of Actions is controlled, how the inputs are consumed and how the outputs are created.

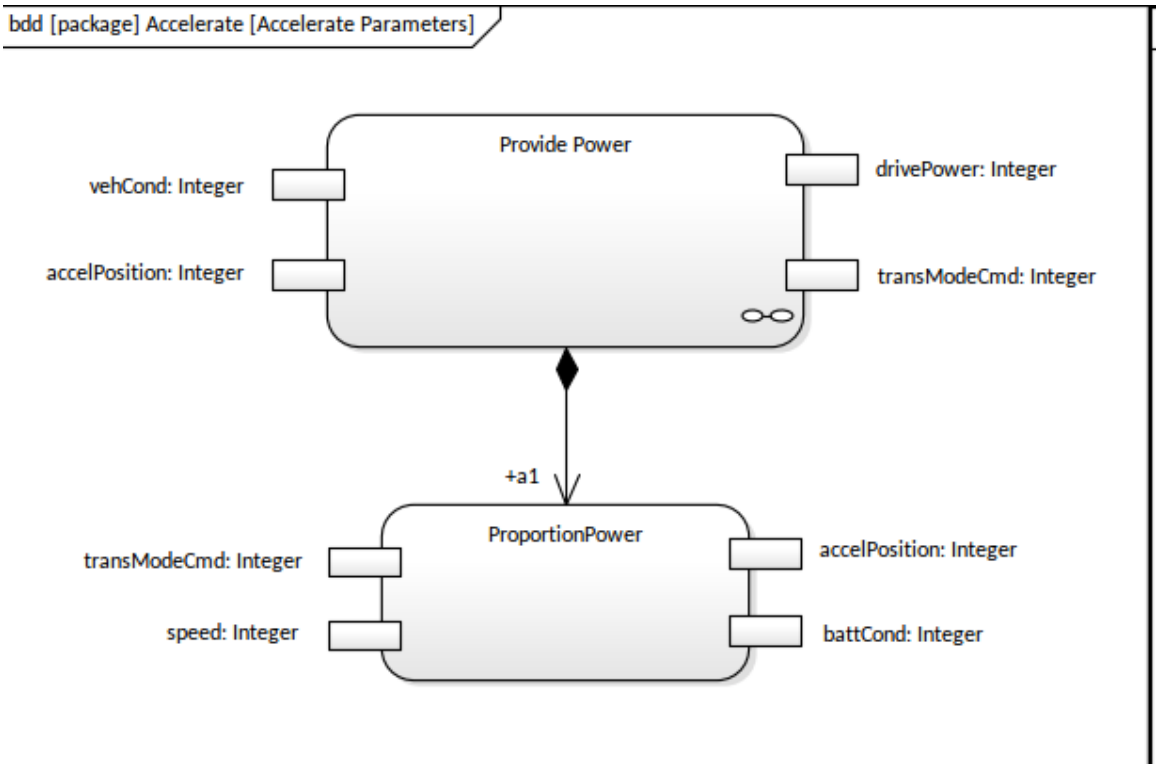
The significant difference between Activity diagrams and any of their close cousins, such as Flow Charts or Process diagrams, is the ability to create relationships between these behavioral elements and structural elements.

A fundamental aspect of the discipline of Systems Engineering is the ability to segregate function from form, but also to be able to create a mapping between them that exposes the seams that relate these two integral parts of architecture and design. Empirical evidence on large scale, complex systems engineering problems has proven that profound benefit results from this approach.

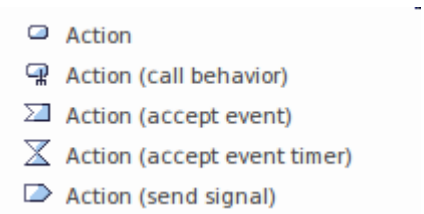
Enterprise Architect provides a rich toolbox to work with these relationships, including the ability not only to allocate system behavior in the form of Activities and Actions to Blocks, but also to relate these elements to behavioral features owned by Blocks, such as operations.

Actions the Fundamental Behavioral Building Blocks

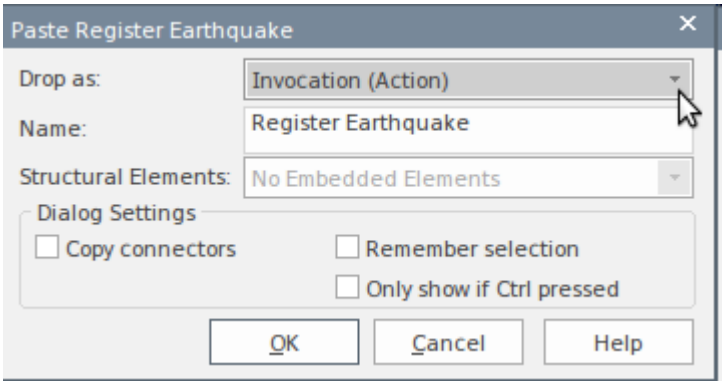
Actions are the behavioral atoms that are connected together to describe the behavior of an Activity, Sub-system, system or one of its parts. Effectively an Activity is made up of a set of Actions that work together to convert items (tokens) that are input into the Activity to items (tokens) that are output by the Activity. The first Action in a sequence will receive its inputs from one of the owning Activity's Input Parameter Nodes and the last Action in the sequence will place the output onto one of the Activity's Output Parameter Nodes. The Actions themselves have input and output devices called Pins - an Action will receive tokens on its Input Pins, perform its work and place the resulting tokens on its Output Pins.



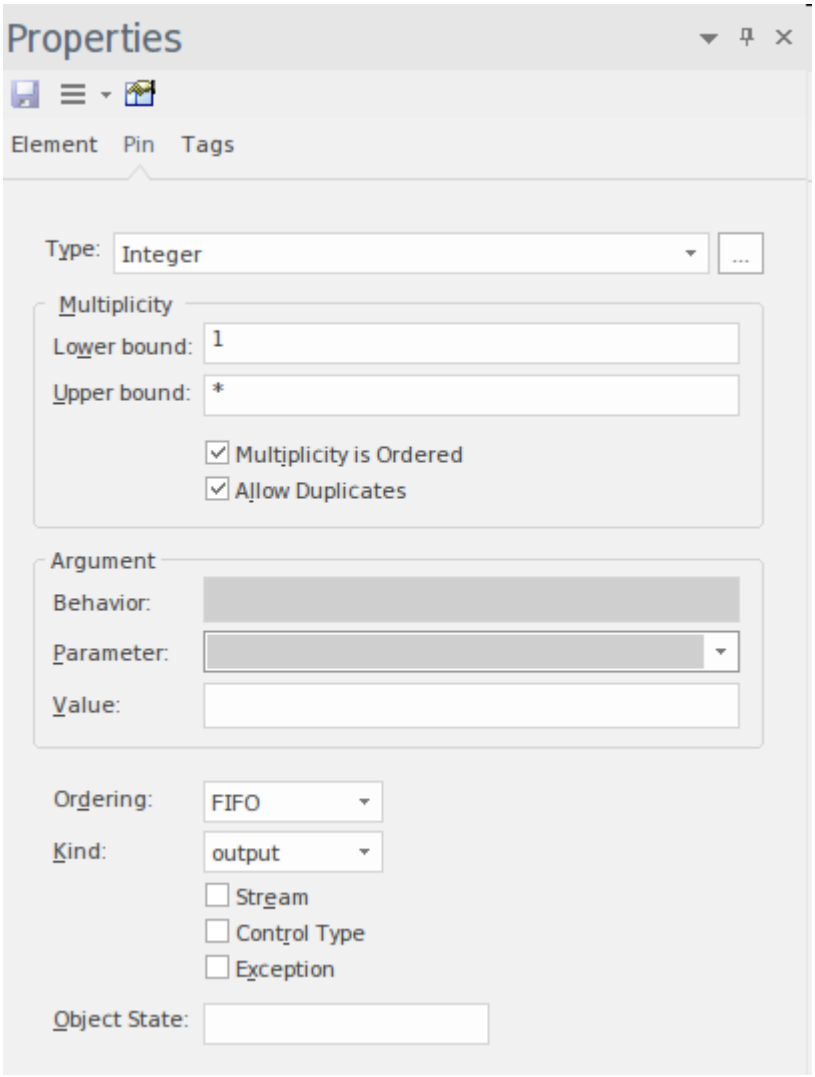
Enterprise Architect has rich support for modeling Actions and their inputs and outputs, and various parts of the user interface can be used while working with these fundamental building blocks of behavior. There are a number of different types of Action available from the Toolbox.



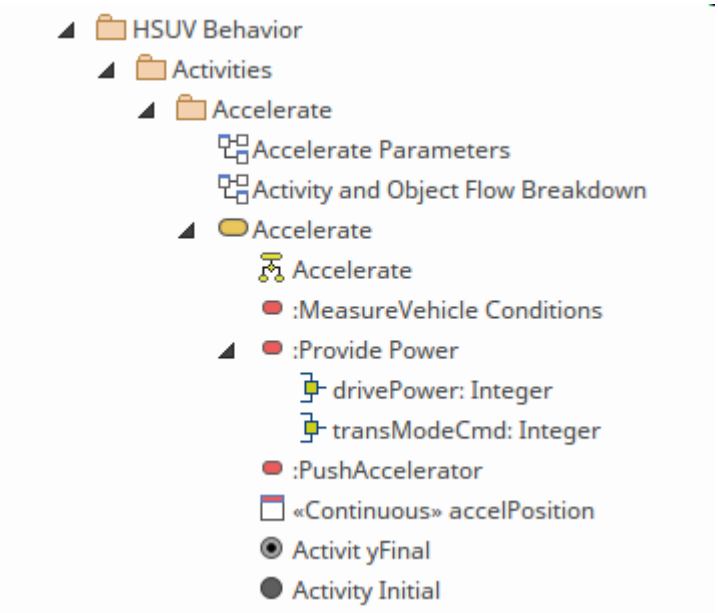
An engineer can add an Action directly from the Toolbox, but it is more common to create Actions from existing Activities that have been defined in hierarchies, as described in the topic *Creating Activity Hierarchies*. To do this an Activity would be dragged NOT from the Toolbox but from the Browser window and dropped onto an open Activity diagram as an Invocation - this has the effect of creating an Action based on the Activity and placing it in the diagram.



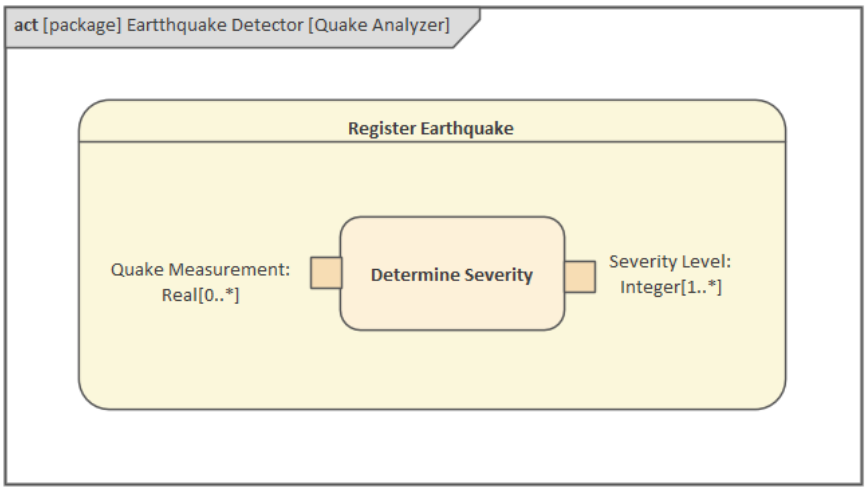
The integrated Properties window makes it easy to work with Actions and their Pins. An engineer can specify a range of properties for the Action and its Pins, including the Pin Name, Type, Multiplicity, Direction and much more. The Properties window can be docked or made to float, and even dragged onto a different monitor; as elements are selected in the Browser window or a diagram the properties can be viewed, created or changed.



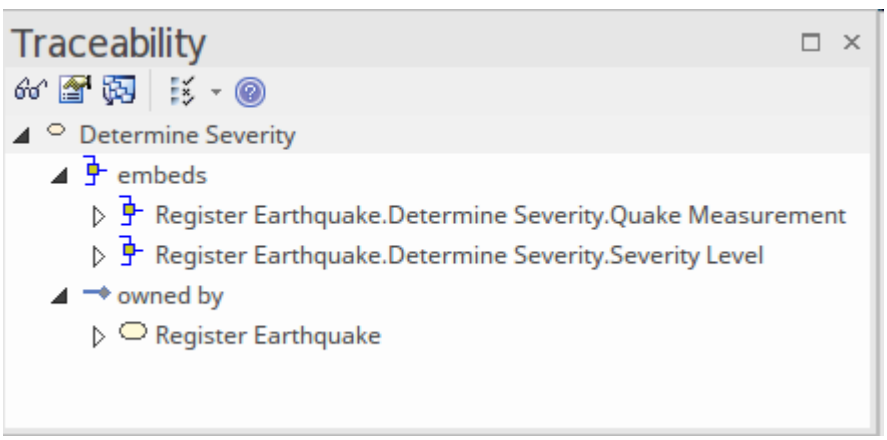
As a diagram is created, the elements that are added are automatically inserted into the Browser window, allowing an engineer to see a structural view of the Activity, Actions, Pins and other elements. Even if the diagram is not open the elements can be selected in the Browser window and edited directly in the Properties window.



A number of these properties will be displayed in the diagram in the Pin label, including name, type and multiplicity. The Multiplicity specifies both a lower bound and an upper bound. The lower bound specifies, for a given execution of the action, the allowable number of tokens that the pin can consume or create, and the upper bound specifies the maximum number of tokens that are consumed or created on that pin.



There is also a wide range of windows that can be useful when working with the Actions, including the Traceability Window, which shows how elements are related regardless of where they are located in the repository; it also displays their structural features such as Pins and Parameters.



Introducing Activity Diagrams

The Activity diagram is a diagram that can be used to show the sequence of Actions that describe the behavior of a Block or other structural element. The Actions are sequenced using control flows, and can contain input and output Pins that act as buffers for items that flow from one Action to another (or from Control or buffer Nodes). The work carried out by the Actions either consumes or produces these items. The items can be either material, energy, or information, depending on the system and the activity being described.

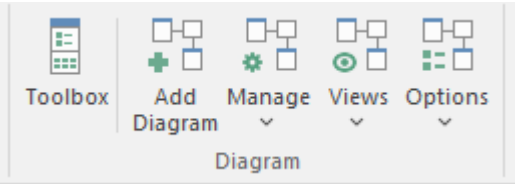
Creating Activity Diagrams

An Activity diagram can be created from a number of places in the User Interface such as the:

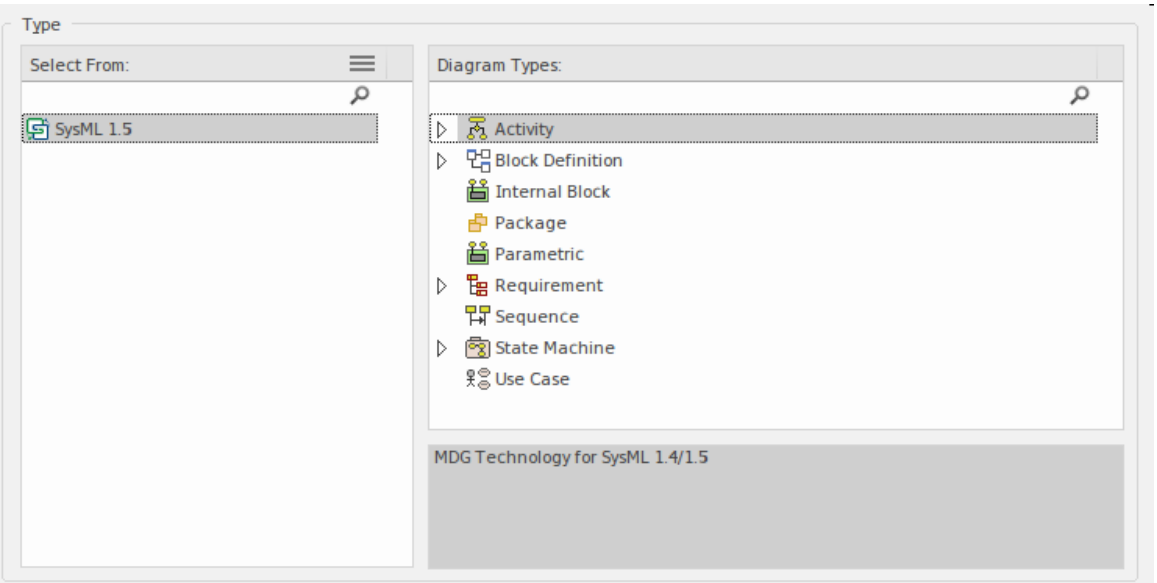
- Design Ribbon - '*Add' Diagram icon on the 'Diagram' panel*
- Browser Toolbar - *New Diagram* icon
- Browser Context Menu - *New Diagram*

We will use the Design Ribbon to create an Activity diagram. Firstly, select the location in the Browser where you want the Activity diagram to be created. As with all diagrams, this can be either a Package or an element, but it is common to insert Activity diagrams into a Package. Once the Package location has been selected in the Browser, select the ribbon option:

Design > Diagram > Add Diagram



Selecting this option will open the *Diagram Builder* tab page in the *Model Builder* dialog, allowing you to change the name of the diagram (which defaults to the name of the Package or element that contains the diagram, as selected in the Browser). With the SysML perspective chosen and the version of SysML selected, a list of diagram types will be displayed from which you select the Activity diagram. When you click on the *Create Diagram* button, a new Activity diagram will be created in the location selected in the Browser. The diagram canvas will be opened, allowing you to start adding elements and connectors that describe the value that the system will provide to its users. Enterprise Architect will also display the 'Activity' pages of the Toolbox, which contain the elements and relationships defined by the SysML specification to be applicable for constructing Activity diagrams. Any number of other Toolbox pages can be opened if required, in addition to the Common Elements and Common Relationships Toolbox pages that are always available.



The most import elements and connectors used with the Activity diagram are:

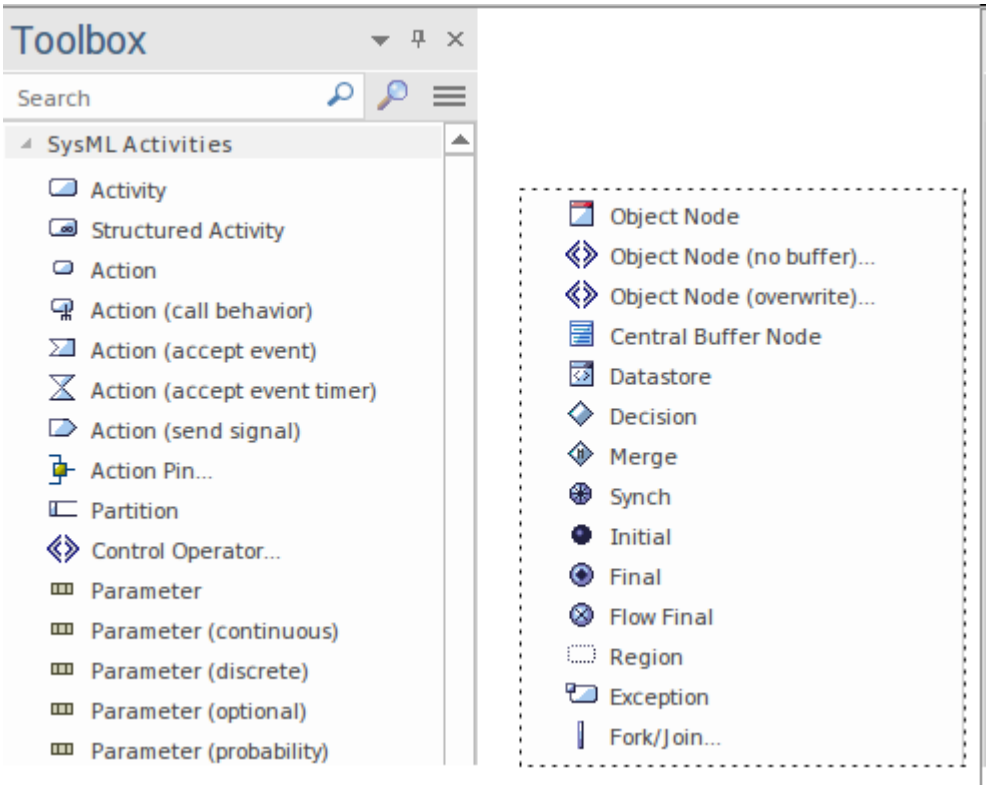
Elements

- Activity
- Action
- Action Pin
- Partition
- Parameter
- Initial
- Final
- Decision
- Fork and Join
- Data Store

Connectors

- Control Flow
- Object Flow
- Interrupt Flow

There are many other elements and connectors that can be used on these diagrams that are important for more advanced modeling; some of these might be needed as modelers become more experienced, or more complex parts of a system's behavior are being described or designed. These include Activity Parameter, Merge, Central Buffer Node, Regions, Fork and Join, Decision and Merge.



Elements can be added to the diagram by dragging-and-dropping them from the Toolbox onto the diagram canvas. It is considered good practice to start with an *Initial* and one or more *Final* elements, which are named appropriately to describe the way the Activity starts and the potentially multiple ways it might finish. Leaving the name of these elements blank or giving them a name that is hackneyed such as 'starts' or 'end' will not help to make it clear to the reader what system or part of a system is being modeled, and can lead to misinterpretation of the diagram. When these nodes have been added and appropriately placed in the diagram, Actions and Object Nodes can be added to the diagram. The Actions can be connected using the Control Flow relationship, defining the sequence in which the Actions will be executed.

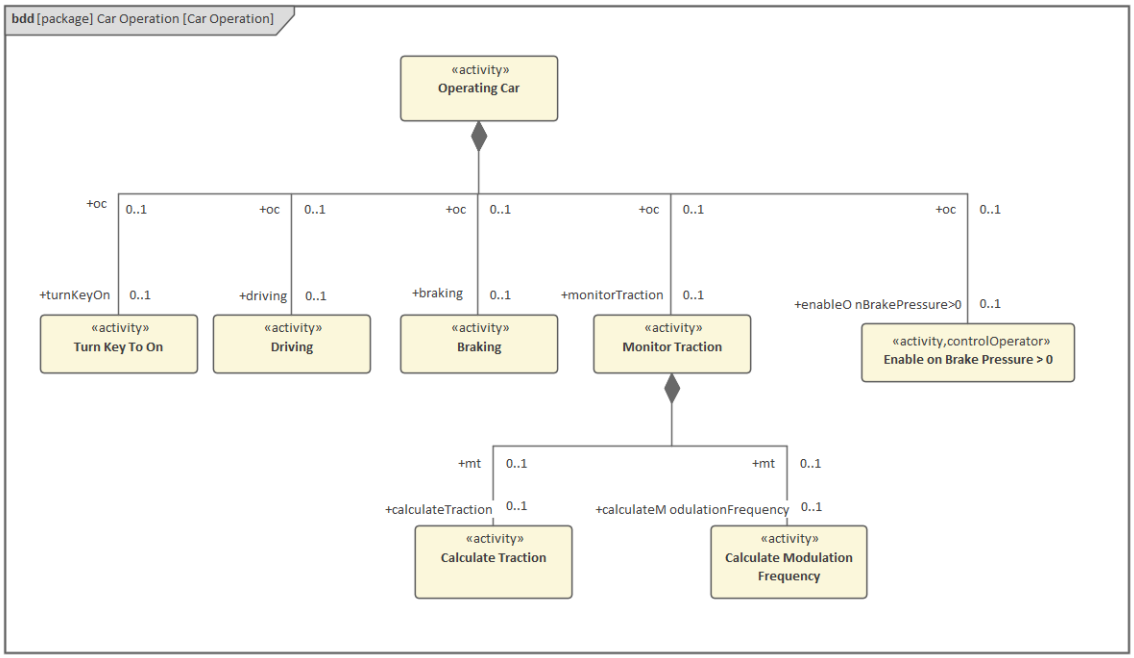
Once a basic diagram has been created, and as knowledge of the domain and the system's behaviors is further accrued, it is possible to structure or embellish the diagram using the additional elements and relationships including:

- Control Structures for Object flows: Forks and Joins, Decision and Merge nodes
- Activity Input and Output Structures: Activity Parameters (Streaming and Non-Streaming)
- Grouping sets of Actions: Interruptible Regions and Edges
- Token Storage Structures: Data Stores and Central Buffers

As stated earlier, the Activity diagram has a rich set of language devices and the engineer is encouraged to use these devices to make the system description richer, but some caution needs to be exercised to ensure that these language mechanisms can be understood by the intended audience.

Creating Activity Hierarchies

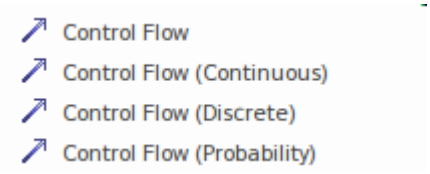
Newcomers to the Systems Modeling Language and Enterprise Architect might be surprised to learn that it is not Activities but Actions that are used on Activity diagrams. Activities that are the classifiers of Actions are typically visualized on Block Definition diagrams. This might seem a little counter-intuitive, but when you understand that the Action is the fundamental atom of system behavior it makes more sense. Activities are classifiers and as such can, like Block elements, participate in a wide range of structural relationships, which is why the relationships such as Associations marked with Composition can be used between Activities.



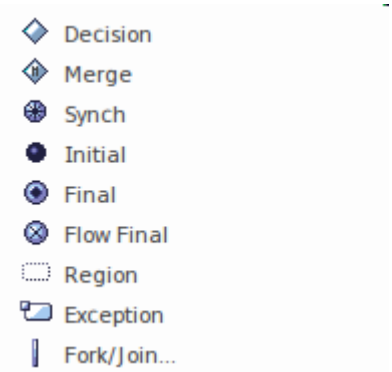
In this diagram a break-down structure has been used where an activity is decomposed into a number of more granular activities using the Composition relationship. There are a number of relationships which have been grouped together to make the diagram more appealing, using one of the flexible line styles available from the diagram context menu.

Specifying Action Sequence with Control Flows

Actions are executed within the context of an Activity, and the order in which the Actions are executed is largely controlled by the use of special connectors called Control Flows. These connectors are directed lines drawn between Actions and act essentially as a conduit for control tokens - allowing the tokens to flow from one Action to the next in the direction of the arrow. An Action cannot commence its work until all incoming Control Flows have received a token; once they have and the Action is executed a token is said to be placed on the outgoing Control Flow, which implies it will travel to the next Action in the sequence. Control Flow relationships are available from the 'Activity' pages of the Diagram Toolbox

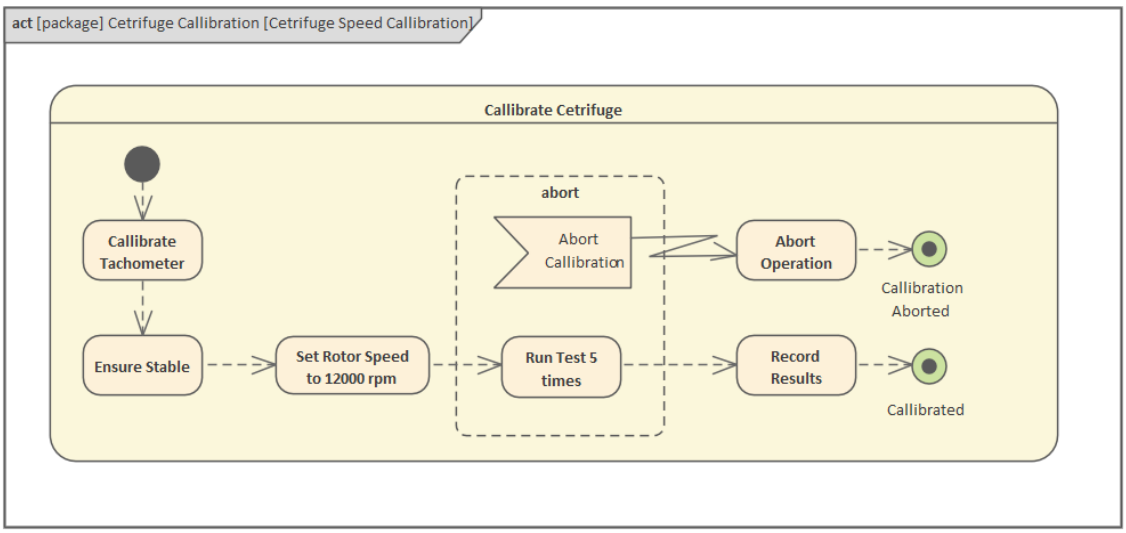


There are also Control Nodes that can be used with Control Flows to orchestrate the way the Flows work with the use of Forks, Joins, Decisions and Merges. There are three specialized nodes: Initial, Final and Flow Final, that act as the start and finish of the flow respectively. The Final (formally Activity Final) node is used to indicate that when a token arrives the entire Activity terminates, whereas the Flow Final will consume incoming tokens but will have no effect on the enclosing Activity.



Interrupting Normal Flow

There are a number of circumstances during the execution of an Activity when a modeler might want to specify a way of stopping the behavior in a part of an Activity. For example, in a real-world scenario a user might get part way though using a machine function such as calibrating a centrifuge, and then decide that they want to end a particular part of the calibration process. This scenario might be provided by a Cancel button on the interface. The SysML allows this situation to be modeled using an Interruptible Region and an Interrupting Edge. The notation allows the Interruptible Region to be drawn to include a number of elements such as Actions and other Nodes. Typically, when something unusual occurs an Event is fired in the Activity and received by an Accept Signal Action; this element has no incoming control flows and a single outgoing Interrupting Edge, which targets an Action that resides outside the Region.

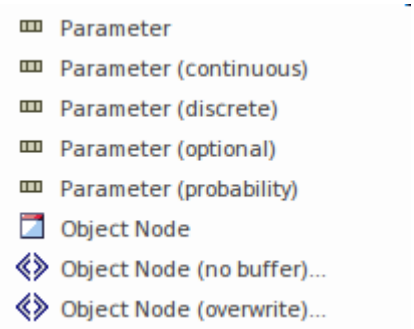


In this diagram, an engineer has modeled the process used for performing a speed calibration for a centrifuge. The centrifuge calibration process can be interrupted for various reasons; for example, if the centrifuge has become unstable or the operator is called away to perform other duties. An Accept Event Action is used to show that the Activity has a mechanism to listen for a required interruption within a specified Region of the Activity. The special Interrupt Flow connector then targets an Action outside the Region, which is used to shut down the centrifuge; finally this flows to the calibration Activity being terminated.

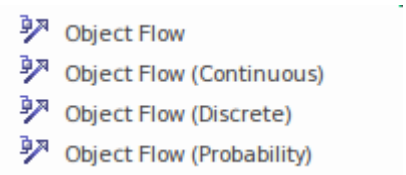
Specifying Item Flow with Object Flows

Activities, and the Actions they are composed of, typically do work by processing items that arrive on Input nodes and, when the work is complete, placing the resultant items onto Output nodes. As was discussed earlier, Activity modeling in the SysML is based on a branch of mathematics called Petri Nets, which is concerned with discrete State Event systems. The items that arrive at the input structures must pass through the graph of Activities and their contained Actions in an orderly and systematic way. The passage is created by Object Flows that act as conduits to carry tokens from one node to another. The tokens represent a number of different types of 'thing' including information, structures or physical items such as solids, liquids and gases. There are thus two important parts to the way that items pass through the Activity - the nodes that act as origins and destination of tokens, and the connectors (conduits) that transmit the items.

Enterprise Architect has full support for modeling these flows, and when a diagram is created or opened for editing, the Toolbox contains the Object Nodes as shown:

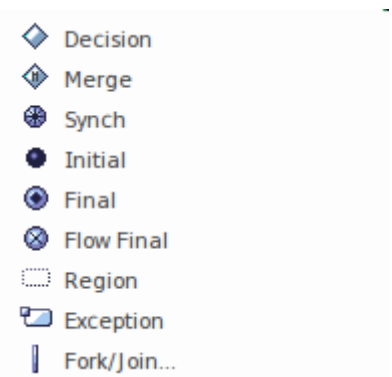


It also contains a section that lists the Object Flow relationships that can be used to connect the nodes, creating the conduit for the tokens to flow from one node to another.



Orchestrating the Flow of Tokens

When modeling complex systems there is often the need to create more elaborate paths (conduits) for the token flow, such as forking and joining paths to allow tokens to be sent to a number of object nodes so that work can be done simultaneously, or to allow tokens to be routed down a particular path based on some condition. These Control Nodes control flow and are grouped together on a page of the Diagram Toolbox.



Enterprise Architect allows the connectors to be manipulated to create any path that is required. This can be done by utilizing the line styles from a connector's context menu; the most flexible of these is the Custom Line Style, but there are several other styles that are very useful. A modeler can also fix the connector ends to a specific part of the Source or Target element.

Storage for Tokens in Transit

During the execution of an Activity it is sometimes necessary to store tokens for a longer period of time than is possible with Activity Parameters and Action Pins, which act simply as temporary storage devices. A common circumstance is when a number of Actions require access to a stream of tokens - the tokens can be stored in a Central Buffer and made available to the nodes that require them. The Central Buffer accepts all tokens on its incoming flows, then makes the tokens available to downstream nodes; once accepted, the tokens are then removed from the buffer.

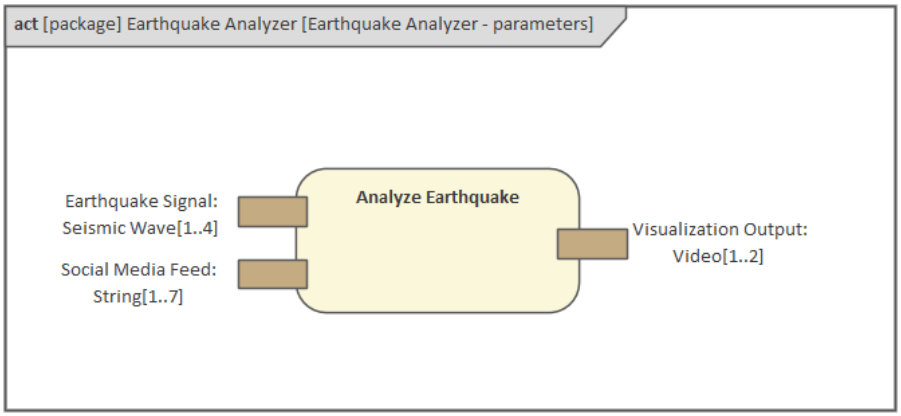
The Central Buffer can be created by dragging the 'Central Buffer' icon from the Toolbox onto an open Activity diagram; it can then be connected to other object nodes using Object Flows.

Thus the Central Buffer can, during the execution of the Activity, be replete with tokens or empty depending on the consumption of tokens. Another type of node is the Data Store, a specialization of the Central Buffer where, as tokens are consumed by downstream actions, a copy is made and stored back in the buffer. This has the effect of the Data Store having the appearance of a permanent store - but only for the lifetime of the Activity's execution.

The Data Store can be created by dragging the 'Data Store' icon from the Toolbox onto an open Activity diagram; it can then be connected to other object nodes using Object Flows.

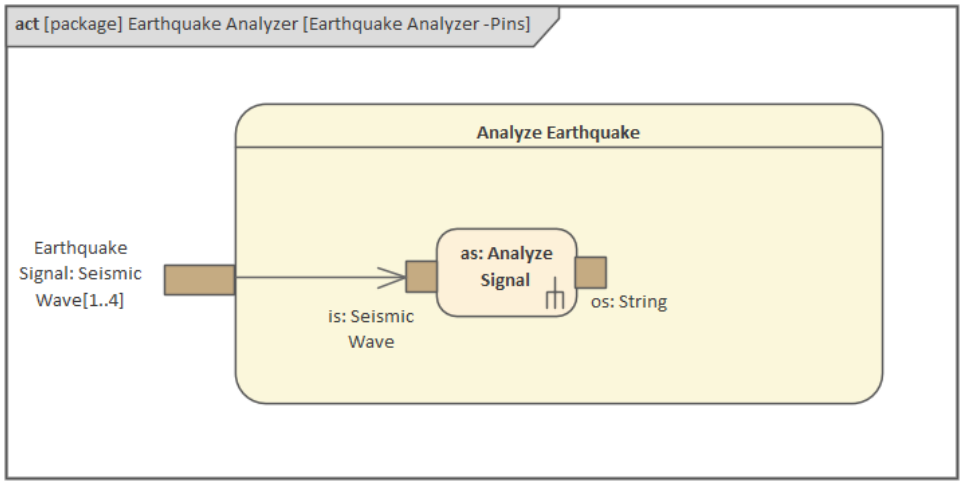
Modeling Inputs and Outputs with Parameters and Pins

Activities and their constituent Actions are the workforce of systems; while structural elements such as Blocks and Parts define the structure or anatomy of a system, the Activities define the physiology. When an Activity is executing we see the structural elements being called into action to accomplish some type of system behavior. Much of the work that a system does, and the behaviors that define what the work is, are dependent on system inputs that the executing Activity consumes in order to produce outputs.

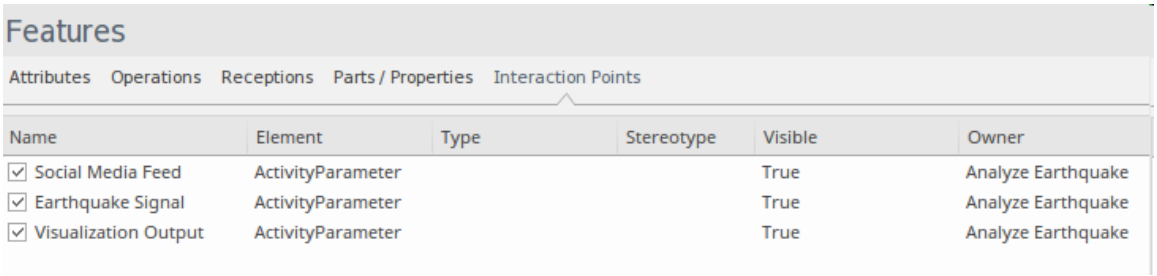


Inputs and outputs vary greatly between systems and can include things such as control signals, materials, light, fluids, energy, numbers and information. The inputs and outputs are called parameters, which can be typed and can have multiplicities. Typing ensures that the Activity specifies what kind (type) of 'thing' it is expecting. Thus if a distiller had an input parameter with a type of liquid defined or, even more specifically, a liquid-contaminant, then the Activity would be ill-formed if it received a gas or an Integer Value as an input on this parameter. The types can be any one of a defined set ranging from a simple Integer to a compound Structure. Inputs and outputs can be typed by a Block, so that you have a well defined structural element - for example, a grocery item that passes through a self scanning system at a supermarket checkout. There is a range of other properties that can be defined for a parameter, including Streaming or Non Streaming, Multiplicities, and Direction. Streaming is used when there is a continuous flow into the parameter, such as with a fluid, or a communication or information signal such as an audio or visual stream. Multiplicities define the upper and lower bounds of the number of tokens that are consumed by an input parameter or produced by an output parameter. While Direction defines if the parameter is receiving input (in) or producing output (out) or a combination of both (inout).

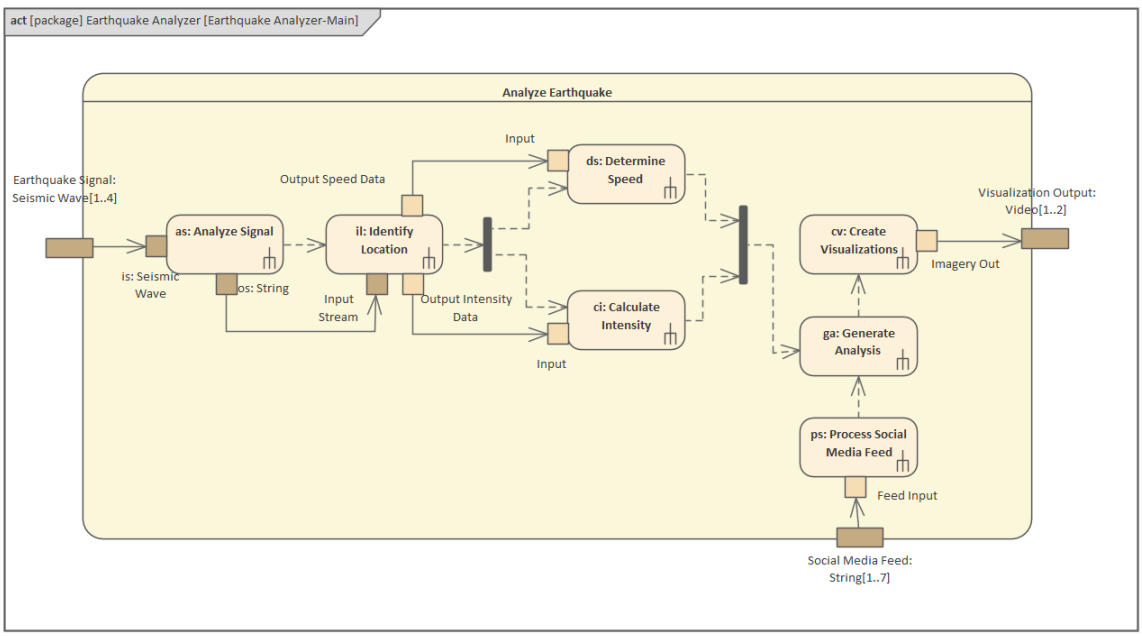
When Activities are placed on an Activity diagram as invocations they are represented by Actions, and any Parameters owned by an Activity will be modeled as Pins on these Actions. The Pins receive tokens on incoming Object Flows and the owning Action performs its work and places any specified number of tokens on the output Pins. The Pins can have a simple type such as an Integer, a complex Structure such as a matrix or even a Block such as a video stream. Multiplicities specify a lower and upper bound that define the minimum and maximum number of tokens permissible to arrive and depart from any given Pin. This unfinished diagram shows an Action with an input and an output Pin and the transmission of the tokens from the Owning Activity's input parameter along the Object Flow.



The Parameters and Pins are collectively known as Interaction Points, signifying that they are locations where an element interacts with its environment; they can be selected for inclusion on a diagram by using the multi-purpose Features window.



Enterprise Architect allows you to create a diagram that shows the owning Activity as a container for the other Activities included on the diagram as Actions. In this diagram, the Activity Parameters defined on the owning Activities are expressed as Pins on the boundaries of the Actions that have been included as invocations of the Activities. The diagram shows an Activity with two input parameters and a single output parameter. The inputs in the form of tokens can be traced through the diagram as they arrive at Pins. Once the Action has completed its work, tokens are placed on the output Pins. The Control Flows show the sequencing of the enclosed Actions. Notice that a Fork and Join are used to show that two Actions can be carried out in parallel. Notice also that a number of the Pins have been defined as a stream, which is indicated on the diagram by the solid color of the Pin.



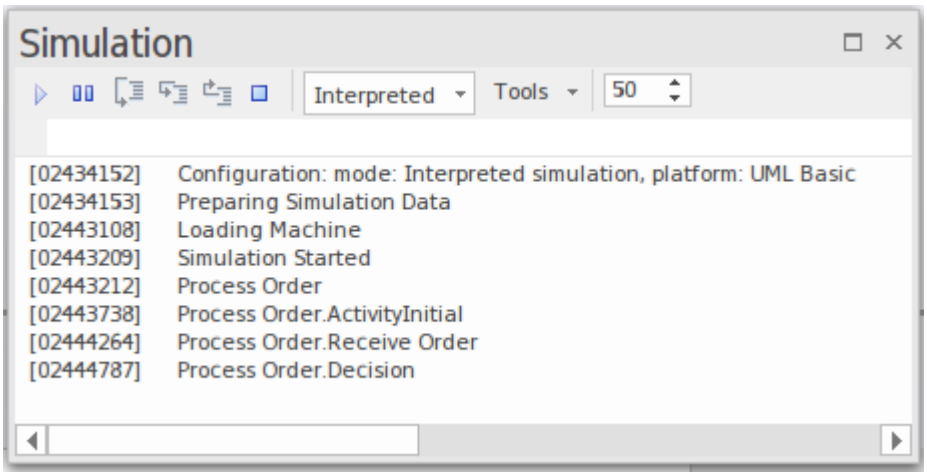
Visualizing Activities with Simulations

Any of the SysML Activity diagrams in your models can be simulated using the built-in dynamic model Simulator. This provides a compelling way of visualizing the diagrams and is useful for running demonstrations or walk-throughs with the user and others in the engineering community.

Using the Model Simulator, you can simulate the execution of conceptual model designs containing behavior. When you start a Simulation, the current model Package is analyzed and a dynamic Simulation process spawned to execute the model. As the Simulator analyzes and works with UML constructs directly, there is no requirement to generate intermediary code or compile simulation 'executables'. This results in a very rapid and dynamic simulation environment in which changes can be made and tested very rapidly.

Simulation Window

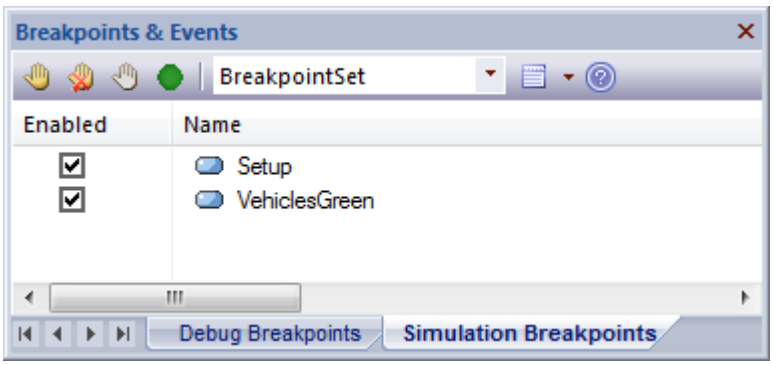
The Simulation window provides the main interface for starting, stopping and stepping through your Simulation. During execution it displays output relating to the currently executing step and other important information. See the Run Model Simulation Help topic for more information on the toolbar commands.



Note the text entry box just underneath the toolbar. This is the Console input area - here you can type simple JavaScript commands such as: *this.count = 4;* to dynamically change a Simulation variable named 'count' to 4. In this way you can dynamically influence simulation at run-time.

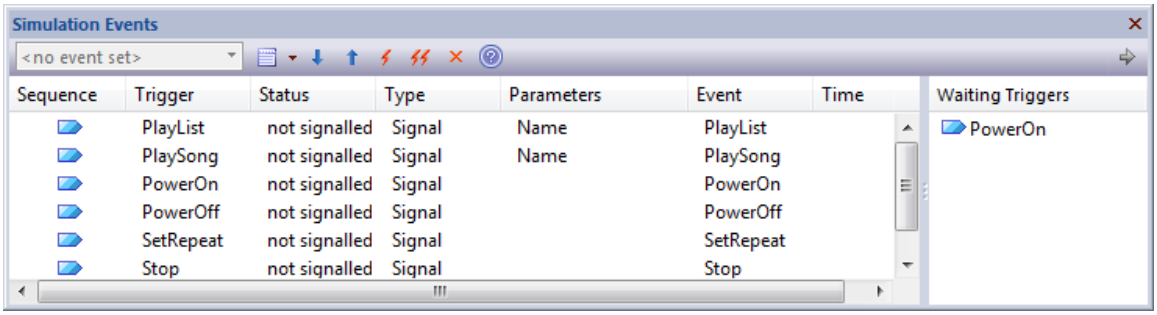
Breakpoints and Events Window

The Simulation process also makes use of the 'Simulation Breakpoints' tab of the Breakpoints & Markers window ('Simulate > Dynamic Simulation > Breakpoints'). Here you set execution breakpoints on specific elements and messages in a Simulation. See the Simulation Breakpoints Help topic for more details.



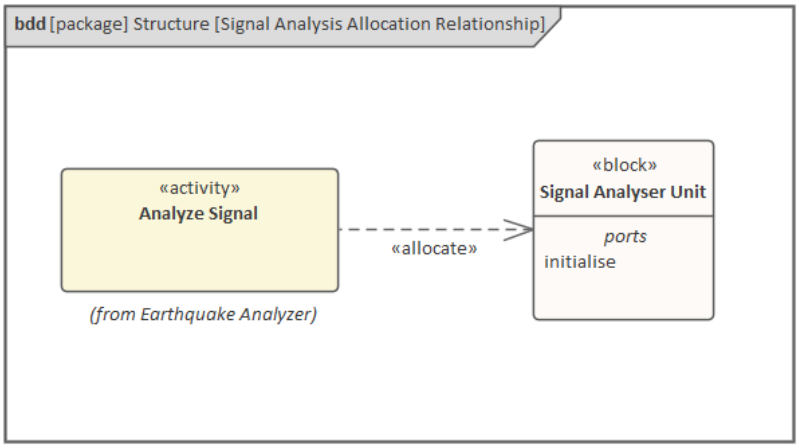
Simulation Events Window

The Simulation Events window ('Simulate > Dynamic Simulation > Events') provides tools to manage and execute triggers. Triggers are used to control the execution of StateMachine transitions.

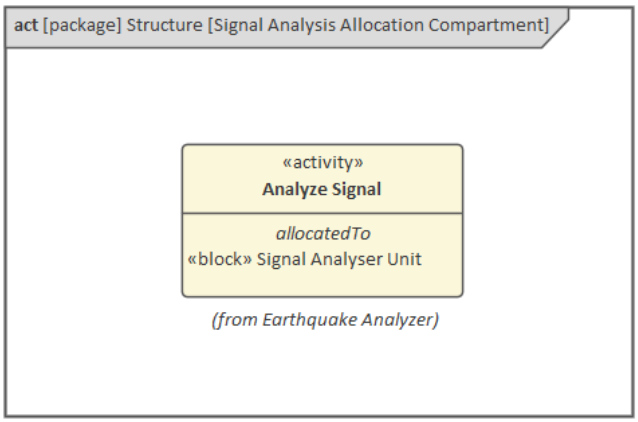


Allocations and other Relationships

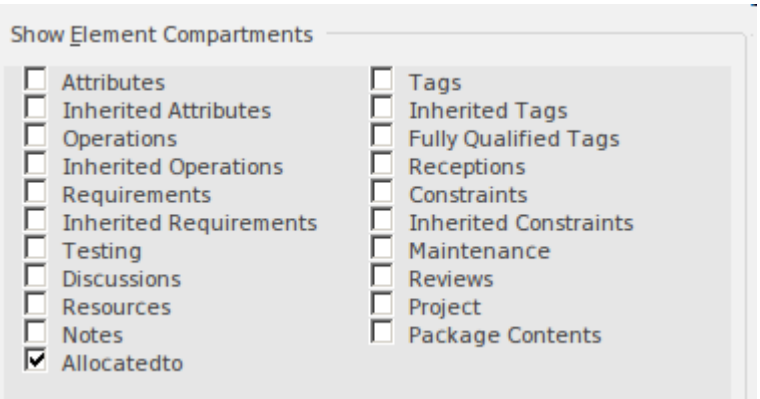
A fundamental aspect or discipline in Systems Engineering is to relate structural elements with behavioral elements. These two important aspects of a model will become intertwined as a model develops, but when a number of engineers start work to define the system it is often difficult to say exactly how the behavior and the structure will be related. The Allocation relationship is particularly useful in these situations. It can be used as a way of showing the relationship between behavioral elements and structural elements that will inform the more rigorous modeling that will be employed as the notions described in the model become more certain.



Enterprise Architect also supports a number of other ways of representing the Allocation relationship, including as a compartment in either the behavior or structure element.



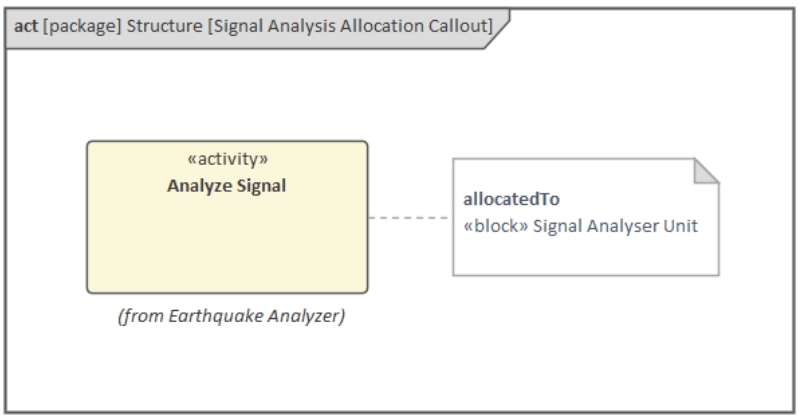
Any element that allows compartments can be configured to hide or show any number of available compartments. The list of compartments is specific to an element and is dynamic, meaning that a compartment will only be visible if the element participates in one or more relationships of the specified type and the relationship is not visible on the containing diagram. The same options can be used to display a range of other structural or semantic aspects of the elements as shown here.



It is also possible to show the relationship in a callout notation, where a note is connected to the element and displays the name of the relationship and the details of the related element. This diagram shows the notation for an Activity, showing the Block that it has been allocated to. To achieve this an engineer must:

1. Ensure the relationship is displayed in the diagram.
2. Select the relationship and display the context menu.
3. Choose the 'Create Linked Note' option.

This callout notation can be used with any type of SysML element or relationship, and is a useful way of displaying the relationship for some types of audiences.

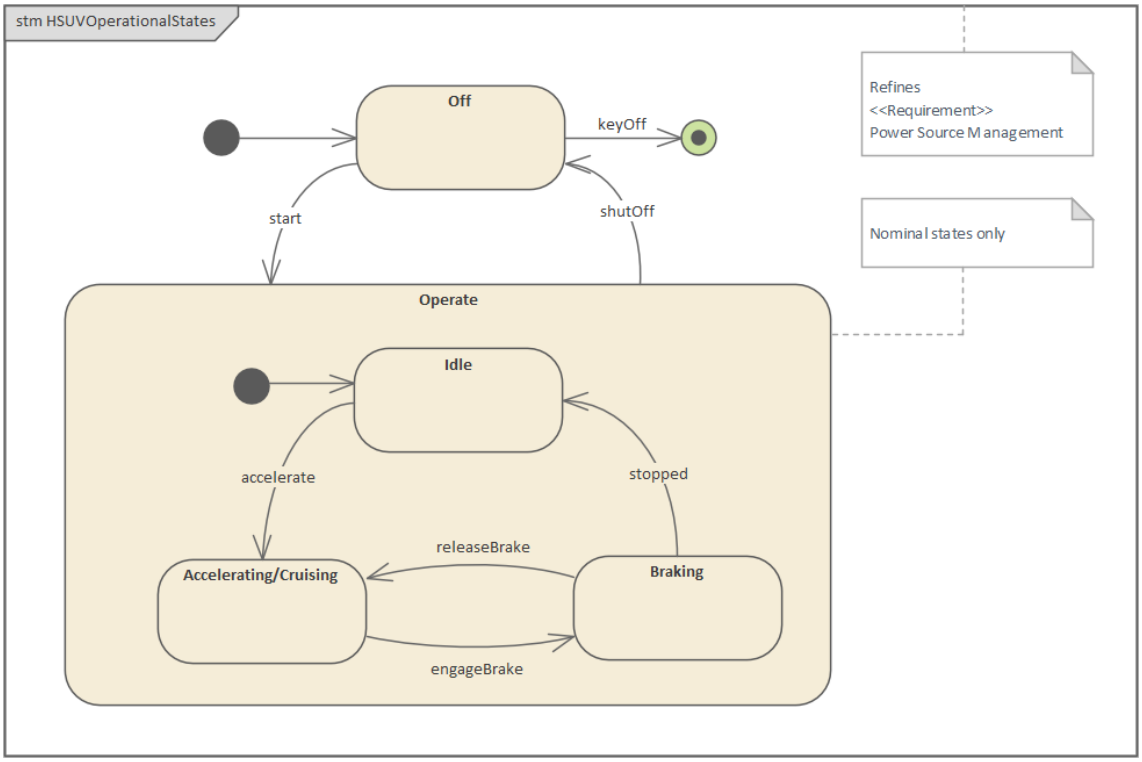


Modeling Change with StateMachines

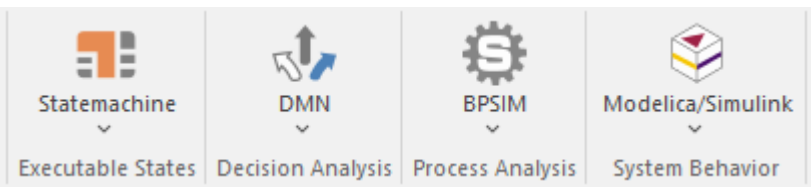
Our world is in constant flux as 'things' change or evolve, passing from one state to another. Water freezes, glaciers deform and flow, ice melts, traffic lights cycle between green, amber and red, aircraft take off, climb, cruise, descend and land. The SysML StateMachine is used to describe how structure, in the form of Blocks, changes its state in a time-boxed life cycle. Our concern is not with the structure of the Block Instance but its behavior, which can in turn impact its structure. We are not interested in every single state a 'thing' can be in but rather the significant states. So the important states for water molecules could be a *solid*, *liquid* or *gas* but we are not normally interested in liquid water at a temperature of 67 degrees Centigrade. If we were looking at a movie reel of an object's life time, a StateMachine would pick out the significant frames where important and relevant changes occurred.

Deciding what is relevant is the prerogative and privilege of the modeling engineer, and the same Block could have any number of StateMachines defined by the same or different engineers. An aircraft's state could be modeled from the perspective of passenger embarkation and disembarkation, from the perspective of its maintenance schedule, from the perspective of lift, or any number of other perspectives.

This StateMachine diagram describes the operational states of an SUV motor vehicle. The diagram uses Composite States, which nest States inside other States. There are three high level States - Off, Operate, and the unnamed End State. The Operate State has a number of sub-states, namely Idle, Accelerating/Cruising and Braking. Together with the transitions this describes the states of the vehicle as it Starts, Accelerates, Breaks, Stops and finally when the ignition is turned off.

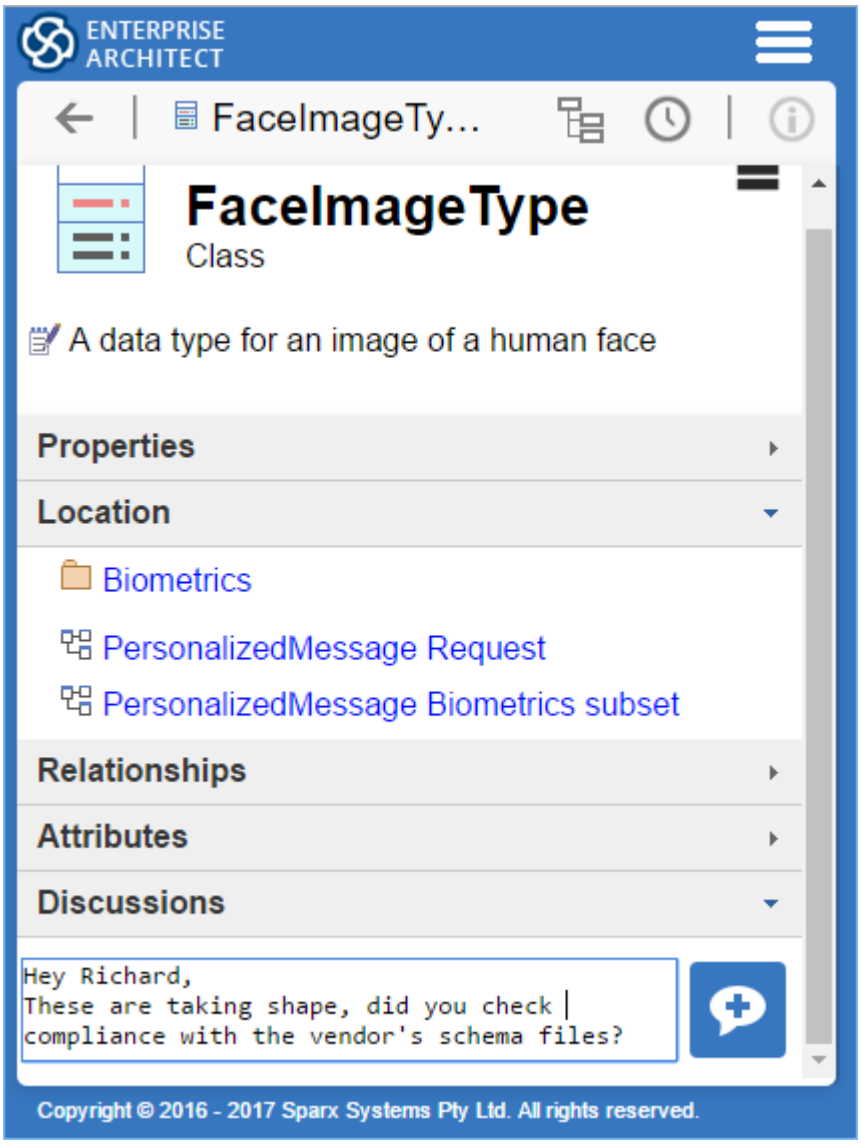


Using Enterprise Architect an engineer can create StateMachines and define the transitions from one state to another, including Events that trigger state change and Actions that are fired. In addition to these standard modeling representations, the tool has a range of features that can help to visualize and reason about this important linguistic mechanism that ties structure and behavior together. One of these facilities - which we will look at in this topic - is Executable StateMachines, available from the Simulate ribbon.



StateMachines can be defined at any level of granularity as they are an expression of a Block's behavior. Many newcomers to SysML are confused about this point. Because a Block can represent something very simple - such as a switch on a submarine control panel - or something complex like the submarine itself, so too can a StateMachine represent the states of both the switch and the submarine. The two StateMachine models could have the same complexity, even though the things being modeled are themselves clearly at either end of the spectrum when it comes to complexity.

StateMachine diagrams can appear quite simplistic to the inexperienced modeler, but they are highly effective tools for the description and analysis of complex problems that cannot be solved in other ways. It takes a different mindset and approach, and often the kernel of the problem is focused on the selection of the level of Block, its context and the perspective for the StateMachine, rather than the details of the diagram. Often the best results are achieved heuristically by a number of engineers working together. This can be accomplished using Enterprise Architect's collaboration features, allowing engineers dispersed geographically to communicate within the model, either by mail, discussions, chats and formal reviews via the desktop client, or in a Browser on a Smart Phone, Tablet or Notebook.



The StateMachine has its origin in discrete event-driven Behaviors, using a finite StateMachine based on an object-oriented variant of David Harel’s StateCharts formalism.

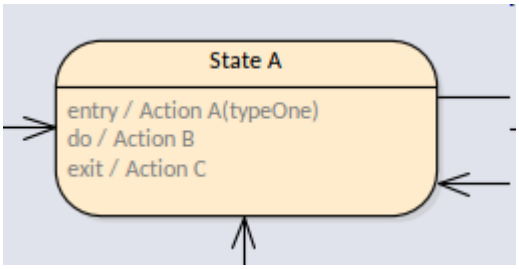
States and Behaviors

A State is created within the context of a StateMachine and is used to model the engineer-defined significant condition of the owning Block. It is important to remember that the StateMachine is describing the lifetime of the Block from a particular perspective, and the States must be defined from this point of view - not all States, but the significant ones such as On/Off, Open/Closed, Green/Amber/Red or Ice/Water/Vapor. Formally, a State models a situation in the execution of a StateMachine Behavior where some invariant condition holds for a particular duration.

A Block typically spends some time in a given State, which might last nanoseconds or days depending on the context; this temporal aspect is not typically codified in the models but can be set in a simulation. There are three behaviors (called Actions) that can be defined with respect to any given State:

- *Entry* - Fired when a State is entered
- *Do* - Fired after the Entry behavior and before the Exit behavior
- *Exit* - Fired before the State is exited

This diagram shows how these States are represented in a StateMachine diagram. Enterprise Architect can conditionally display these and other compartments at the level of an individual element, or collectively for all elements on the diagram.



It is also important to note that the Final node is formally also a State, but it does not have the same semantics of behavior as the States represented on diagrams as rectangles with rounded corners.

There are three fundamental types of State, each of which is important for modeling a different class of problem:

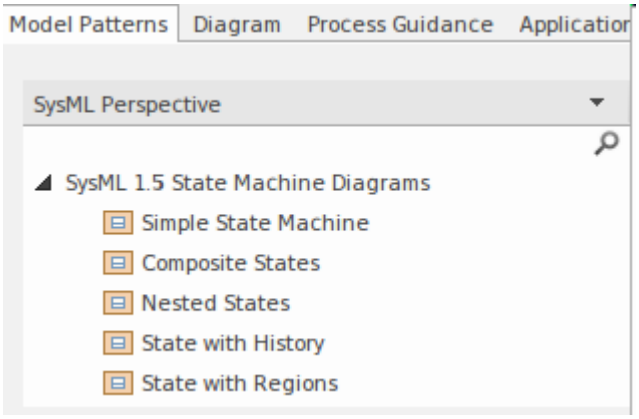
- *Simple* State - does not contain internal States
- *Composite* State - contains a least one region that owns States
- *Submachine* State - represents an entire StateMachine that is nested within the owning State

Enterprise Architect allows you to model each of these State types, and a modeler can use them productively in StateMachine diagrams to express real world engineering problems and solutions.

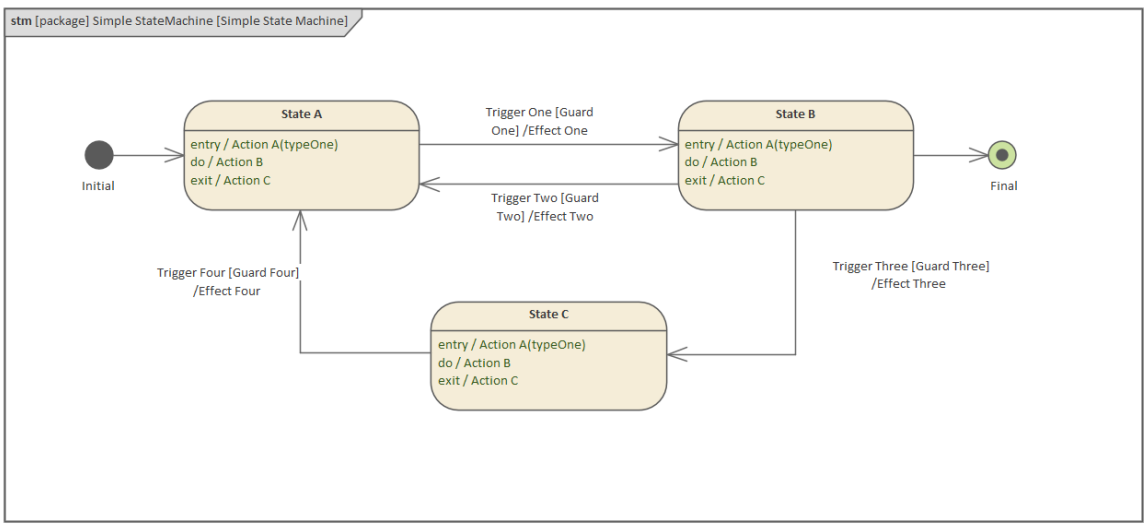
Introducing StateMachine Diagrams

The StateMachine diagram is one of the nine core SysML diagram types; it is used to create and visualize the behavior of Blocks as they change states. The key elements on the diagram are States, Transitions and Pseudostates. The States represent the significant time in the lifetime of the Block from a particular perspective, the Transitions represent the movement from one State to another and the Pseudostates, as we will see, act as traffic controllers that influence the way that transitions work.

Enterprise Architect helps a modeler to create any number of StateMachine diagrams, and each diagram can have any number of States, Transitions and Pseudostates. Each of these diagram elements and connectors can in turn have other information added that will embellish the diagrams with more detail. The application has a pattern library productivity tool that is very useful for newcomers and welcome equally to experience modelers. This screen capture shows the list of model patterns that can be used to create StateMachine diagrams.



The pattern can be used to create a number of different StateMachines; in this example we create a simple (single region) diagram that has all the appropriate detail added to the States and the Transitions. A modeler can create this diagram in the appropriate location in the repository and then edit the States and Transitions and diagram to make it suit their own modeling context. The initial StateMachine diagram created from the pattern will resemble this:



This handy feature prompts engineers to complete details such as the Trigger and Guard conditions on a Transition, or the Entry and Exit actions on a State, that they might not otherwise have been aware of - the result being diagrams that are aligned with best engineering practice, producing better outcomes for customers.

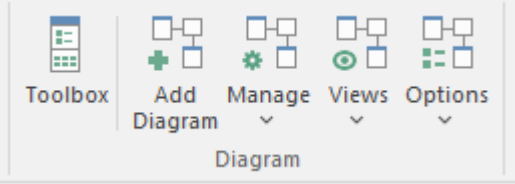
Creating StateMachine Diagrams

A StateMachine diagram can be created from a number of places in the User Interface by using the:

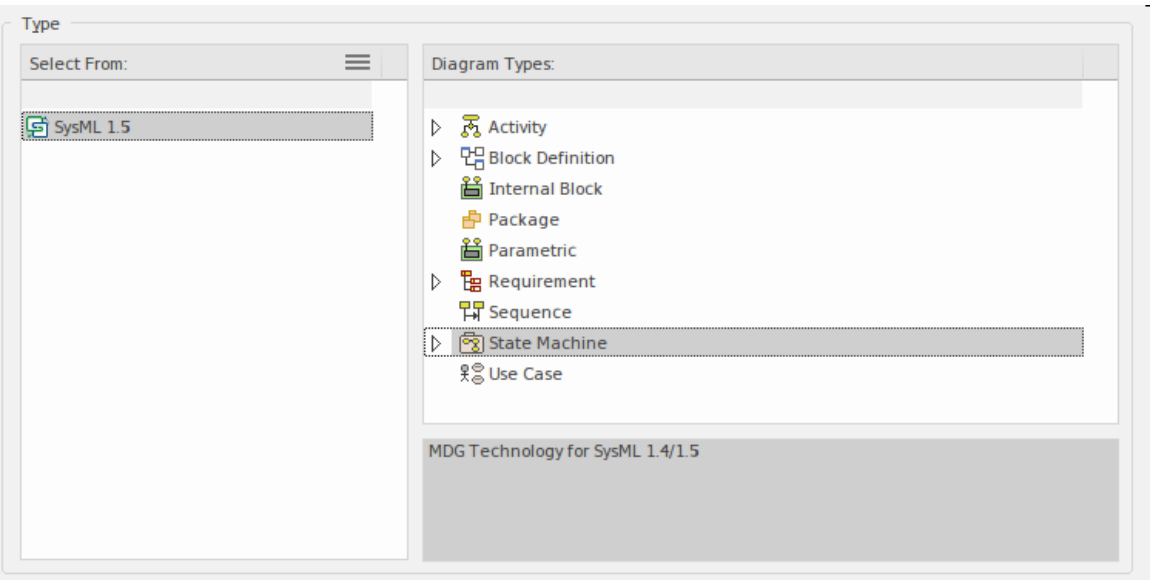
- Design ribbon - 'Add Diagram' Icon on the 'Diagram' panel
- Browser window toolbar - 'New Diagram' icon
- Browser window context menu - 'New Diagram' option

We will use the Design ribbon to create a StateMachine diagram. Firstly you select the location in the Browser window where you want the StateMachine diagram to be located. As with all diagrams, this can be either a Package or an element, but it is common to insert StateMachine diagrams into an element such as a Block to describe the important phases in a Block's lifetime. Once the location has been selected in the Browser window then select:

Ribbon: Design > Diagram > Add Diagram



Selecting this option will open the *Diagram Builder* tab page in the *Model Builder* dialog, where you can choose the diagram type and specify the diagram name diagram; the name will default to the name of the Package or element that contains the diagram, but you can change it. With the SysML perspective chosen and the version of SysML selected, a list of diagram types will be displayed from which you can select the StateMachine diagram. Once the *Create Diagram* button is selected a new StateMachine diagram will be created in the location selected in the Browser. The diagram canvas will be opened, allowing you to start adding elements and connectors that describe the important phases in the lifetime of the subject. Enterprise Architect will also display the StateMachine pages of the Diagram Toolbox, which contain the elements and relationships defined by the SysML specification to be applicable for constructing StateMachine diagrams. Any number of other Toolbox pages can be opened if required, in addition to the Common (Elements) and Common Relationships pages that are always available.



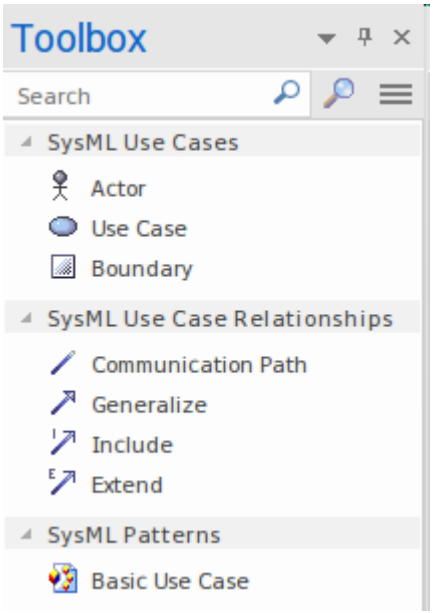
The most important elements and connectors used with the StateMachine diagram are:

Elements

- State - defines a significant phase in an entity's lifetime
- StateMachine - defines a set of States
- Initial - defines the entry point to a Region
- Final - defines the last State an entity will have
- History - acts a memento or bookmark when a State is exited and re-entered

Connectors

- Transition - represents the movement from one State to another

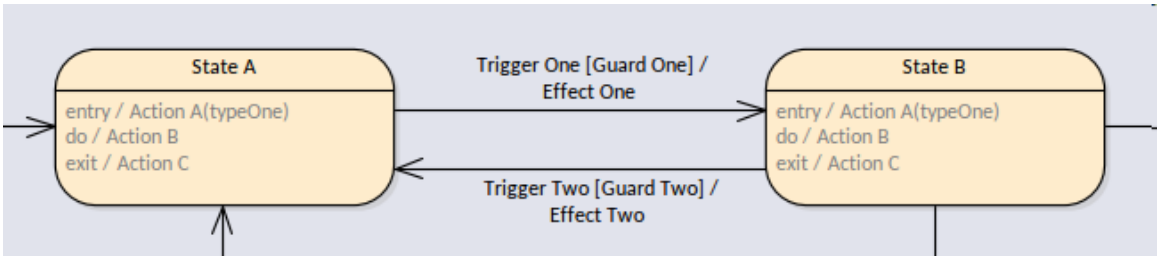


Elements can be added to the diagram by dragging-and-dropping them from the Toolbox pages onto the diagram canvas. It is considered good practice to start with an Initial and one or more Final State elements, which should be named appropriately to describe the way the StateMachine starts and the potentially multiple ways it might finish. Leaving the name blank or giving it a name that is hackneyed such as 'starts' or 'end' will not help to make it clear to the reader what system or part of a system is being modeled, and can lead to misinterpretation of the diagram. With these nodes added and appropriately placed in the diagram, States and Transitions can be added, thus defining the important phases in the lifetime of the entity being modeled.

Once a basic diagram has been created, and as knowledge of the domain and the system's behaviors is further developed, it is possible to add Triggers, Events and Guards to the Transitions, and Entry, Do and Exit behaviors to the States. The newcomer can often perceive these diagrams to be trivial, but they can reveal profound insights that would not otherwise be possible to see.

Triggers and Transitions

The majority of connectors that you see on a StateMachine Diagram will be Transitions; these are the lines that connect one State to another, indicating the allowable ways the owning Block (instance) can change. The order in which they change and the behaviors that are executed will depend on the conditions and real world context of the Block. For example, a traffic light might flash amber until the maintenance engineer has rectified a fault, or an aircraft might maintain a holding pattern until the Control Tower at the destination airport gives landing clearance. This diagram shows two transitions that are directed in different directions, effectively creating the possibility of a cycle between the two States.



The transitions always originate from one State and target another, including the special case of a self-transition where the origin and the target are one and the same. The lines in the diagram have a label that can display a number of different options: *Trigger*, *Guard* and *Effect*. We will discuss these options in detail because they express important semantics about the transitions, including whether the transition will be executed at all. A transition can be in three conditions:

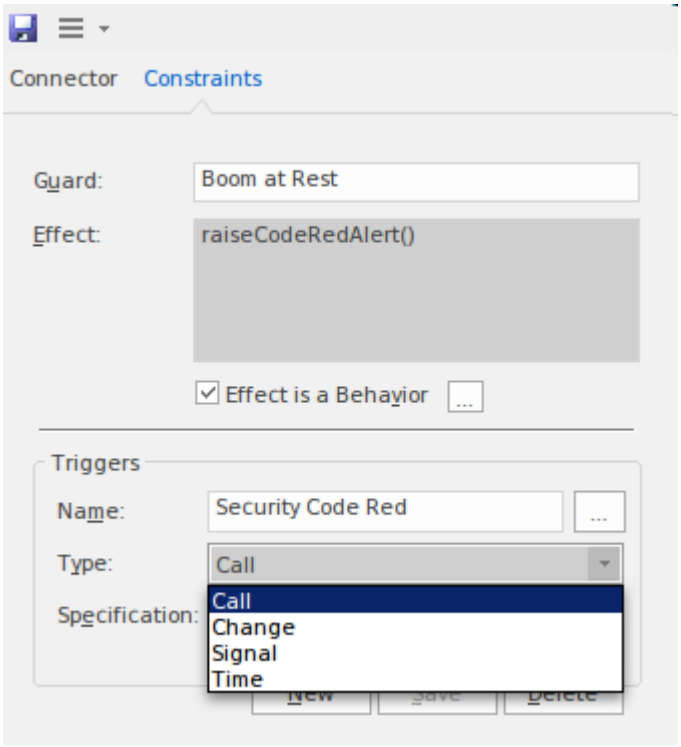
- *Reached* - the originating State (or Vertex) is active and ready to complete its behaviors
- *Traversed* - the transition is being executed (including any defined effect behaviors)
- *Completed* - the target state has been reached and is ready to execute entry behaviors

These terms will be useful to system engineers and others when working collaboratively, discussing the execution of a StateMachine and its description of the behavior of the owning Block.

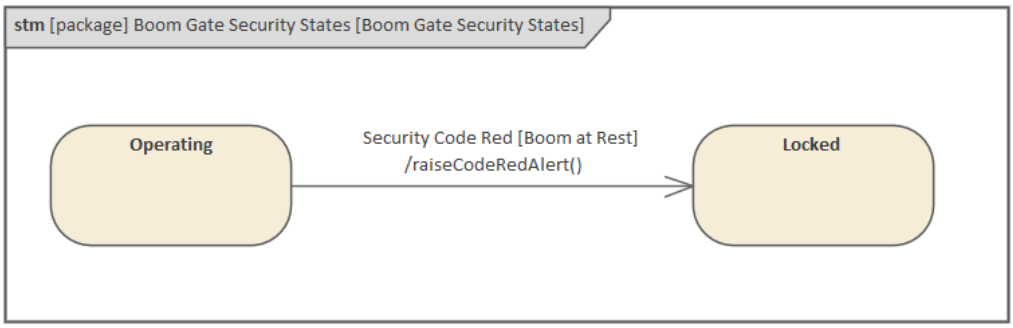
Triggers

Triggers are the initiators of a transition and are mapped to events that are said to trigger event occurrences. It is these triggers and their related events that result in a transition executing (firing) and the owning Block moving from one state to another. When a state is active it is effectively waiting to be triggered by an event, and as long as its entry action is completed - regardless of any other factors - it is ready to receive and respond to events.

This screen capture demonstrates how the Trigger, Guard and Effect can be entered and viewed in Enterprise Architect.



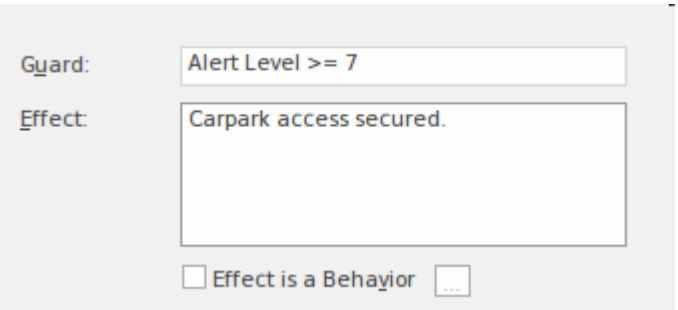
This diagram illustrates the way the Trigger, Guard and Effect are displayed on a diagram. The Effect in this case has been defined as a behavior and has been linked to an Operation defined on the Block.



Guards

Guards are the 'gate keepers' of a transition and it is only when the guard's expression evaluates to True that the transition will fire. If the expression evaluates to false the event will be consumed and there will be no observable change in the Block's state resulting from the trigger.

A guard's expression can be defined in plain English, but typically it is written in the form of a constraint using a formal constraint language such as the Object Constraint Language. When working with simulations or Executable StateMachines the condition is expressed in the syntax of the code language that it is to be generated in, for example JavaScript or C++. This also applies to Effects. In this diagram we can see a mathematical expression that can be evaluated by a human or a machine.

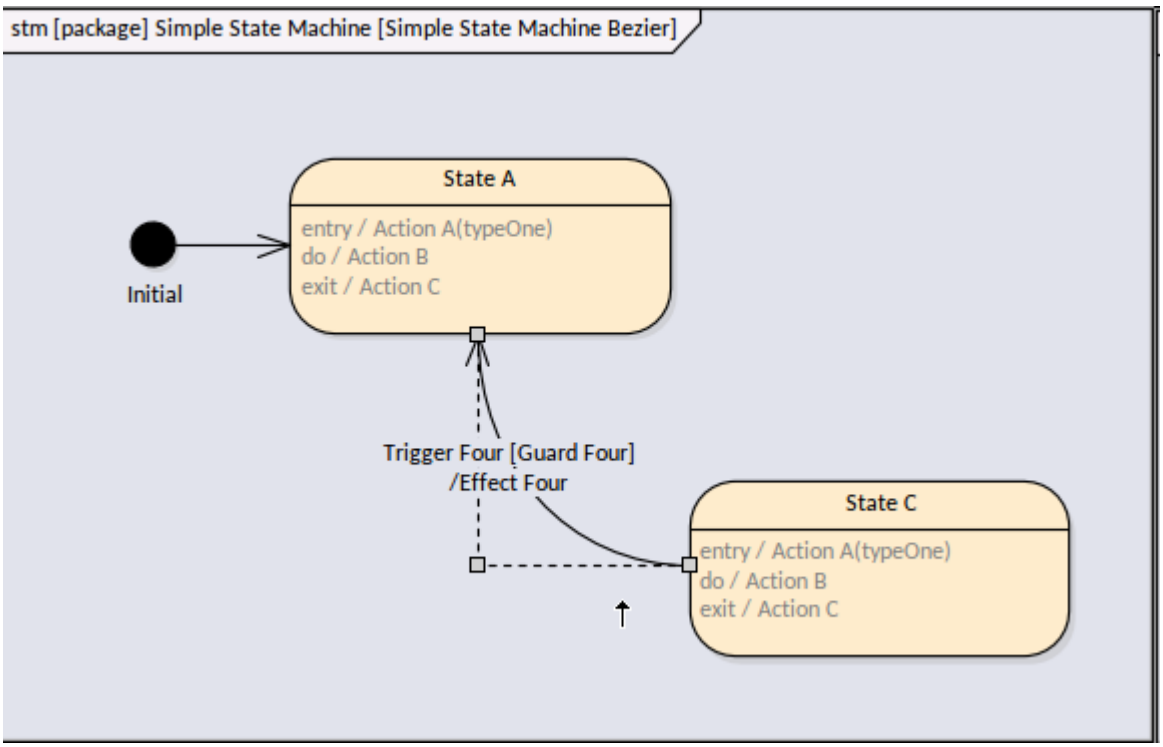


Effects

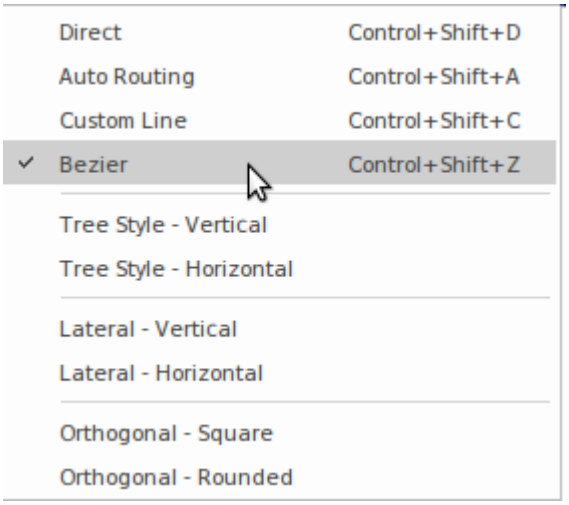
Apart from moving the owning Block from one state to another, the significance of the Transition relationship is that it can execute a behavior that could be an Activity or an Operation on the Owning Block or on any other Block. This behavior is in addition to the Exit behavior that might have been defined on the source State, and the Entry behavior on the target state. This ensures that there is a mechanism to change the behavior during the execution of the Transition.

Bezier Curves

Enterprise Architect has a wide range of tools and facilities for working with diagrams, including StateMachine diagrams, and these can be used to help create and visualize the information codified in the StateMachines. Particularly useful when working with Transitions is the ability to route connectors that help make the diagram more appealing. The line style for the connector in this diagram has been set to a *Bezier Curve,* giving the Transition a less rigid appearance.



The shape of the curve can be altered by dragging the construction point to a new location. Any one of a number of line styles can be used, providing the modeler with a toolkit of options for diagram presentation. This context menu can be selected and the line style set for each connector individually. The color and thickness of the line can also be set from the Layout ribbon.


Direct Control+Shift+D
Auto Routing Control+Shift+A
Custom Line Control+Shift+C
Bezier Control+Shift+Z
Tree Style - Vertical
Tree Style - Horizontal
Lateral - Vertical
Lateral - Horizontal
Orthogonal - Square
Orthogonal - Rounded

Composite States and Regions

The modeling of states has to be hierarchical to deal with the complexity of engineering systems; the SysML provides two mechanism for modeling this hierarchy in a StateMachine:

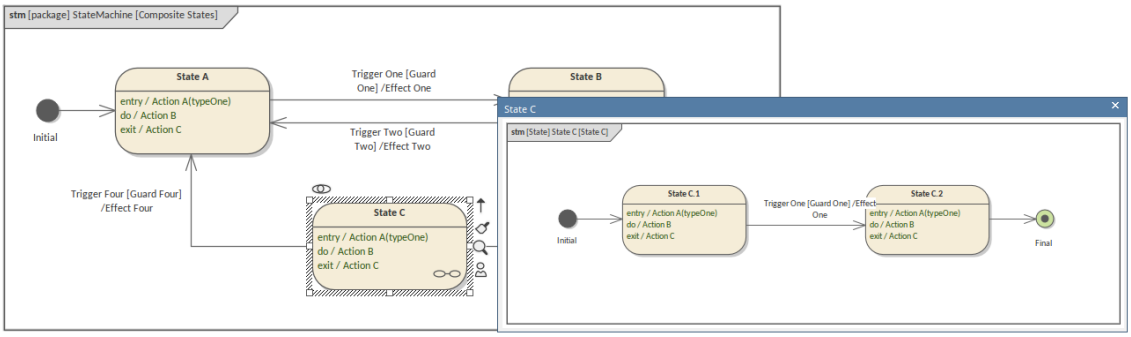
- **Regions** - which are separate parts of a StateMachine
- **Composite States** - which are States that contain other States

Systems and the objects they contain often exhibit concurrent behavior where two things can be occur at the same time; often these separate behaviors interact with each other to create complex state-based behavior. These behaviors are typically represented in the Block that the StateMachine is describing and might involve parts that have differing lifetimes. These situations can be modeled using regions; a StateMachine can contain any number of regions, each with its own set of States, Pseudostates and the transitions that connect them.

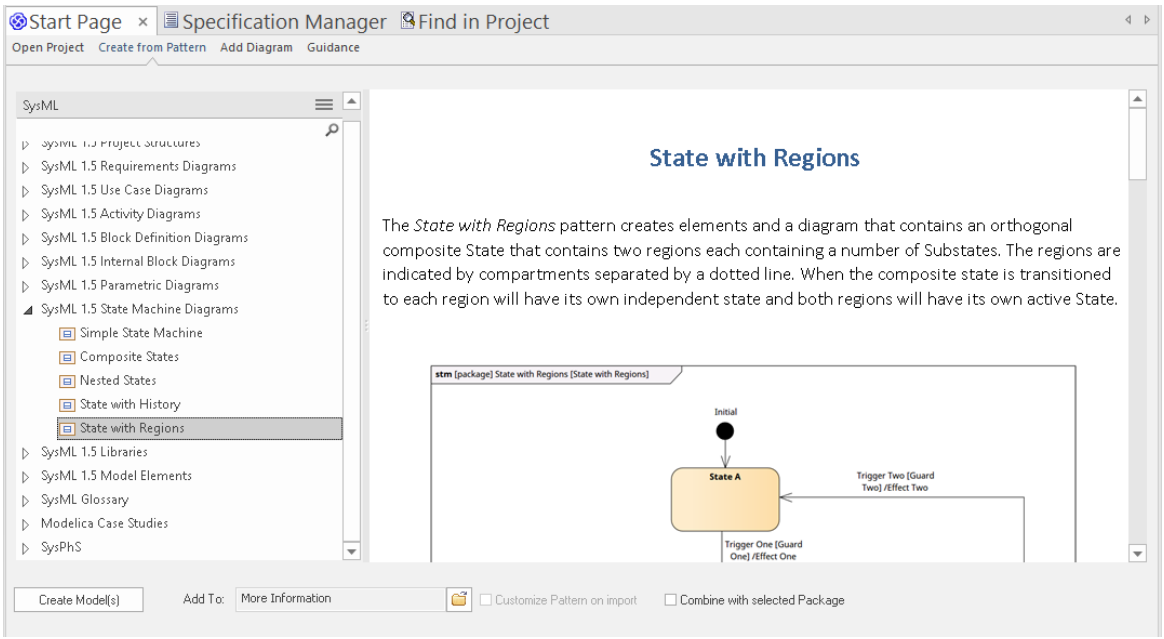
A State itself might need to be hierarchical where a single State can be decomposed into a number of sub-States representing the states that form part of the composite State. For example a Robot might have Operating and Maintenance States; the Maintenance State could be decomposed into a number of sub-states such as Recharging Battery, Updating Environment and Updating Software Modules. Each of these States could in turn be decomposed into a number of other States.

Enterprise Architect provides a useful starting point for modeling complex state behavior, by providing a series of model patterns that can be used to model all aspects of StateMachines, including modeling Composite States and Regions.

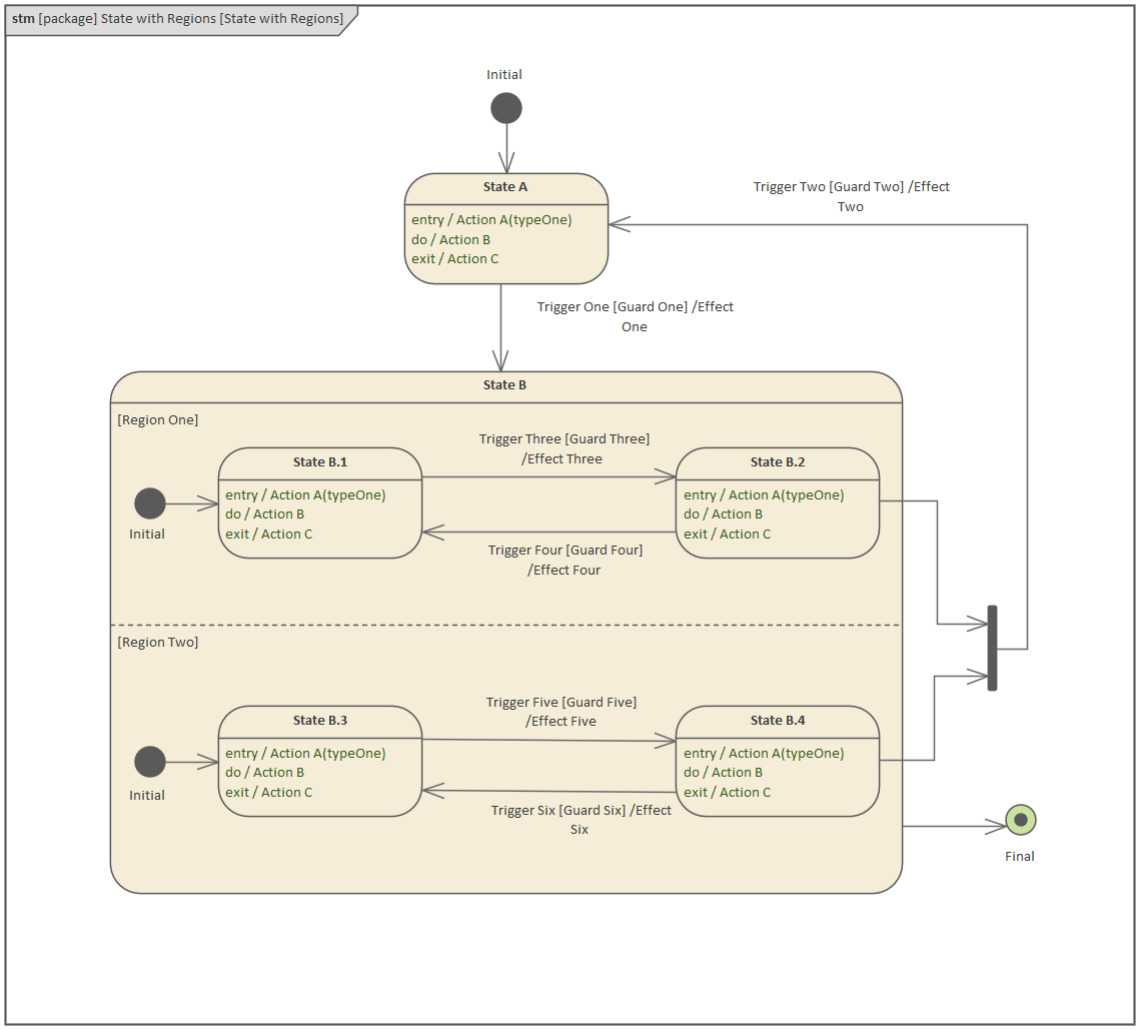
Composite States are indicated by the oo ('infinity') symbol and reference a child diagram. The child diagram can be previewed by clicking on the  icon. Double-clicking on the element or on the diagram preview will open the diagram for editing.



These patterns can be accessed using the Model Builder.



This image shows a diagram that has been created from the *State with Regions* pattern, which has been simply injected into the model and acts as a starting point for the modeler. The modeler can tailor the diagram by replacing the names of States and Transitions and adding or deleting regions as required.



The regions in State B provide a parallel flow of processes in Region One and Region Two.

Pseudostates - The Traffic Police

The Transition relationships that connect States in a StateMachine diagram need to be orchestrated to ensure that the StateMachine is representative of the freedoms that a Block has to change its state in the physical world. Pseudostates are nodes that are used to direct the flow along transitions. The nodes can appear at the beginning, along the path of a transition or at the end. Types of pseudostate include:

- *Initial* - used to initiate a StateMachine
- *Fork and Join* - used to split and reunite a Transition
- *Terminate* - used to end a StateMachine
- *Join* - used to reunite a number of Transitions
- *Junction* - used to split a Transition
- *Entry and Exit Point* - used on the boundary of a SubMachine State
- *Deep and Shallow History* - used as mementoes when a composite State is exited

We will look at each in a little more detail and show how Enterprise Architect can be used to create and manage these important nodes. it is important to understand that the Final State - which has an analogous icon to the Initial pseudostate - is in fact a State in its own right.

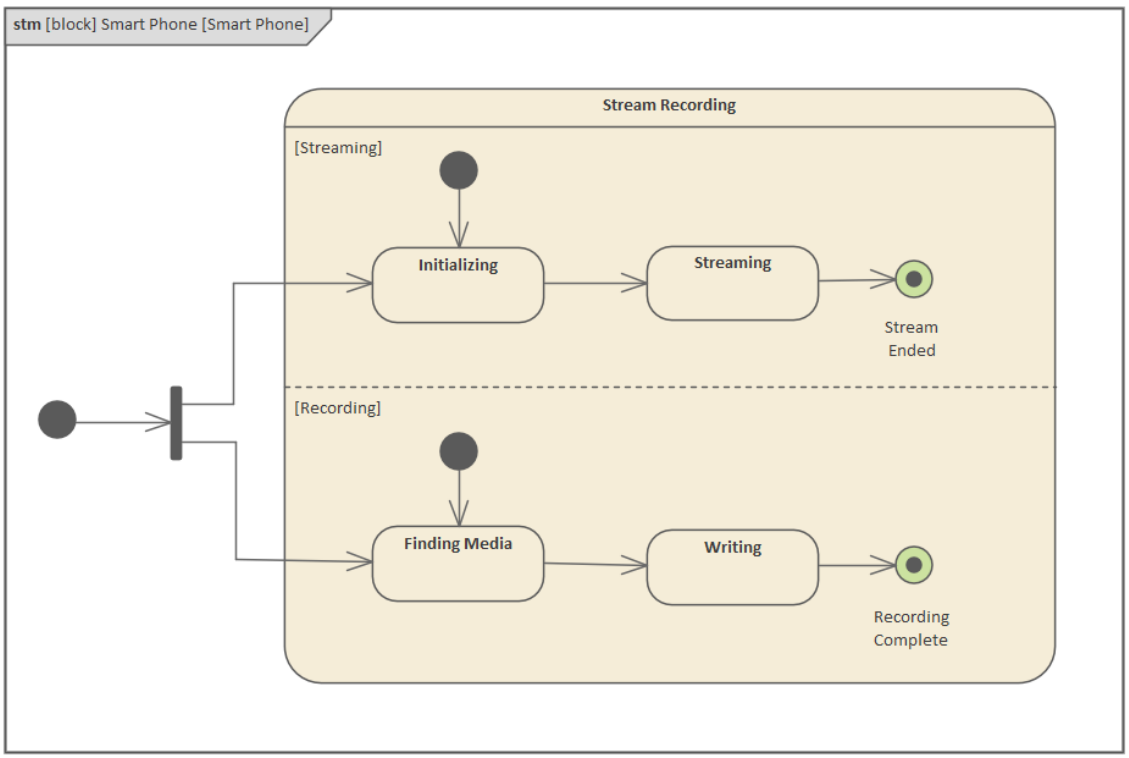
Initial

The Initial pseudostate is the most widely used of all the nodes and represents the stating point for a region. There can only be one *Initial* in a region and a single transition is permitted to emerge from the Initial pseudostate. Because it is the starting point it would not make sense to have a trigger or a guard - the pseudostate simply becomes active when the region is entered, a modeler can however define an effect. Some system engineers will leave this all-important pseudostate off diagrams, reasoning that its position is implied, but it is considered good practice to include them as formally the starting point is *undefined* without them. It is, however, common practice to leave the Initial pseudostate unnamed.

Enterprise Architect will also rely on the StateMachines being well formed, and that each region has an initial pseudostate defined, when a modeler is working with Executable StateMachines or running simulations to visualize the States a Block instance will transition through in its lifetime.

Fork and Join

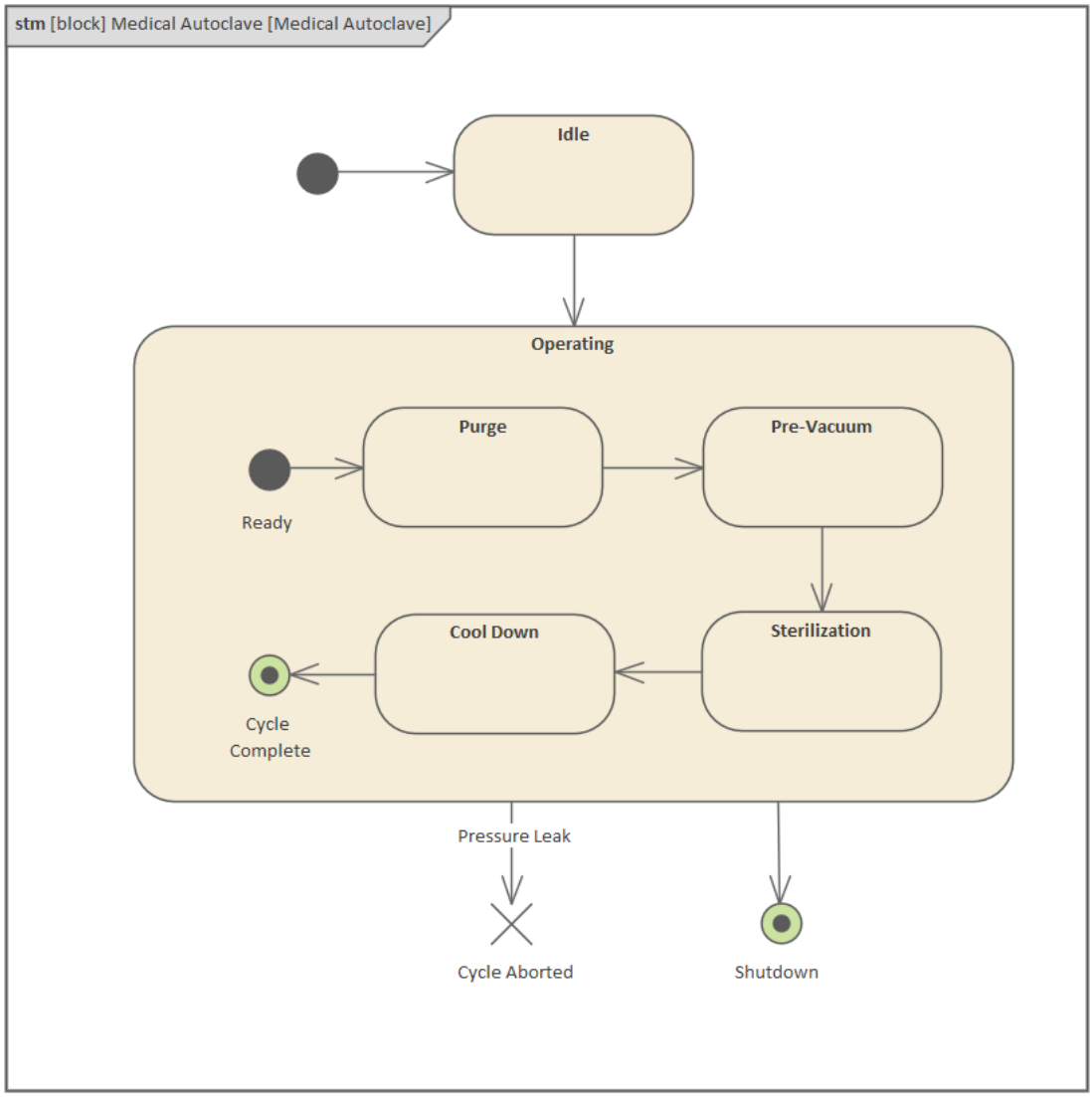
When Regions are used in StateMachine diagrams, it often necessary to split a transition that targets the State into multiple transitions such that each outgoing transition targets a given State in each orthogonal Region. The *outgoing* transitions from a Fork pseudostate are restricted and cannot have a guard or a trigger defined but an Effect can be defined. This ensures that multiple regions can simultaneously have active states.



Joins work in an analogous but opposite way to unite incoming transitions from multiple regions. The reverse restriction applies such that the *incoming* transitions cannot have a guard or a trigger defined but an Effect can be defined. The Effects for all incoming transitions must be completed before the outgoing transition can fire.

Terminate

The Terminate pseudostate is a useful node to ensure that an entire StateMachine is shut down. Regardless of what level in a State hierarchy the node is located, all regions and all levels of the hierarchy will terminate. It is a final node in the sense that the owning StateMachine will immediately stop executing. The termination is not 'graceful' and any behaviors that are currently being executed will simply stop; no exit behaviors will be executed. This diagram shows a Composite State with a single region; if there is a *Pressure Leak* the operation of the Autoclave must be immediately terminated, so there is a transition from the operating state to the Cycle Aborted Terminate pseudostate.



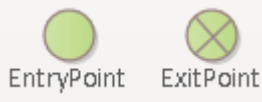
Enterprise Architect allows this node to be placed at any level and in any region and will honor its semantics in Executable StateMachine simulations.

Junction

The junction pseudostate is used to model transitions with compound paths There are both inbound and outbound transitions that are connected to a junction, but during the execution of the StateMachine only one of the inbound and one of the outbound transitions will fire. The outbound transitions are protected by guards and only the transition with a guard whose expression first evaluates to Boolean true will fire and carry the outgoing token.

Entry and Exit Point

The Entry point and Exit point pseudostates are used to allow a StateMachine to be reused as a sub-machine State in multiple contexts; they can also be used on a composite State. These pseudostates appear as small circles that straddle the boundary of a composite State or a sub-machine State; the entry point is empty whereas the exit point has a small x inside the circle.

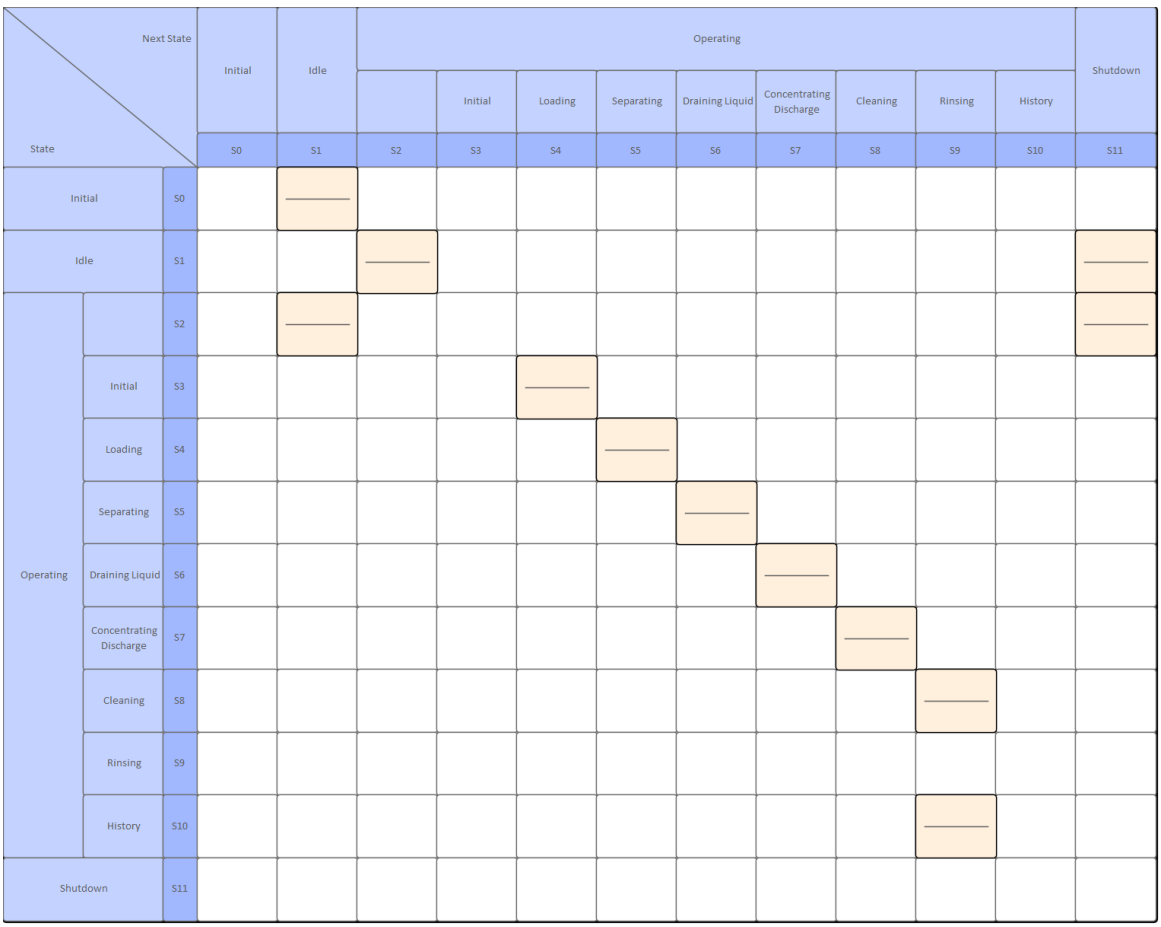


Their position on the boundary is significant because, from a visual syntax point of view, they allow messages between the inside and outside of the element to be conveyed.

Deep and Shallow History

The History pseudostate is like a bookmark or memento for a Composite State, and simply stores the name of the sub-state that was active when the region was exited. When the region is subsequently re-entered, the StateMachine resumes its transitions from the sub-state specified by the History. It is possible that for some reason a region could be re-entered and the History is unable to provide the last State; this situation can be handled by a modeler pre-emptively creating a Transition from the History pseudostate to a target default sub-State; the Transition would only be used in the event that the History was unable to provide the memento.

In this example of a Tubular Centrifuge, a System Engineer has placed a Shallow History State in the operating Composite State, indicating that if that start is exited while a given State is active, and then the State is subsequently re-entered, the execution will resume at the active sub-state. The transition exiting the History indicates, in the event that the owning State is re-entered and the machine did not know which State to make active, *Rinsing* would be selected.



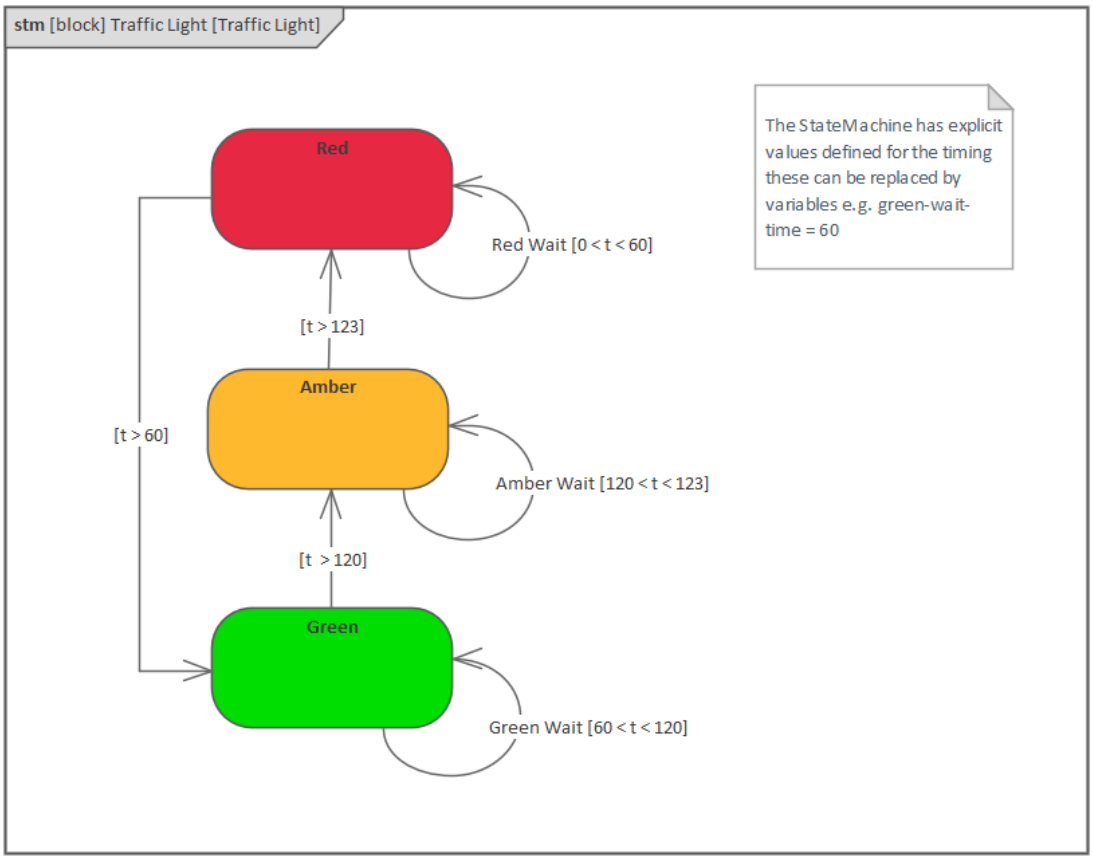
Both Shallow and Deep History States work the same way except that a Shallow History pseudostate only remembers the active sub-states in the owning Region, a Deep History can remember down to any level in a sub-state hierarchy. The Deep History is indicated visually by an asterisk placed after the H*.

State Tables - Another View

A StateMachine can be visualized in a number of different ways. We have already looked at a diagram view of the StateMachine, but Enterprise Architect has a useful tool that allows the StateMachine to be visualized in a table. There are three variants of the table visualization:

- State Next State View - where the States and Sub-States are organized as rows and columns and the cells represent the Transitions
- State Trigger View - where the triggers are organized as columns and the states as rows and the cells represent the Transitions
- Trigger State View - where the triggers are organized as rows and the triggers as columns and the cells represent the Transitions.

This diagram of the changes of a traffic light, like any StateMachine diagram, can be converted to a State Table.



This view will be appealing and more natural to some audiences, and the engineer can simply toggle from one view to another. The States and their Sub-States are represented on both the rows and the columns of the table, and the transitions (representing the pathways between States) are represented in the cells.

Next State / State		Red S0	Amber S1	Green S2
Red	S0	Red ... 0 <...		t > 60
Amber	S1	t > 123	Amb... 12...	
Green	S2		t > 120	Gree... 60 ...

The two other table views, as outlined previously, allow the States to be viewed against the Triggers that initiate them. These are helpful views when the engineer is more interested in the causal analysis and wants to view or analyze how events and triggers result in State behavior of the owning Block. This diagram shows the same Traffic Light machine represented as a table of triggers and States.

State / Trigger		Red S0	Amber S1	Green S2
Green Wait	E0			[60 < t < 120] S2
Amber Wait	E1		[120 < t < 123] S1	
Red Wait	E2	[0 < t < 60] S0		
<None>	E3	[t > 60] S2	[t > 123] S0	[t > 120] S1

Enterprise Architect also allows these tables to be exported so that they can be analyzed using a spreadsheet. This is a useful mechanism, particularly when the StateMachine is complex and there are large numbers of transitions.

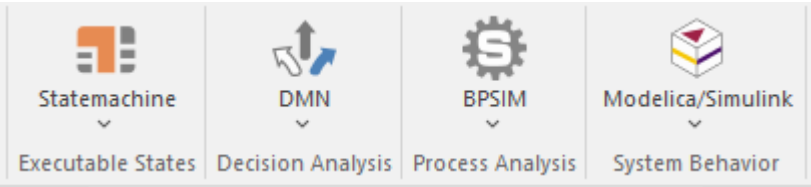

State
Trigger
Red
Amber
Green
S0
S1
S2
Green Wait
E0
[60 < t < 120]
S2
Amber Wait
E1
[120 < t < 123]
S1
Red Wait
E2
[0 < t < 60]
S0
<None>
E3
[t > 60]
S2
[t > 123]
S0
[t > 120]
S1
State Table Options...
Properties...
Lock Diagram
Save Current Changes
Control+S
Statechart Editor
Export Statechart to CSV file...
Execute Simulation
Help...

Visualizing and Implementing with Simulations

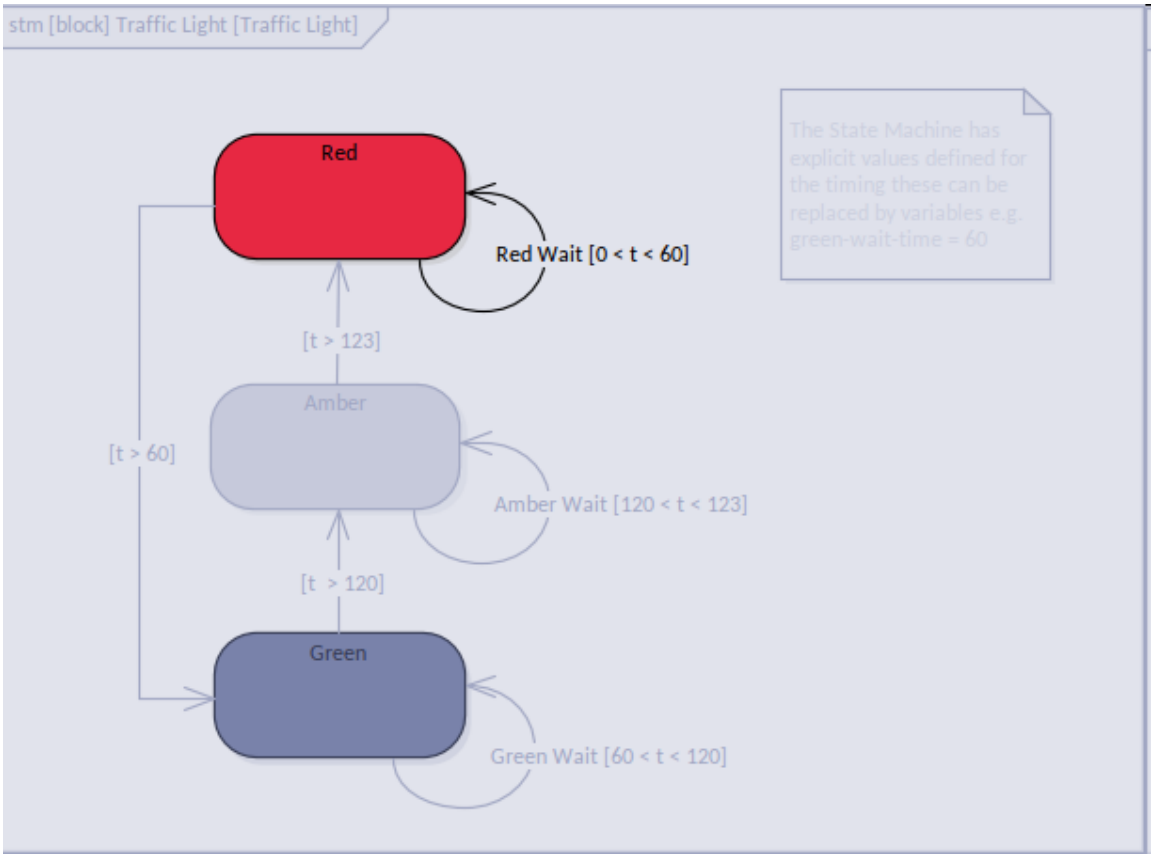
Enterprise Architect is not only a platform for creating and managing StateMachine diagrams, it also provides sophisticated Simulation facilities for engineers and other stakeholders to visualize the StateMachines. This brings the models to life and provides a visualization tool, not only for the Engineer who is developing the models, but also for the other audiences, both technical and non-technical, who need to understand what the model is saying. It is somewhat like an author reading a newly written passage of text out aloud, and it can help the engineer find errors in the models or aspects of the models that should be corrected or reworked. It is particularly useful as the models become more complex, with nested sub-states, complex Triggers and Guards, and pseudostates such as Forks and Joins that split and reunite transitions.

With extensive support for Triggers, Trigger Sets, nested States, concurrent States, dynamic effects and other advanced simulation capabilities, the feature provides a sophisticated environment in which to build interactive and working models that help explore, test and visually trace complex business, software and system behavior. There is a ribbon dedicated to simulation, which provides a range of items that can be used for both dynamic and executable simulations of StateMachines. This image shows the core tools for working with dynamic simulations.

The second image shows the other advanced tools, including the Executable StateMachine, that can be used to create executions of the StateMachine to produce fully implementable and compilable programming code directly from the simulated StateMachines. The image also shows a number of other facilities, including Modelica and Simulink, which are sophisticated tools for running complex parametric simulations.

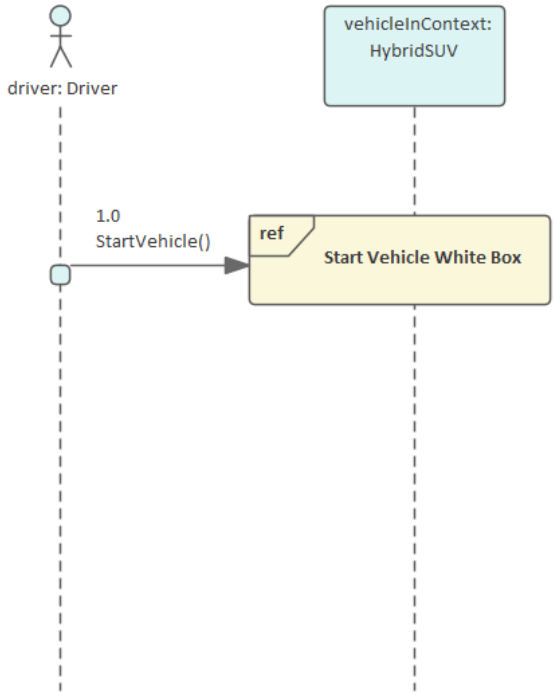


This diagram shows a simple dynamic simulation of the traffic light system that we looked at in the previous exercise. It shows the dynamic simulation in action.


stm [block] Traffic Light [Traffic Light]
The State Machine has explicit values defined for the timing these can be replaced by variables e.g. green-wait-time = 60
Red
Red Wait [0 < t < 60]
[t > 123]
Amber
[t > 60]
Amber Wait [120 < t < 123]
[t > 120]
Green
Green Wait [60 < t < 120]

Interactions as a Sequence of Messages

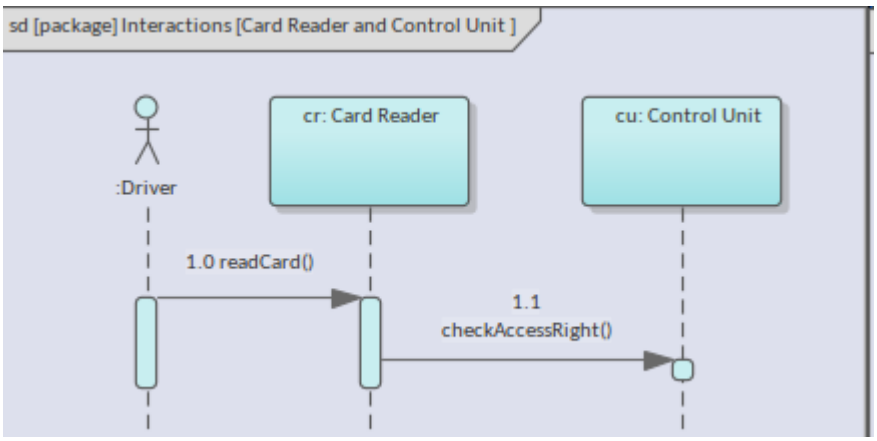
Systems are made up of parts and the overall behavior of a system is provided by these parts working together in an orchestrated way. Communication between the parts and the synchronization of their behaviors is critical, both from a design perspective and from a visualization perspective. The structural units of a system, represented primarily by the Blocks, exchange messages and signals that trigger behaviors, resulting in coordinated system behavior that represents the system's functions.



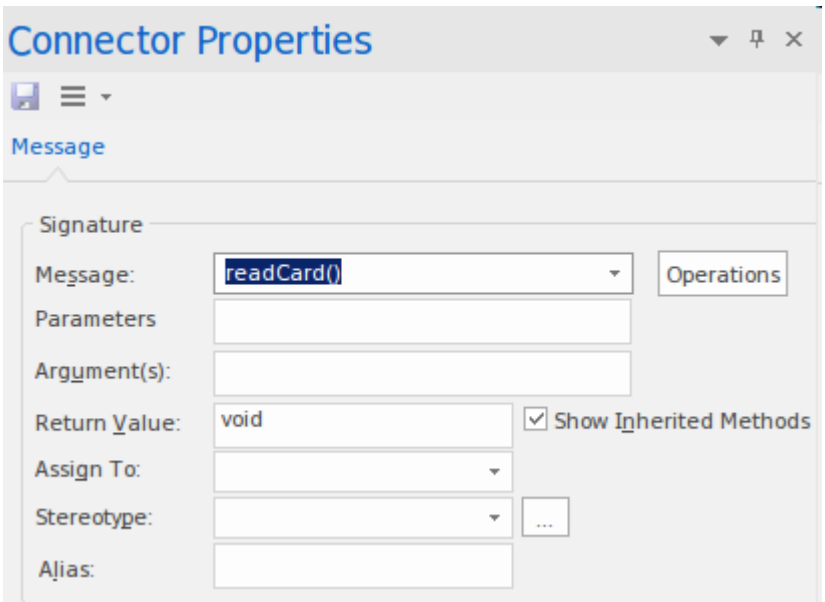
This exchange of messages and signals, and the consequent behaviors, can be represented on a Sequence diagram that shows the time-sequenced messages and signals between Block instances that participate in a specific interaction.

Lifelines, Messages and Activations

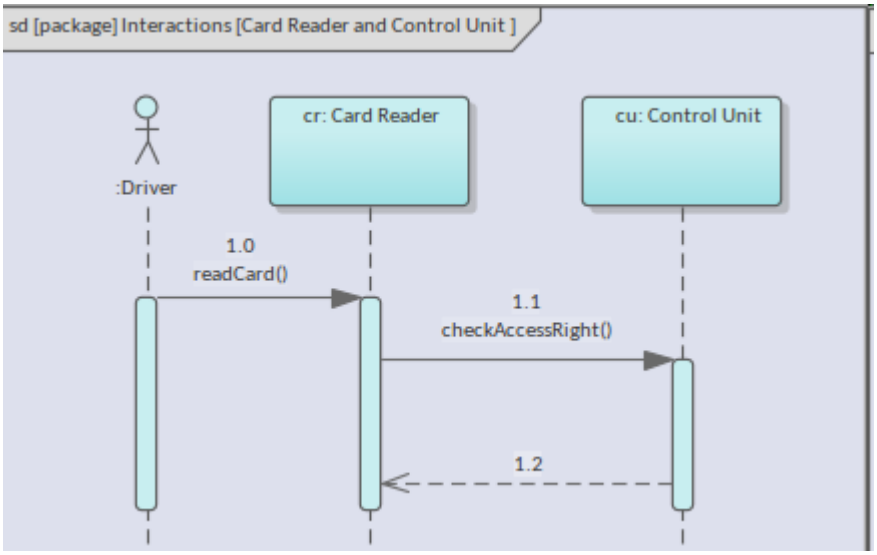
In a Sequence diagram, the Blocks that participate in the interaction have a lifetime that is represented by a dashed line, emanating from the base of the element and continuing vertically for the life of the element. Elements can be created or destroyed at any time during the period represented by the Sequence diagram, and the lifeline therefore represents their existence. Elements that are present at the top of the diagram are created at the beginning of the interaction. A message exchange between a sender and a receiver will originate in one lifeline (the sender) and end in another (the receiver).



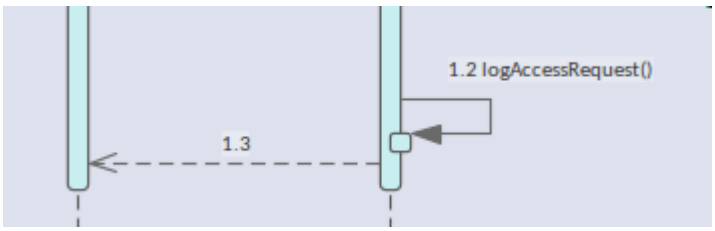
The sender is effectively calling or activating some behavior provided by the receiving lifeline. Enterprise Architect provides useful mechanisms to utilize behaviors already defined in the form of operations. This illustration shows the properties of an existing message, which the modeler is able to select from a drop down list of previously defined operations. Alternatively, the modeler can define a new operation by selecting the Operations button.



Formally, when a message targets a lifeline an execution occurs, meaning that a behavior is initiated or augmented. This execution is represented visually by an activation, which is drawn on the diagram as a thin rectangular overlay on the lifeline, the length of which represents the relative duration of the behavior. The extent of the rectangle activation is ended when a reply message is sent back to the caller.

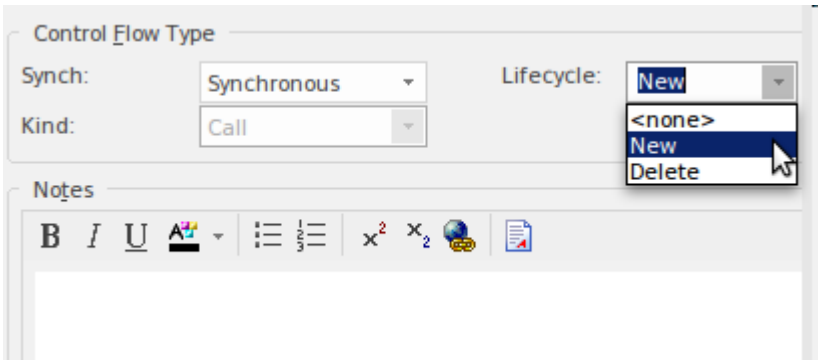


A Block (Instance) can be both the sender and the receiver of a message, which can be referred to as a reflexive message because it starts and ends on the same lifeline. In this case a second and shorter activation rectangle is overlaid on the first activation but offset to the right.

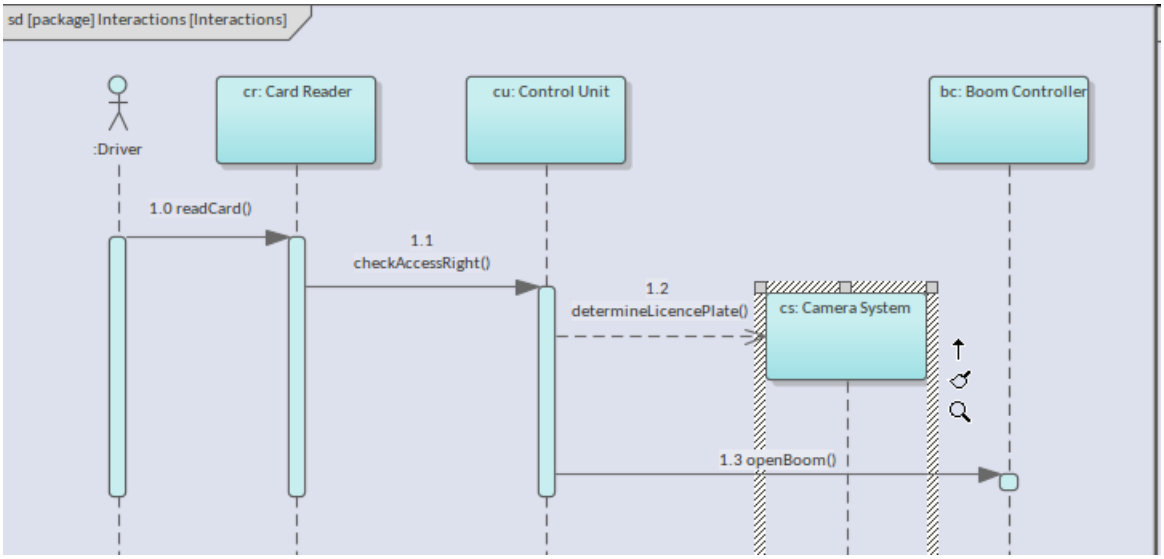


Messages that Create and Destroy Blocks

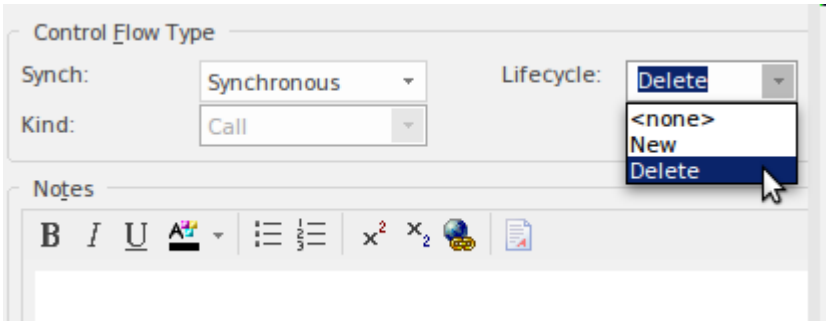
Any number of Block Instances can form part of an interaction, and often a set of these instances will be present for the duration of the time represented by the Sequence diagram. These Blocks will be positioned in a row across the top of the diagram. It is, however, possible for Blocks to be created and destroyed during the sequence of the diagram. For example, a particular Block might only be needed for a short time and so could be instantiated, perform its function and then be destroyed. Enterprise Architect allows an Engineer to specify that a message is a *Create* message, which means that the receiving Block will be instantiated at that point in the diagram's time sequence. This is achieved by setting the *Life Cycle* message action to 'New' as indicated in this illustration.



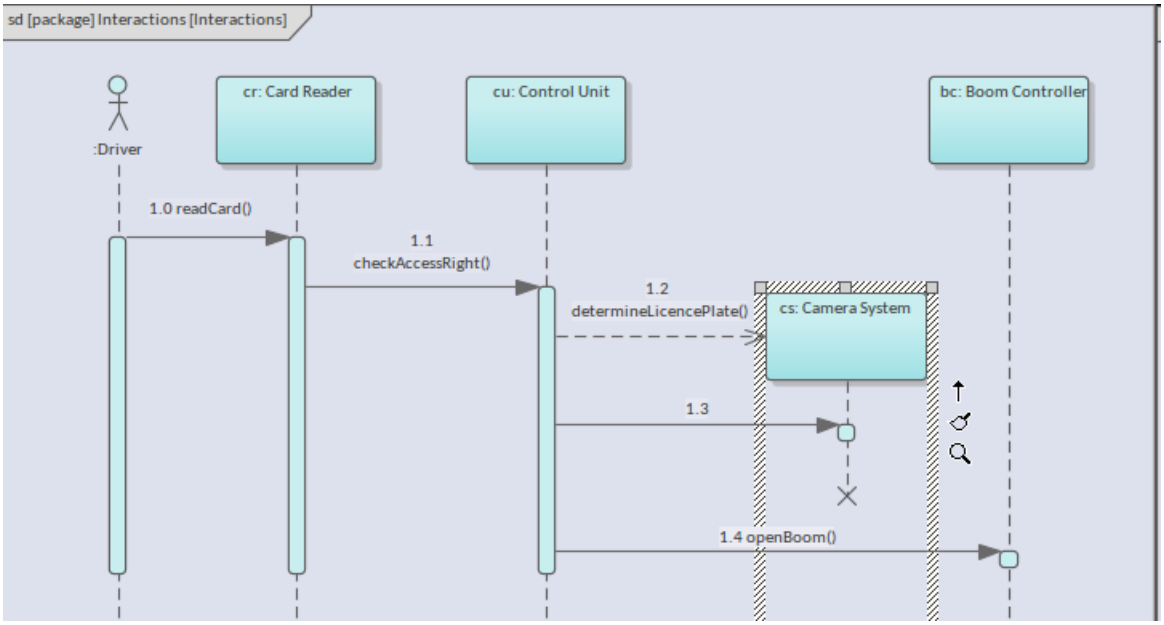
Setting this property has the effect of moving the targeted Block to a position lower in the diagram, signifying that the Block Instance would not be created until this point in the overall time sequence of the diagram. The message line style is also altered to a dashed line with an open arrow-head to reflect that it is a *Create* message. Other aspects of the semantics and effect of this message are unchanged.



In an analogous way the lifetime of an Instance can be ended by sending a *Destroy* message. Practically, this means that the Instance has served its purpose and is no longer required. This can be achieved easily by once again setting a message property but this time we set the *Life Cycle* message action to 'Delete'.



Setting this property has the effect of immediately terminating the lifetime of the Instance; this is represented visually by the dashed lifeline being arrested by a small cross on the lifeline at the point in time that the message arrives.

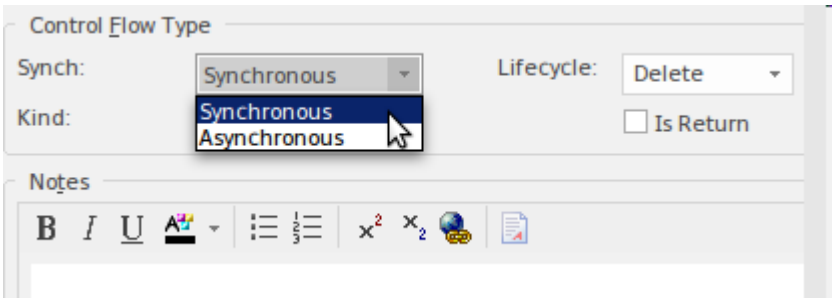


When a Sequence diagram is representing a software system that has bounded memory available, the destruction of the targeted instance will result in allocated memory being returned to the memory pool. In systems engineering there can be a variety of other pay-offs from managing the lifetime of electro-mechanical objects, such as power consumption, over-heating, availability, or even risks such as security.

Synchonous and Asynchronous Messages

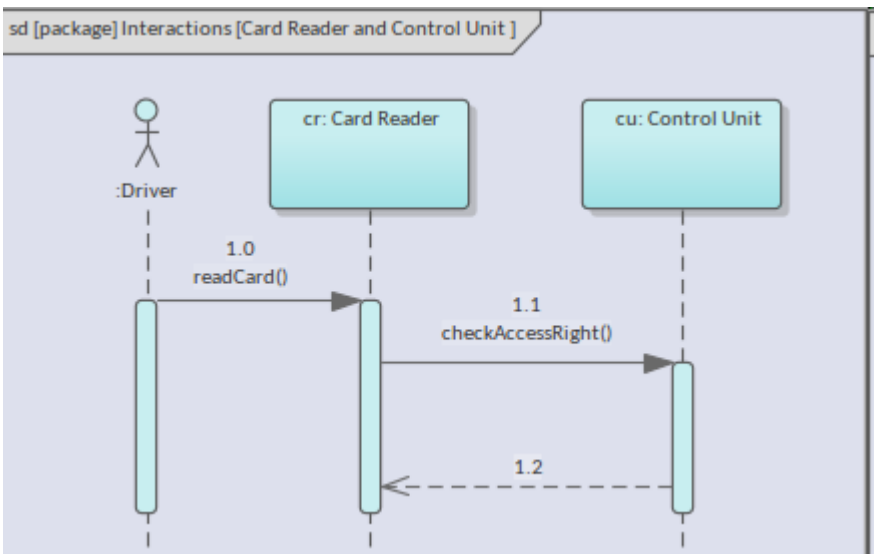
Messages essentially represent sending some type of request from a sender to a receiver. There are two fundamental ways the sender of a message can interact with the recipient. The first type of message is called a *Synchronous* message because the messages occur at approximately the same point in time. With this type of message the sender waits until the recipient replies before sending additional messages. The second type of message is called *Asynchronous* because the sender does not wait for a reply from the recipient before continuing on with the execution, including sending additional messages to this or other recipients.

Enterprise Architect by default creates *Synchronous messages,* but the message type can be altered by setting the 'Synch type' property on the 'Properties' dialog. When the message type is set to Synchronous (default) this sets the line style to solid with a closed arrow head pointing to the recipient Instance. The line can be annotated with the name and parameters of the message.



Asynchronous messages represent calls to operations, or *signals* that are sent to recipients; in either case the sender does not expect a reply nor pause its execution in wait of one. In the case of the call to an operation, the operation itself would be defined as asynchronous and the system or machine represented by the diagram would know not to wait for a message return (reply). Enterprise Architect allows this message type to be set through the 'Synch' property as explained previously. When the message type is set to Asynchronous this sets the line style to dashed with an open arrow head pointing to the recipient Instance. The line can be annotated with the name and parameters of the message.

There is a third type of message that can optionally be used with a Synchronous message, this being the *Reply Message*. This message signifies an operation that has been invoked on the recipient returning a receipt that the behavior has been executed and is complete. The inclusion of Reply messages in a diagram is a stylistic decision. Some engineers and modelers prefer to leave them off diagrams to reduce the visual clutter. If a return type and value have been set, this will be returned as part of the Reply message.



Blocks can have both *Operations* and *Receptions* specified as part of their definition; this paradigm describes the

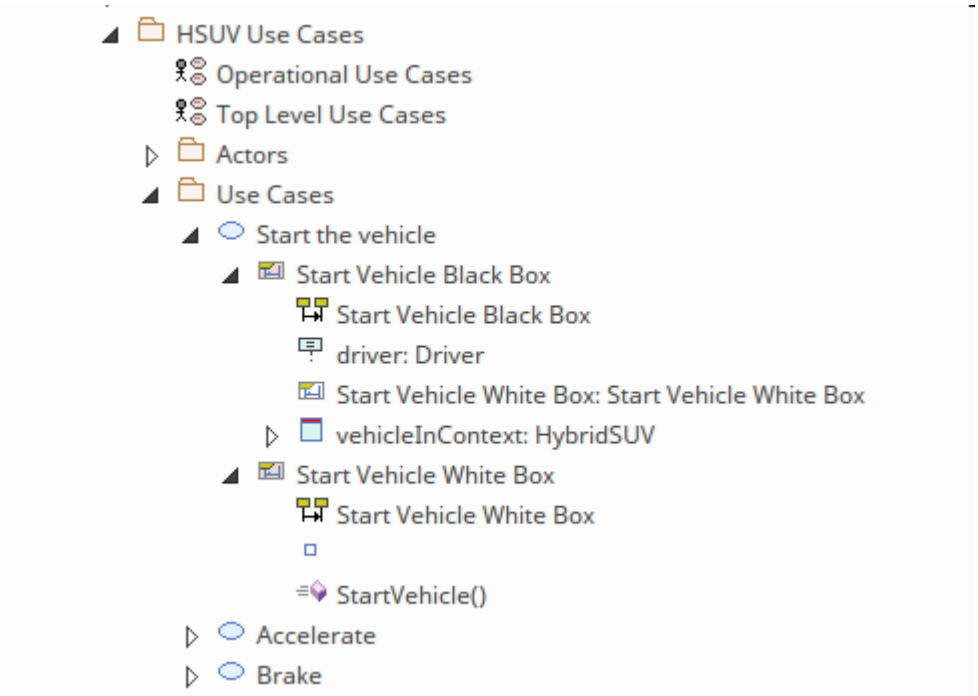
relationship of Operations and Receptions to messages and signals.

- Synchronous Invocation of Operation - Synchronous Message
- Asynchronous Invocation of Operation - Asynchronous Message
- Reception Receipt of Signal - Asynchronous Message

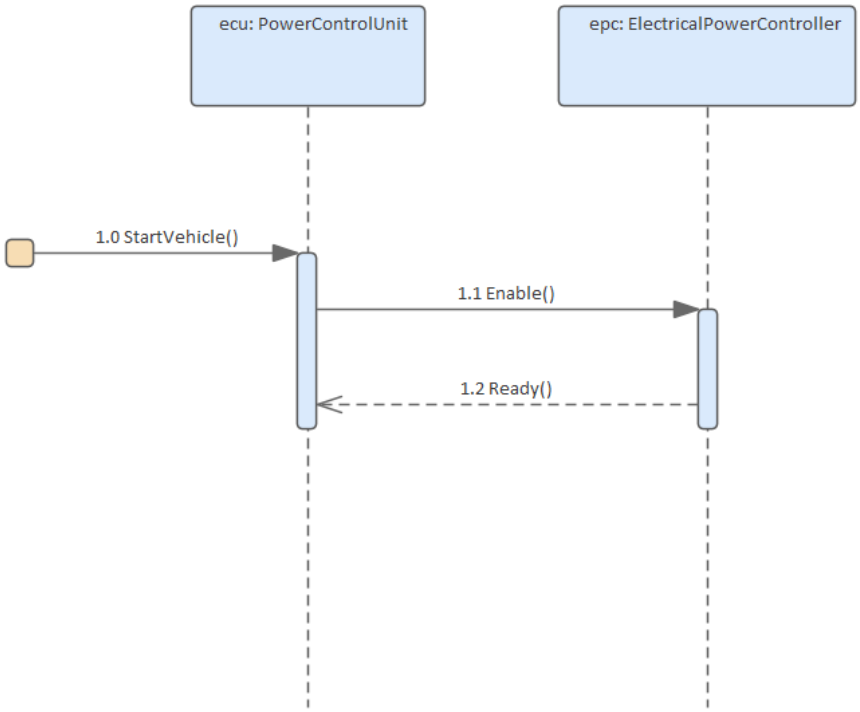
Introducing the Sequence Diagram

The Sequence diagram has its origins in the Unified Modeling Language, and in that language has primarily been used to represent interactions between components in software centric systems. Its usage has been broadened in the context of Systems Engineering, where it is used in a more generic way to represent the time-sequenced exchange of messages and signals between structural units of a system or part of a system.

The Sequence diagram has two axes; by convention the horizontal (x) axis represents the Block (Instances) that participate in the interaction and the vertical (y) axis represents time. The Blocks do not have to be ordered in any prescribed way, but a modeler will typically place them in the order that is most illustrative and that order is often based on when they are used in the interaction. Time does not run on a linear scale and the time scale between any two diagrams could be quite different. For example, the time scale on Sequence diagram representing a high speed photographic system would be very different to the scale on a diagram representing a grocery checkout machine. This diagram shows the location of two Sequence diagrams ('Start Vehicle Black Box' and 'Start Vehicle White Box') that are child nodes of a Use Case named 'Start the Vehicle'.



The tree structure acts as a navigation aid, and by double-clicking the item in the Browser window you would open the diagram from this view.



In the second diagram we see a simple Sequence diagram that represents the sequence of messages involved in starting a vehicle. It can be seen from the diagram that there are two Blocks (Instances) that form part of the interaction, and messages are exchanged between the two Blocks and the initiator of the interaction, and ultimately the Use Case.

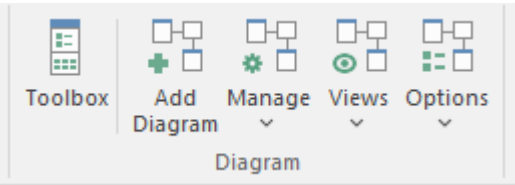
Creating Sequence Diagrams

A Sequence diagram can be created from a number of places in the User Interface by using:

- Design ribbon - *Add Diagram Icon on the Diagram Panel*
- Browser window Toolbar - *New Diagram Icon*
- Browser window *Context Menu - New Diagram*

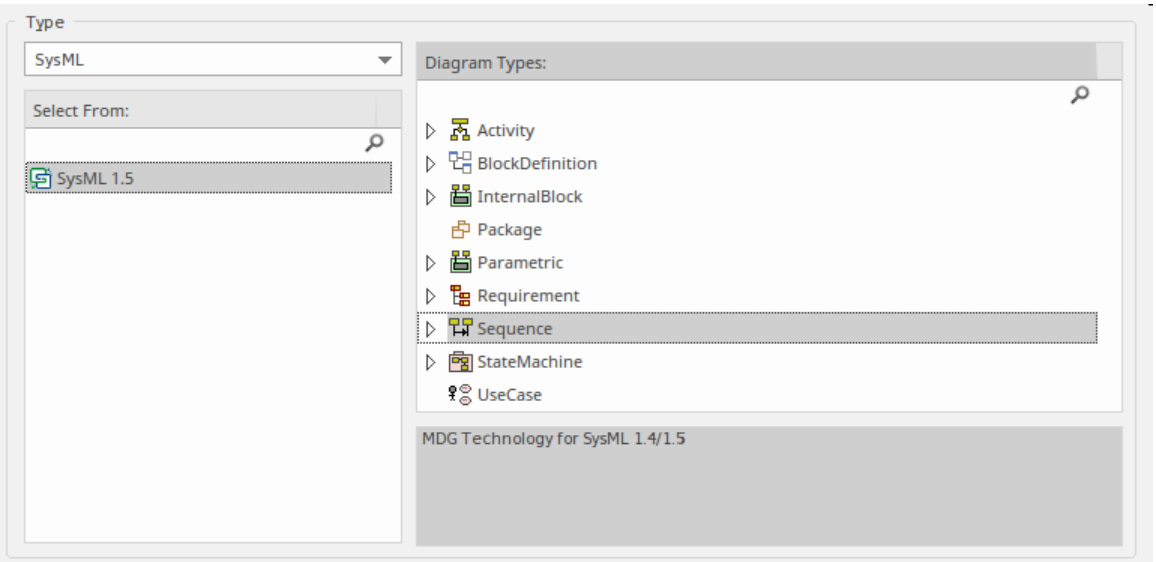
We will use the Design ribbon to create a Sequence diagram. Firstly you will need to select the location in the Browser window where you want the Sequence diagram to be created. As with all diagrams, this can be either a Package or an element, but it is common to add Sequence diagrams to a Package as it typically involves a number of objects in the Package. Once the Package location has been selected in the Browser window, select:

Ribbon: Design > Diagram > Add



Selecting this option will open the 'Diagram Builder' tab page of the Model Builder dialog, where you can choose the diagram type and specify the name of the diagram; initially the name will default to the name of the Package or element that contains the diagram. With the SysML perspective chosen and the version of SysML selected a list of diagram types will be displayed, allowing the selection of the Sequence diagram. Once the *Create Diagram* button is selected a new Sequence diagram will be created in the location selected in the Browser window. The diagram canvas will be opened allowing you to start adding elements and connectors that describe the important interactions between objects. Enterprise Architect will also display the Sequence diagram Toolbox pages that contain the elements and relationships defined by the SysML specification as applicable for constructing Sequence diagrams. Any number of other Toolbox pages can be

opened if required, in addition to the Common Elements and Common Relationships pages that will always be available.



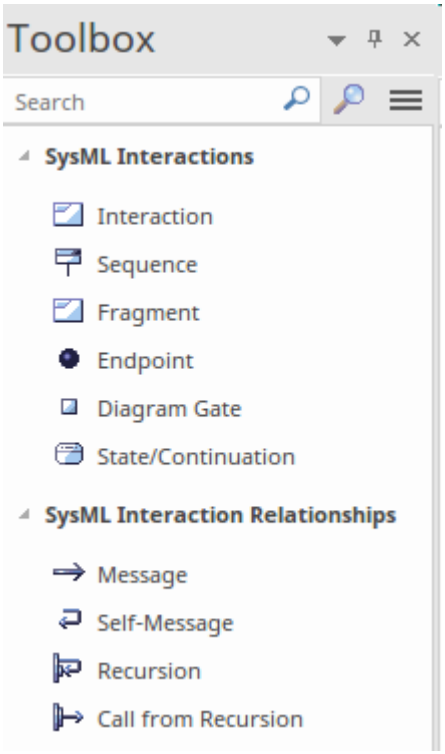
The most import elements and connectors used with the Sequence diagram are:

Elements

- Interaction
- Sequence
- Fragment
- Endpoint
- Diagram Gate
- State/Continuation

Connectors

- Message
- Self Message
- Recursion
- Call from Recursion

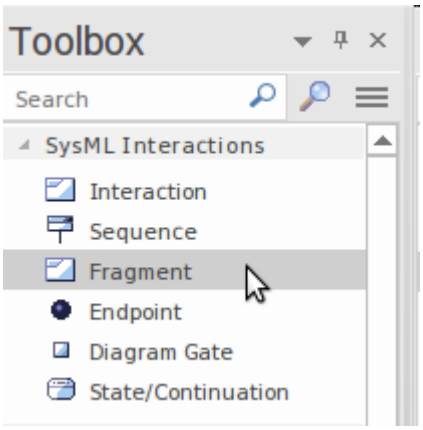


Elements can be added to the diagram by dragging-and-dropping them from the Browser or from the Toolbox onto the diagram canvas. The typical process is to reuse existing elements such as Blocks, which have behaviors in the form of operations that can be selected as the basis for the messages that are exchanged between lifelines. The elements can be added to the diagram as a link but more typically they are added as a lifeline.

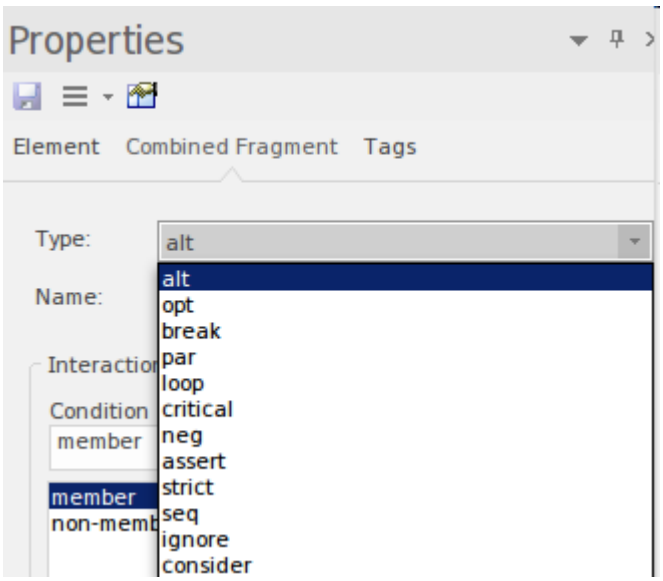
Once a basic diagram has been created, and as knowledge of the domain and the system's interactions are further revealed, it is possible to add Fragments, Endpoints, Diagram Gates and State/Continuation elements.

Message Orchestration with Fragments

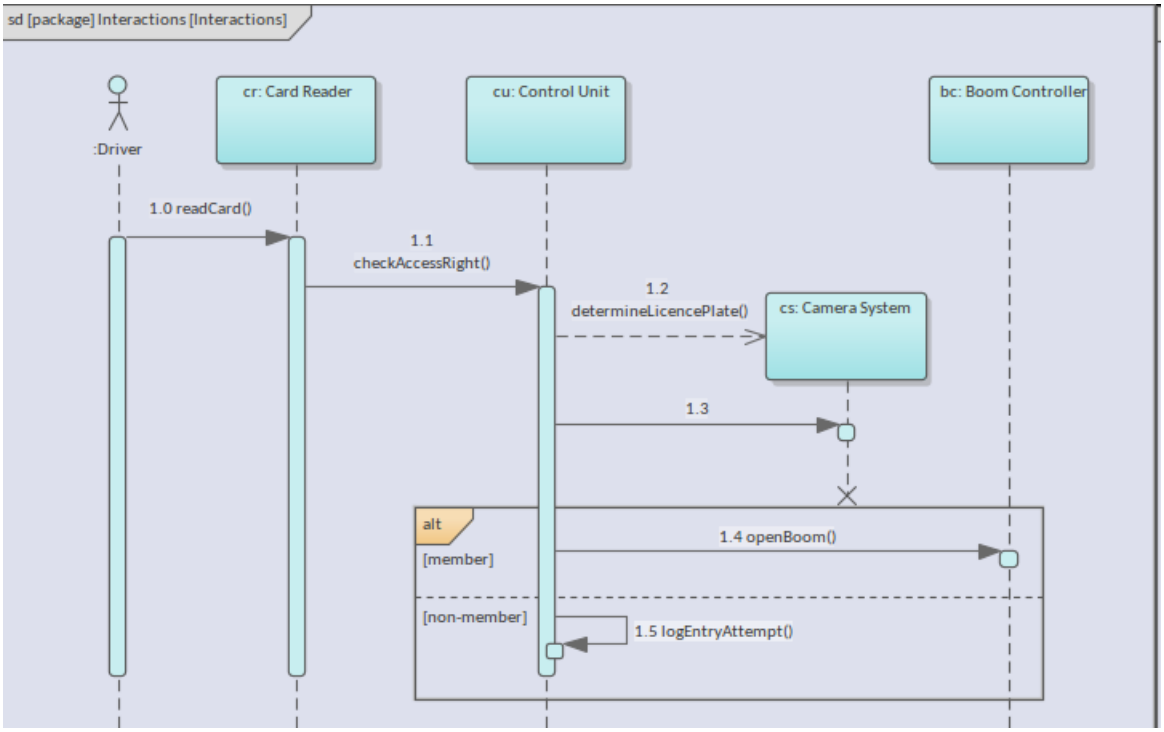
Many systems are inherently complex and while simple Sequence diagrams are useful for conveying an overall picture of a piece of software or an electro-mechanical device they need to be augmented to allow sophisticated models of these more complex systems to be created. One of the options for modeling complexity in message flows is the Combined Fragment. These can be used to sequence messages differently, including being able to select particular messages in certain circumstances or to execute a message a number of times. There is a set of combined fragments that can be used and their operator determines the type of fragment. Enterprise Architect supports all the operators, allowing engineers to create diagrams that can adequately describe the complex engineering systems being modeled. Fragments can be added to a diagram directly from the Interactions toolbox page and can be positioned to overlay the appropriate group of messages.



The element placed on the diagram is a generic Fragment and will need to have its operator set. This can be achieved by selecting a value for the type in the Combined Fragment property sheet.



This will change the fragment to the appropriate type and allow, for example with the *alt* type, to set any number of alternative conditions that will determine which message will be fired.


sd [package] Interactions [Interactions]
:Driver
cr: Card Reader
cu: Control Unit
bc: Boom Controller
cs: Camera System
1.0 readCard()
1.1
checkAccessRight()
1.2
determineLicencePlate()
1.3
alt
[member]
1.4 openBoom()
[non-member]
1.5 logEntryAttempt()

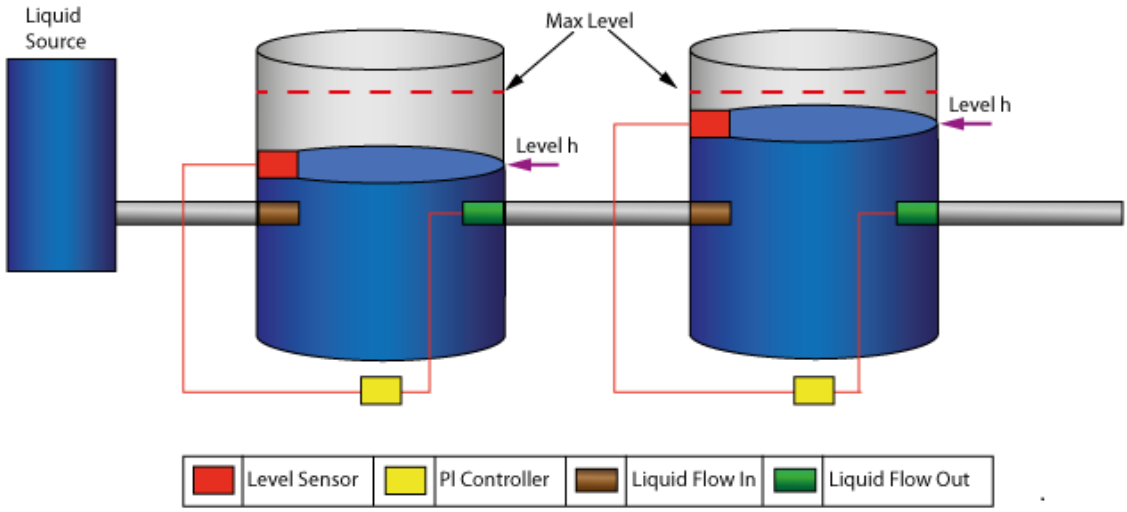
Visualizing with Simulations

Through the centuries, humans have expanded their knowledge of the world through the study of mathematics and physics and the application of systems of thought and equations to real world problems. Using this knowledge we have built motor cars and aircraft, sent humans to the moon, split the atom and solved innumerable other complex problems. This knowledge, in times of antiquity, was transmitted by word of mouth, clay tablets and papyrus scrolls and later scribes' laboriously created books. Then, with the advent of the printing press the knowledge was written down in the form of articles, journals and books and disseminated widely. Four hundred years would pass before the Internet arrived and much of the existing knowledge would be transferred to online material in the form of documents, pages and sites dedicated to these disciplines - making it available to a vast number of people in all corners of the globe. Now a new era has arrived where the knowledge can be used to construct models that allow us to visualize these equations in motion and in context, with parametrics specific to our domain and the problems we are trying to understand and solve.

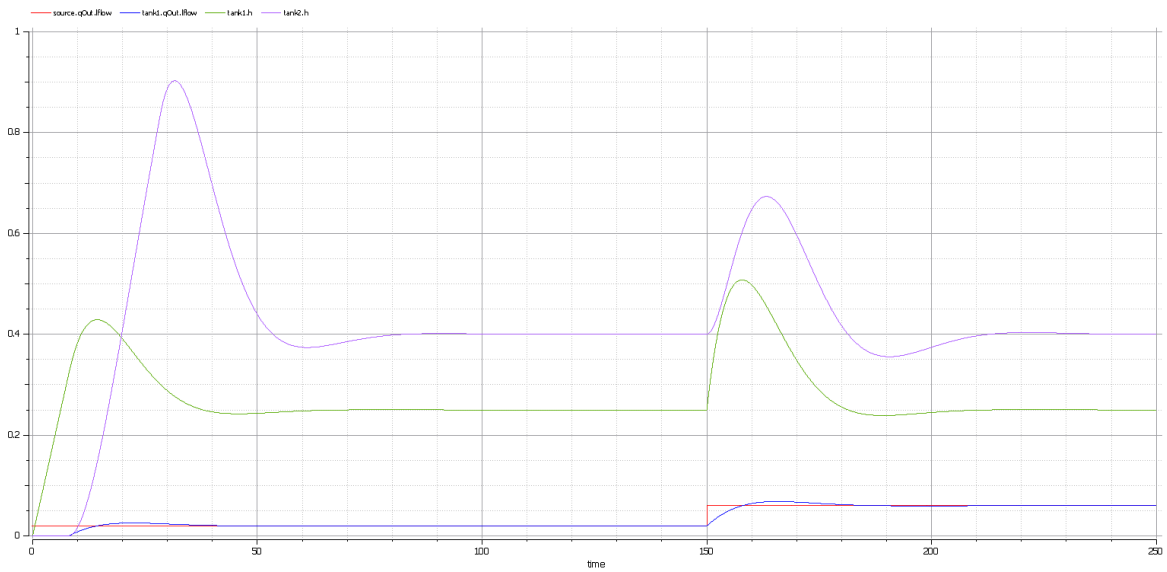
SysML Simulation in Modelica and Simulink

Enterprise Architect, as a leading Systems Engineering tool, allows models to be constructed using industry-compliant modeling techniques and languages for the representation of cyber-mechanical systems. These models act as devices for communication between collaborating engineers, teams of consultants and others, but can also be used to generate advanced visualizations using industry-standard modeling languages used by *OpenModelica* and *MATLAB's Simulink.*

This example shows the power of Enterprise Architect in leveraging existing open standards to visualize solutions. The diagram depicts two tanks connected together, and a water source that fills the first tank. Two continuous controllers are used to regulate the flow of water from the first tank to the second, and the output from the second tank.

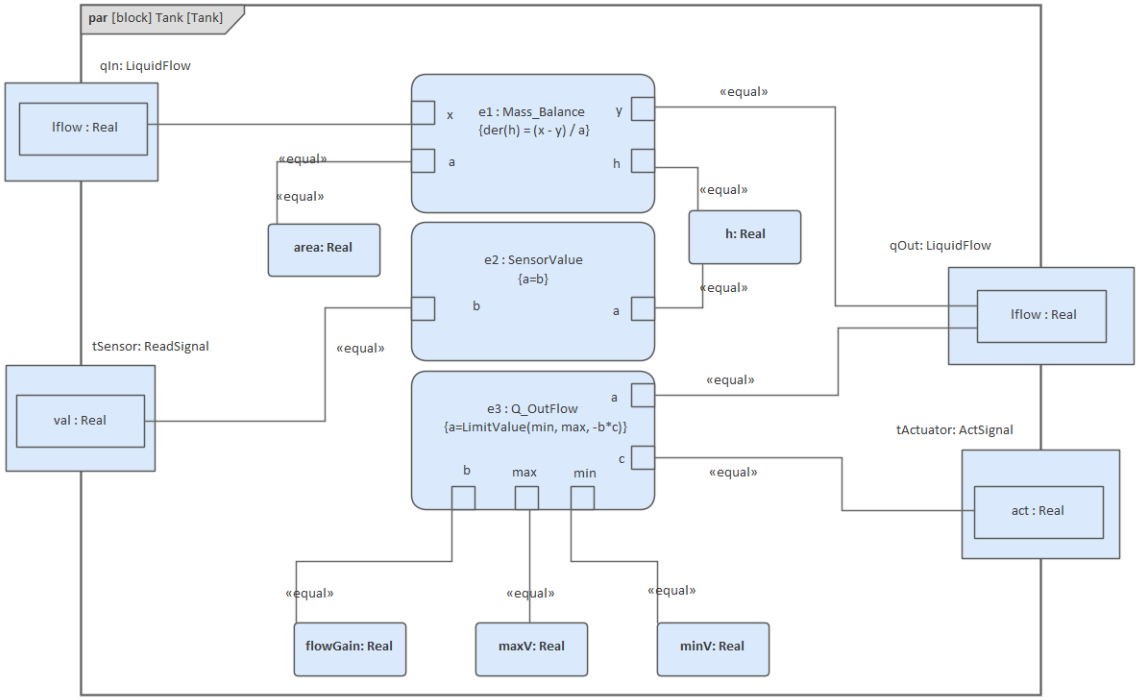


A set of diagrams is created in Enterprise Architect that models the physical aspects of the tanks and the equations (written in the Modelica/MATLAB language) that characterize the flows between the tanks. Once the simulation has been configured and the *Solve button* selected, the output resembles this diagram:



We will discuss the details of the constraint and parametric modeling in a later section of this topic, and see how the models that we create are simply Block Definition and parametric diagrams that we learnt about in an earlier section of the guide. This Parametric diagram shows an example of the modeling for the two-tank problem, using constraint properties and connectors that bind the parameters into a system of equations. Other diagrams are necessary to produce

the result but this is the main diagram that shows the mass, flows and the sensor determining the level in the tank.



How SysML Simulation Works

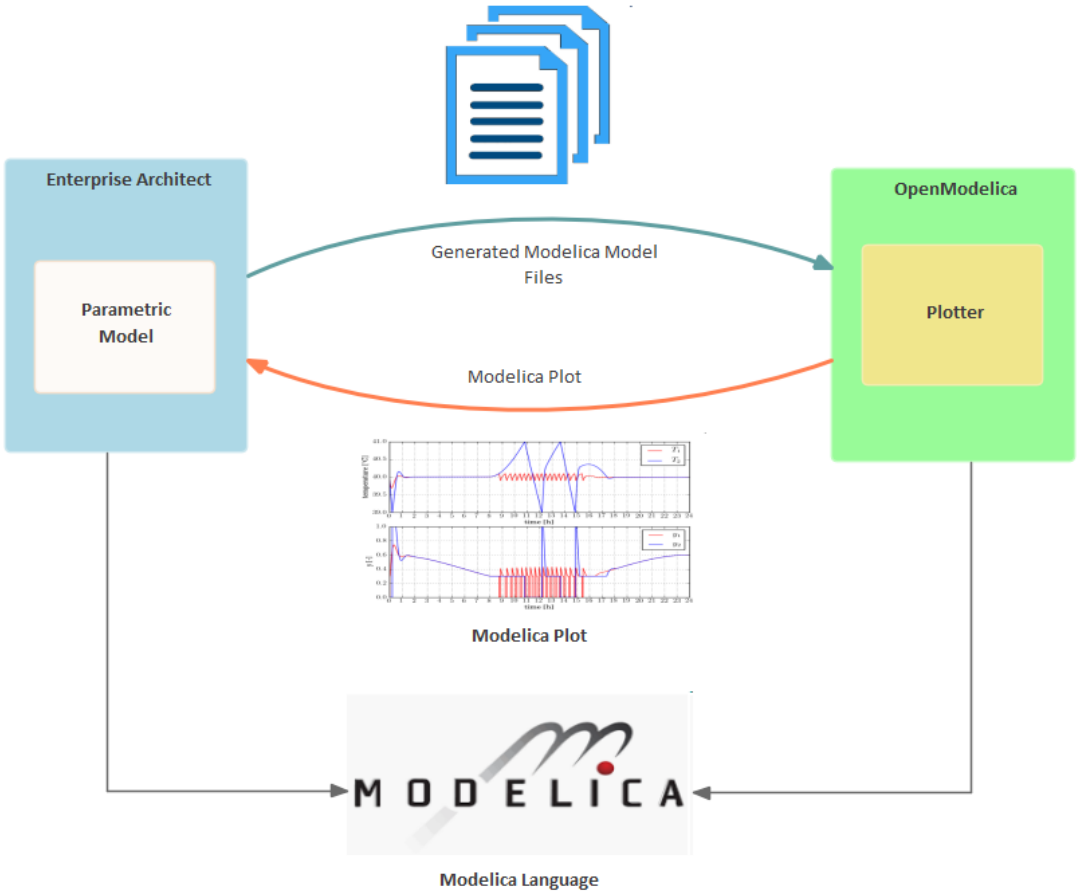
Enterprise Architect has been built on open standards and the product teams are ever-aware of the power and efficiency of utilizing existing solutions and not reinventing the wheel. The tool uses these standards to allow the visualization to take place:

- Systems Modeling Language (SysML) - managed by the Object Management Group (OMG)
- OpenModelica - managed by the Open Source Modelica Consortium (OSMC)
- Modelica - managed by the Modelica Association

We have spent a deal of time in earlier topics learning about the SysML; in fact we have already learnt most of what we require to create the Block Definition and Parametric examples for these visualizations. In addition we will learn how to add some extra information that OpenModelica needs; this will be discussed in the next section.

Modelica is an open and object-oriented language based on equations, allowing the modeling of cyber-mechanical systems utilizing sub-components. Like its mathematical cousins, Modelica is a cross-domain language that has a wide variety of applications, including in mechanical, electrical, electronic, hydraulic, thermal, control, electric power and process-oriented domains, to name the possible sub-components of a Modelica model and the types of system that can be modeled using the tool.

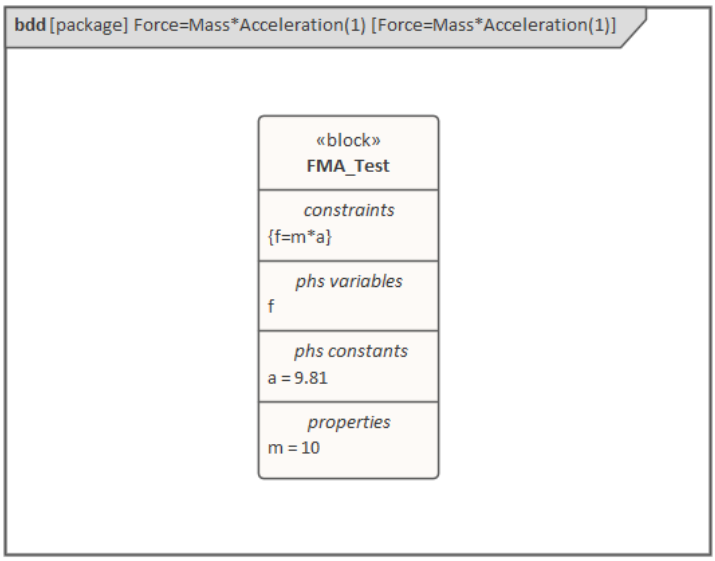
Enterprise Architect is capable of performing basic simulations for process diagrams and StateMachines, but for modeling complex cyber-mechanical systems it makes use of the power of OpenModelica, underpinned by the Modelica language itself to do the heavy lifting. Enterprise Architect allows these cyber-mechanical models to be related to a wide range of other systems and software engineering artifacts, including missions, stakeholder requirements, StateMachines, programming code, Decision Tables, architectures, trade off analysis and much more.



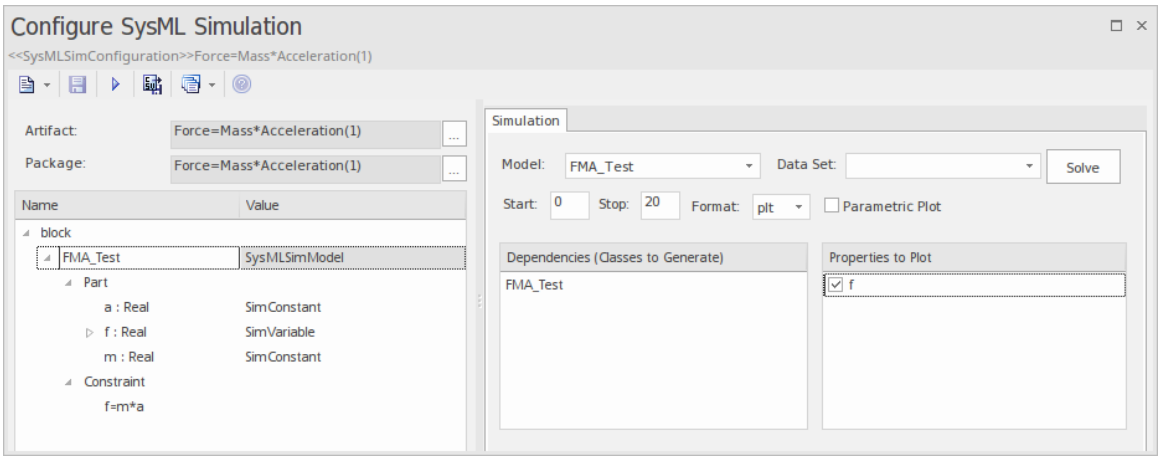
An overview of how it works can best be provided by way of a simplified example. To create a simple visualization of Newton's Second Law - 'The rate of change of momentum is proportional to the force acting and takes place in the

direction of that force.' (F = m*a), an engineer using Enterprise Architect will:

1. Create a Block diagram describing the equations using Constraints and Values.



2. Configure the SysMLSim Configuration Artifact (used to define the information needed by OpenModelica).



3. Run the Simulation by selecting the Solve button on the window.

A chart will be plotted with f = 98.1 (which is the product of the Mass (10) and Acceleration (9.81) expressed in the equation [f = m * a] with the value in place 98.1 = 10 * 9.81). This is a simplistic example aimed at showing the fundamental ingredients to create the visualization; we will look at more advanced examples in later sections showing the use of Constraint Properties and User Defined Data Sets.

Getting Started with OpenModelica

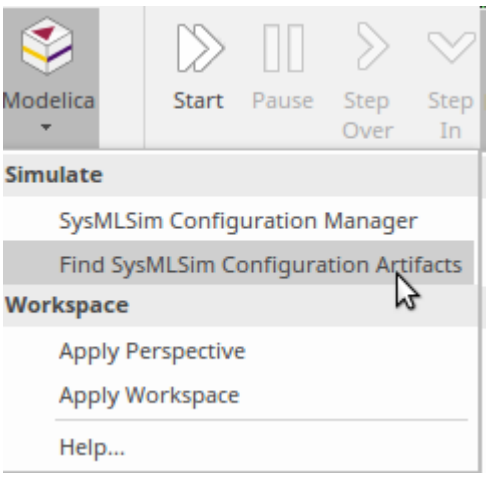
An easy way to get started with OpenModelica and Enterprise Architect's machinery to produce Parametric simulations is to view some existing examples. This is a useful learning device for any feature in Enterprise Architect, but is particularly pertinent when learning the power of OpenModelica as there are a number of new things to learn and this is best done with some learning aids. We will start by looking at a fully worked exemplar taken from the Example model, which is distributed with every installation of Enterprise Architect and is available from the *Help* Item on the *Start* ribbon.

The example we will explore is the Pendulum example, but firstly we need to open the example model, which we do by selecting the ribbon option:

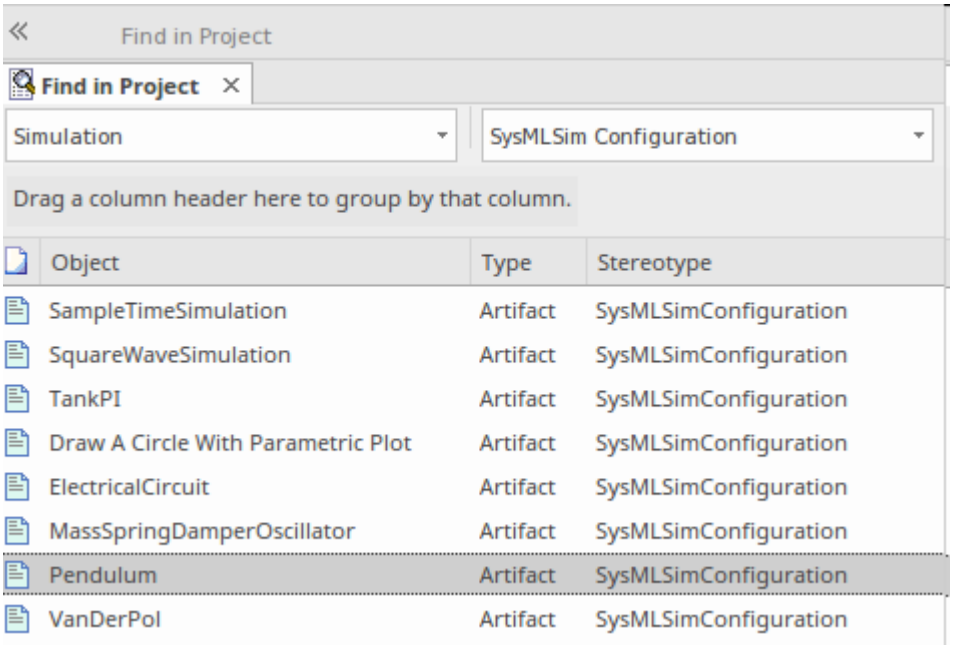
Start > Help > Help > Resources > Open Example Model

The OpenModelica features are conveniently grouped in a single location with other simulation capabilities, and can be launched using the Modelica/Simulink menu available from the 'System Behavior' panel of the 'Simulate' ribbon. The OpenModelica facility keeps company with other simulation tools such as Executable StateMachine, Decision Modeling Notation and BPSim.

With the model loaded we can use the OpenModelica features within Enterprise Architect to locate the Pendulum example in the model. It is common for a model of a complex system to be very large, and there could be any number of existing simulations set up, so Enterprise Architect provides a mechanism to search for these simulations.

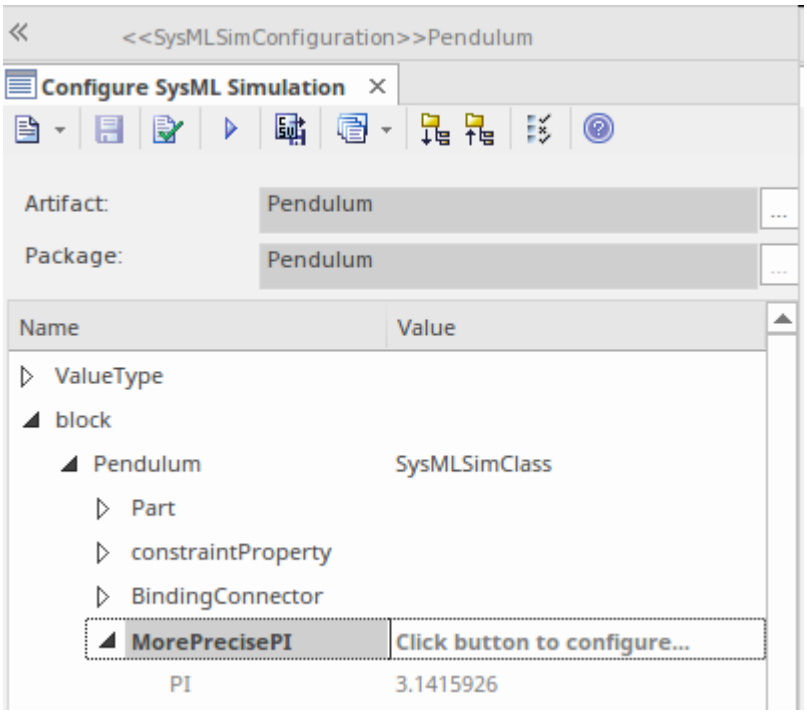


Selecting this option will return a list of SysML simulation configuration artifacts, which are the elements where the OpenModelica details are specified. From this list we can select the Pendulum example, which will launch a window that can be used to configure and solve the problem codified in the example. To be able to run the simulation, a version of OpenModelica must be installed on the machine that is running Enterprise Architect. Details on how to install OpenModelica are contained in the next section.

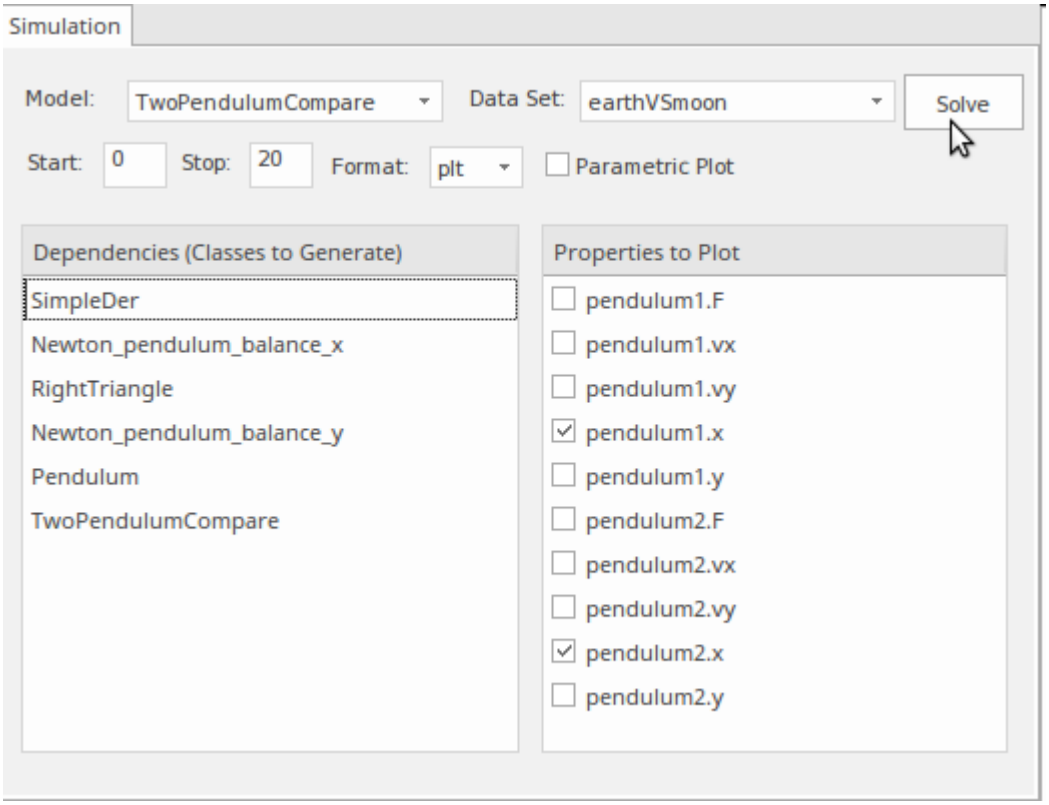


There are two sections to the window:

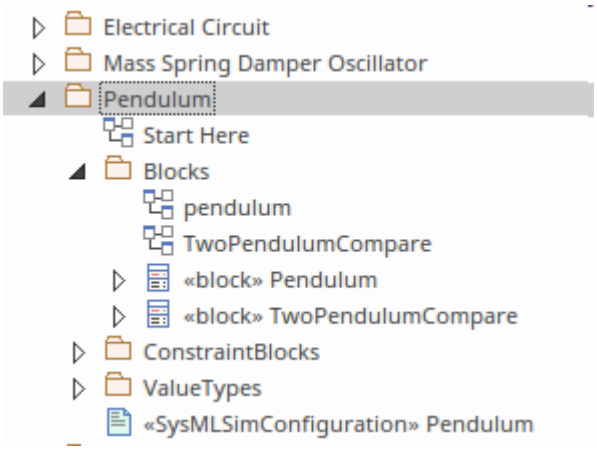
- The configuration (left hand) panel defines the Value Types, Blocks and Constraint Properties, all of which are defined in the repository and - as we will see later - can be viewed in the Browser window and diagrams



- The simulation (right hand) panel is used to select and specify options that will effect the simulation, such as the data-set, the wait time and duration of the simulation, output file types and more.



The information in the window is automatically populated from the model that is visible in the Browser window, and the location of the elements - including Value Types, Blocks and Constraint Properties - can be found using the 'Find in Project Browser' option from the context menu. There is also an analogous option to find the selected element in any diagrams in which it appears.



Installing OpenModelica

Enterprise Architect utilizes the power of the OpenModelica platform, so when you run a simulation from the Simulation window it is effectively calling out to OpenModelica (installed on the same machine) to do the heavy lifting and return the simulation results. This ensures that Enterprise Architect leverages the power of this open tool and all of the brilliant minds that have contributed to its excellence. There are both Windows and Linux version of OpenModelica and you will need to install the one appropriate for your environment. The steps are summarized here.

1. Download the OpenModelica software (Windows or Linux).
2. Install the Software.
3. Check the Installation.
4. Configure the Solver by specifying the path in Enterprise Architect.

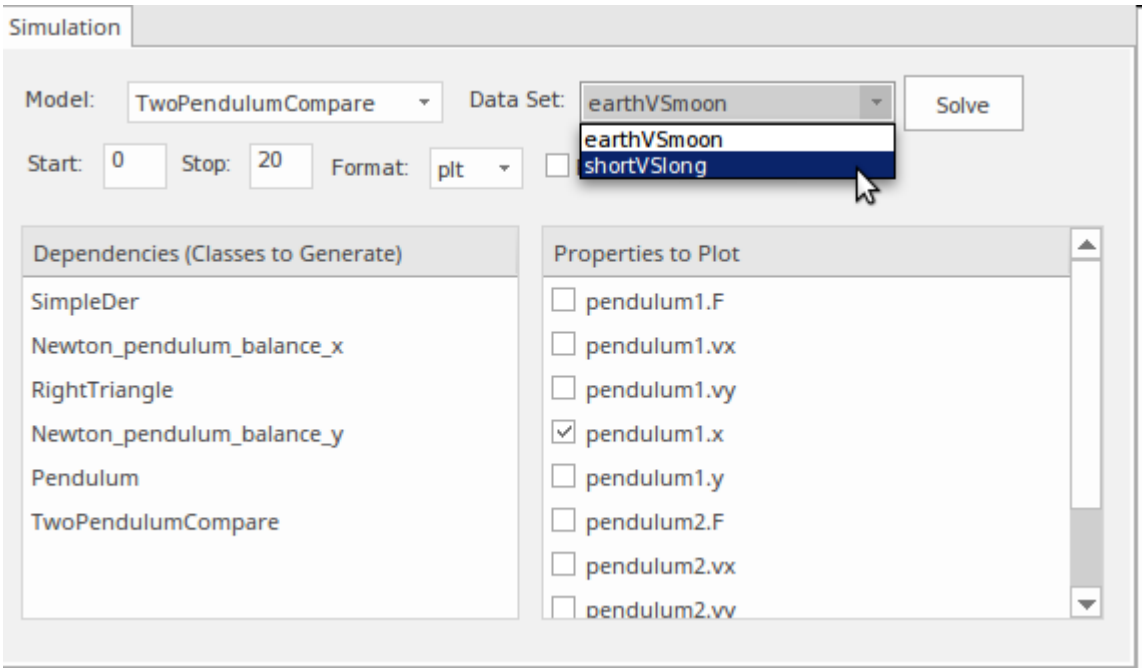
Full details of the installation and configuration can be found in the OpenModelica Integration Help topic.

Simulation with Data Sets

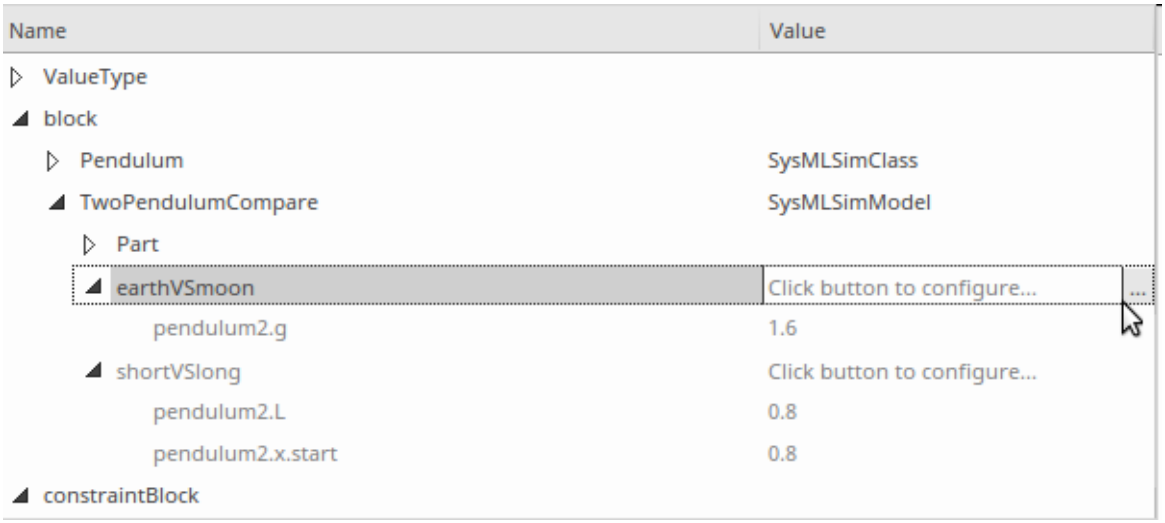
Simulation forms an import aspect of engineering analysis and provides a useful and cost effective way of analyzing a system's behavior. The system might already have been built or it might be planned for development; either way, simulation can be used to visualize component or system design considerations, performance optimization, safety engineering and much more. To provide this capability it is important to have the ability to run a simulation using different values for variables and constants. For example, in our pendulum example we might want to analyze how the system would perform on different planets, to examine, say:

- The effect of terrestrial versus lunar (or other planet's) gravitational force
- The effect of different string length
- The effect of different masses
- The effect of different starting point
- Any combination of the above

Enterprise Architect provides a Data Set facility for simulations that can be applied at the Block level. Using the 'Simulation' tab (right hand panel) of the SysMLSim Configuration window we can select predefined data sets.



Any number of data sets can be defined and can be added to the appropriate Block in the 'Configuration' (left-hand) panel of the Simulation window. These are available as nodes under each Block and there is an option to view and edit the data values in a window by using the Browse [...] button on the datasets row.



When selected, the Configure Simulation Data window will be opened allowing values to be viewed and edited, imported or exported. This mechanism means that the simulation machinery can be reused in many different contexts, and engineering organizations that focus on particular types of problems can create libraries of simulations that could be reused in a multitude of contexts and types of engineering problem. The window illustrated here shows a data set that contains values pertaining to the two pendulum problems we have been looking at and we can see as an example the acceleration due to lunar gravity has been defined as an approximation of 1.6 m/s2 approximately 16.6% of the value on the surface of the earth. This simulation could be reused with a different data sets applicable to Mars or Jupiter or in a more terrestrial example with a different mass or length of string.

Attribute	Stereotype	Type	Default Value	Value
pendulum2	SimVariable	Pendulum		
x	SimVariable	Real	0.5	
L	SimConstant	Real	0.5	
m	SimConstant	Real	1	
F	SimVariable	Real		
PI	SimConstant	Real	3.1415926	
vy	SimVariable	Real		
y	SimVariable	Real	0	
vx	SimVariable	Real		
g	SimConstant	Real	9.81	1.6
pendulum1	SimVariable	Pendulum		

Creating Models for Simulation

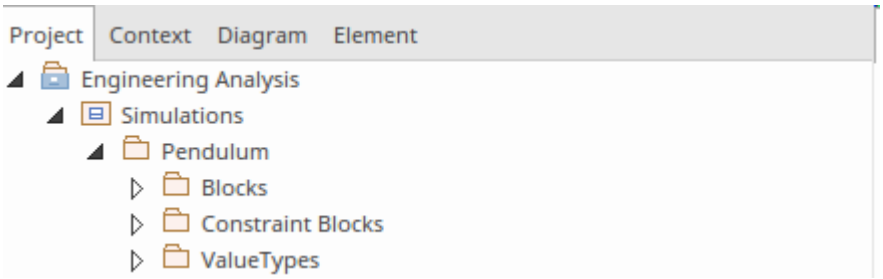
The example that we looked at in the *How it Works* section of this Guide was deliberately trivial; in this section we will look at these steps in more detail, including options for configuring the models. Enterprise Architect provides a lot of flexibility and allows trivial once-off models to be defined, or sophisticated library based multipurpose models to be created that can be reused across multiple projects and domains using different data sets and contexts. In this topic we will explore some of these options for configuring the models so that they are fit for purpose and will create the required engineering outcomes.

Creating a Simulation Model Package

A Model Package can be set up at any location in the Browser window, but typically it is considered best practice to set up a *Simulations* Package under each project or initiative. This could then contain sub-Packages for each simulation. It is anticipated that for a given project there could be a number of different simulations required. It is also likely, as discussed in the Data-Sets section, that an organization will want to reuse some of the simulations across multiple projects. These simulations could be set up in the Project Browser and defined at a supra-project level - for example at an enterprise, organization or engineering department level.. They could then be included in a diagram at the project level, indicating that they are applicable to a given project or problem context. It is also likely that an engineering team will want to reuse Value Types and their concomitant Quantity and Unit Libraries between projects, and these, as discussed in an earlier topic of the guide, are best defined and modeled at a supra-project level.

Enterprise Architect uses the SysML Package Import mechanism to ensure the Value Types defined at the enterprise level can be included and reused at each Simulation Package level. The structure would typically contain these Packages:

1. Value Types (specific to this project)
2. Blocks
3. Constraint Properties



In the next section we will learn how to create and configure the SysML Simulation Artifact, which is stereotyped as <<SysMLSimConfiguration>>.

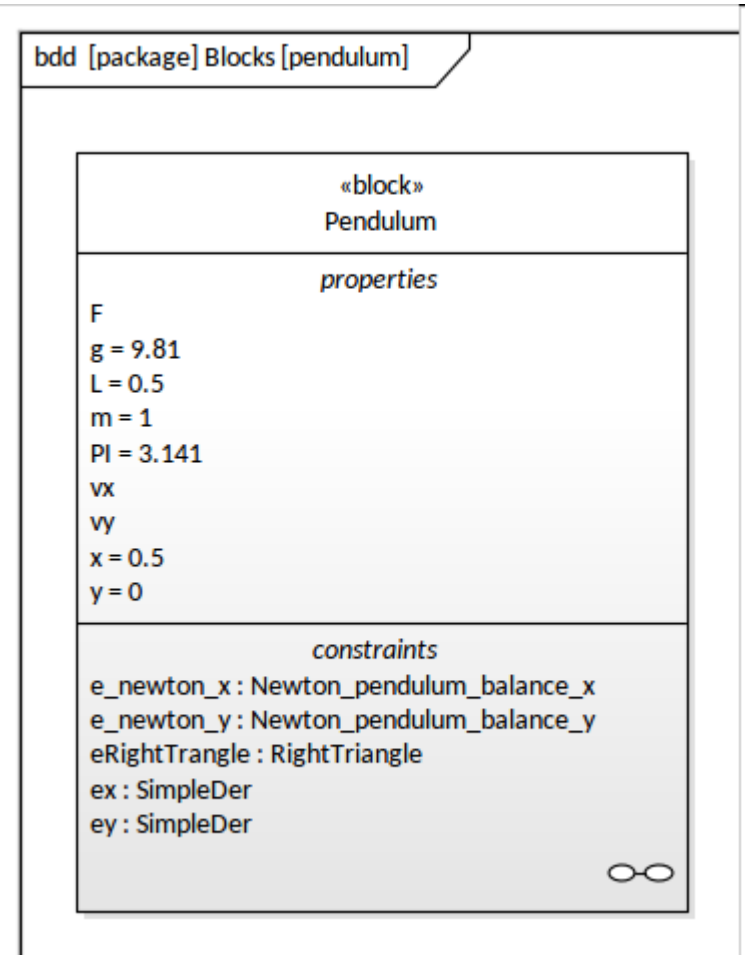
Creating the Simulation Elements

The main effort in preparing a simulation is in the definition of the model elements, using the SySML with the appropriate level of precision to allow the OpenModelica platform to run the simulation. There are a number of ways that the models can be defined, and in this Guide we will focus on the most robust and flexible method as this is what will be used by most practicing engineers and teams.

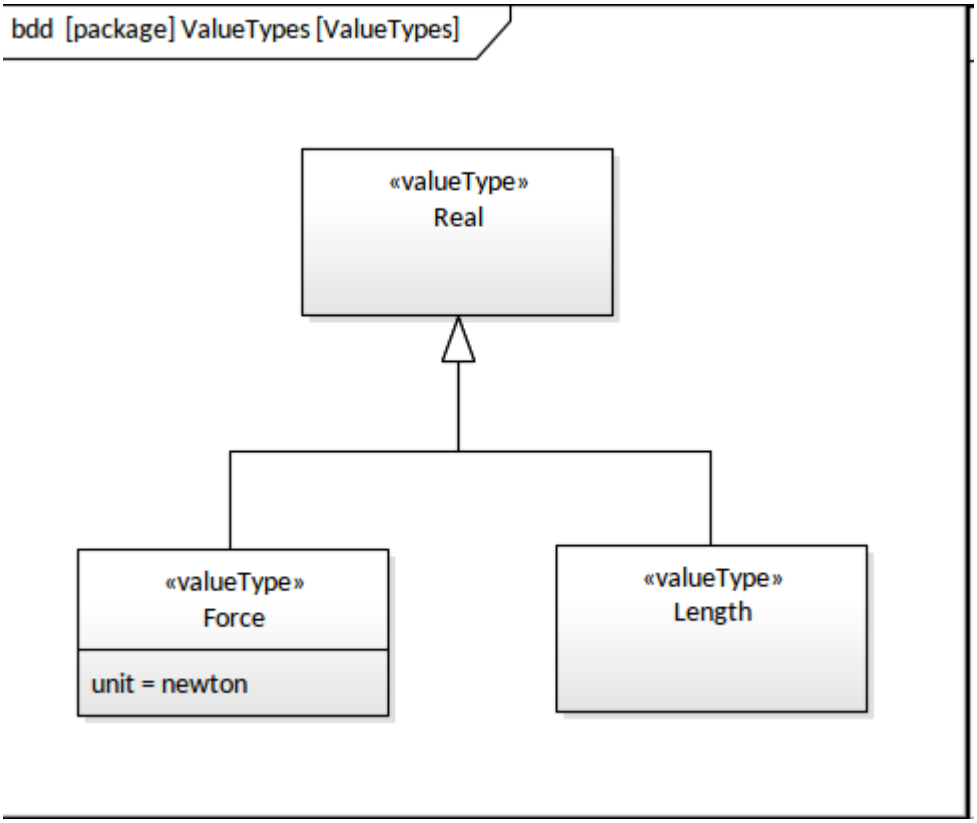
At this point it is worth looking before the topic of simulation to ensure we understand its relevance and connection to other parts of the model, and how simulation elements might be connected to other model elements such as Requirements, Test Cases and more. Typically, simulations are used as a way of investigating some cyber-physical

problem without the need to construct a time-consuming and often expensive physical model. The simulation could be part of problem analysis, trade off analysis, performance analysis or a number of other investigations. The Blocks that are defined as part of the simulation could be allocated to behavioral elements and ultimately to Requirements.

The first elements to be created are the Blocks, which are the fundamental structural elements of the solution. We have learnt how to do this in a previous example; this diagram shows a number of the compartments that have elements, namely the properties.



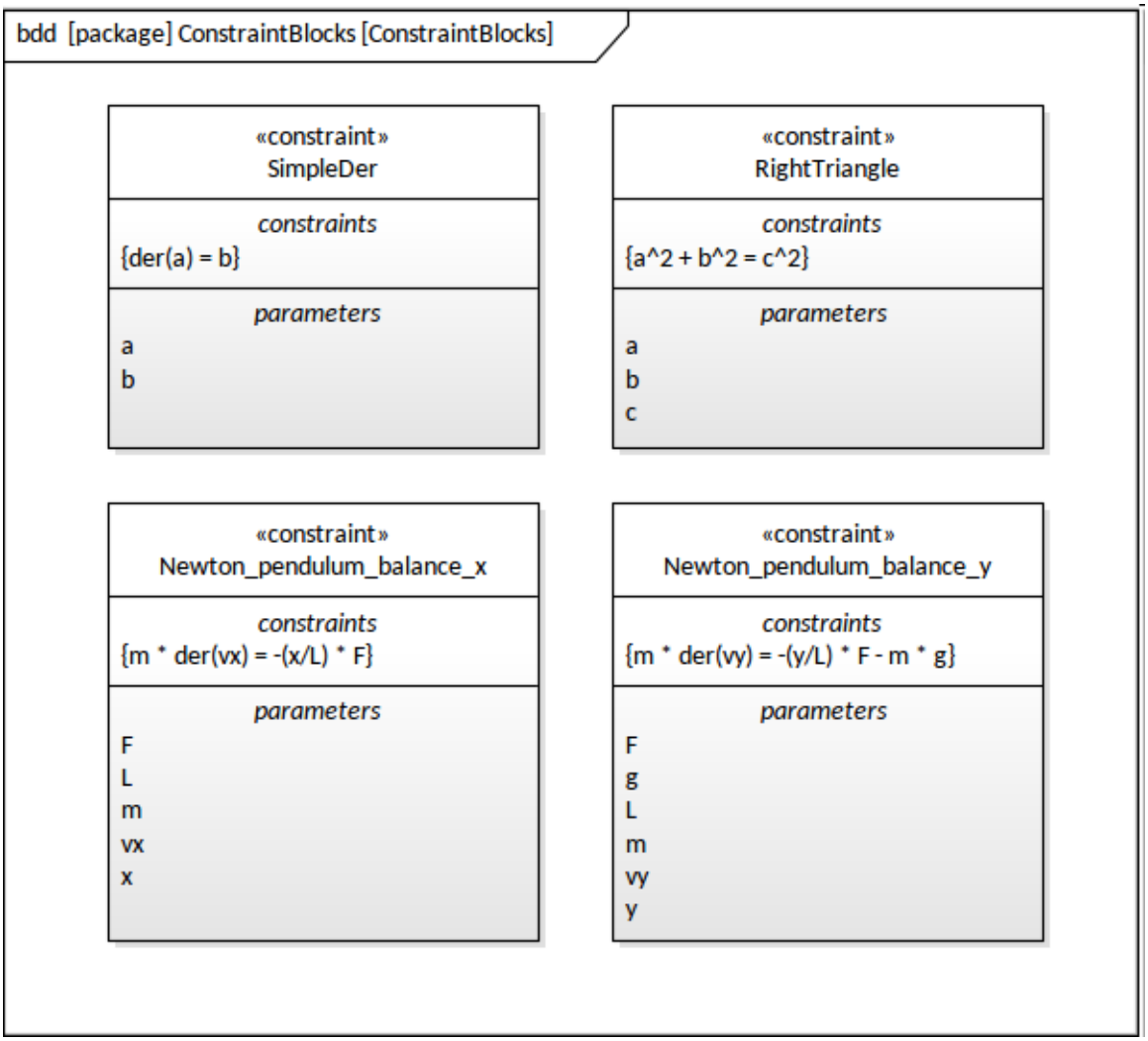
In preparation for defining the equations that define the behavior of the pendulum system, we will define the value types required to ensure the model is precise and check that the simulation parameters are correctly specified. This is done using a Block Definition diagram (bdd), using the Value Type element available from the Diagram Toolbox.



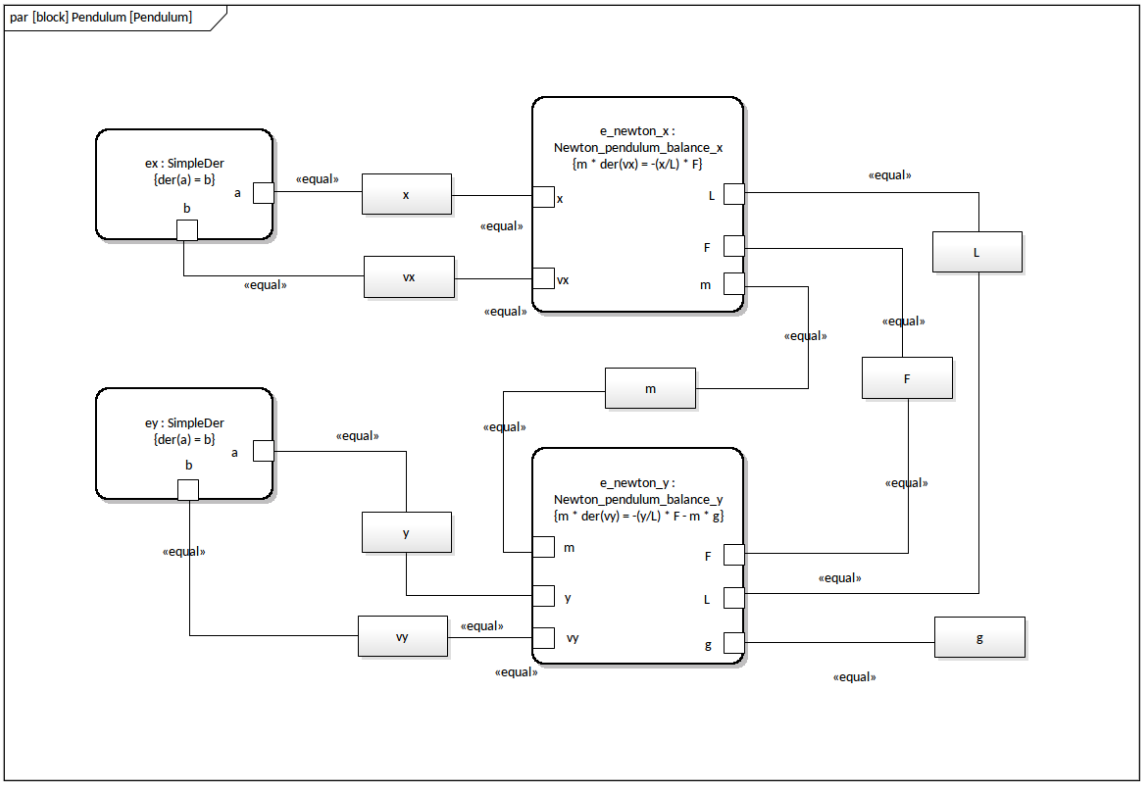
The next step is to define the constraints using ConstraintBlocks. These elements are where we will define the Modelica equations that will govern the behavior of the system being modeled - in our case, a pendulum. As described earlier in this Guide, ConstraintBlocks are defined on a Block Definition diagram, and have a series of parameters defined and a constraint that expresses those parameters in an equation written in Modelica. For example, the equation that constrains the vertical aspect of the pendulum is written as:

$$m * der(vy) = -(y/L) * F - m * g$$

Notice the Modelica keywords such as 'der' signifying a first order derivative. L is the length (parameter) of the Pendulum, g is the acceleration due to gravity (constant), m is the mass (parameter) of the pendulum, x and y are the coordinates in two-dimensional space, and F is the force. Notice that Modelica uses:



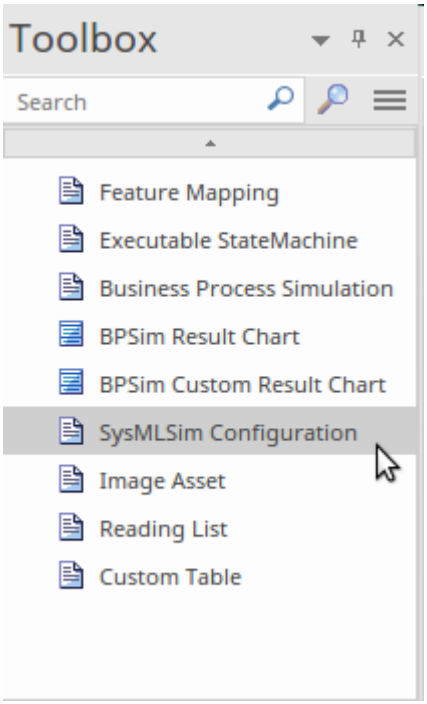
The next step is to create the Parametric diagrams that bind the equations together. As discussed earlier in the Guide, these diagrams are a specialized type of Internal Block diagram and contain instances of ConstraintBlocks called ConstraintProperties that expose their parameters, which are bound by connectors to parameters on other ConstraintProperties.



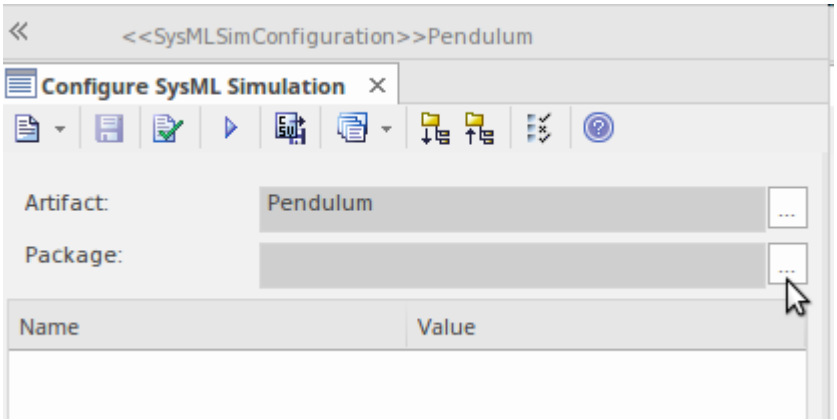
Specifying a Configuration Artifact

The SysML Simulation Artifact is the element that binds the model elements to the OpenModelica platform. The model elements expressed in SysML in the form of Blocks, ConstraintBlocks and their related ConstraintProperties bound together on Parametric diagrams appear in the Simulation window and can be configured with other settings to drive the simulation.

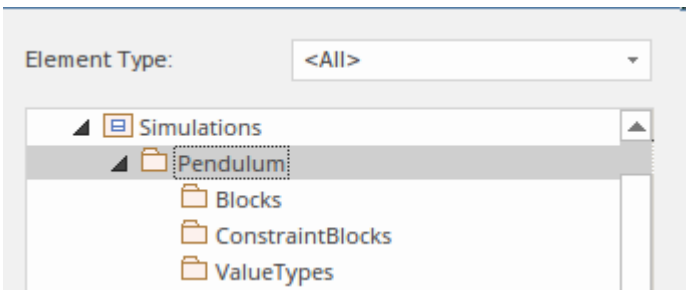
The first step in setting up this facility is to create a SysMLSim Configuration element, available from the 'Artifact' Toolbox page as shown in this screen capture.



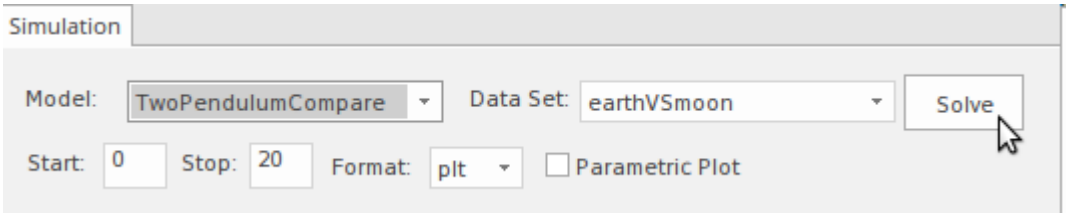
Enterprise Architect does most of the heavy lifting with respect to populating the Configure SysML Simulation window. The engineer simply has to select the Package using the ... icon on the 'Configuration' panel of the Simulation Artifact window.



Enterprise Architect will display the Package selection window, and once the Package is selected the 'Configuration' panel will be populated with the Blocks, ConstraintBlocks and Value Types from the model. From this point the values can be entered for various parameters, or data-sets can be defined. The simulation configuration parameters can be entered and the simulation is ready to run.



Once the data has been entered for the simulation, including Start and Stop values and Output formats, the Simulation can be run by selecting the Solve button as shown in this screen illustration.


Simulation
Model:
TwoPendulumCompare
Data Set:
earthVSmoon
Solve
Start:
0
Stop:
20
Format:
plt
Parametric Plot

Example SysML Model

The System Modeling Language (SysML), as with any language, has to be learned. We are not lucky enough to have grown up hearing our parents speak the language at home, but a number of readers might already be familiar with the language for any number of reasons including:

- It has been used by colleagues or partners in projects
- It has been taught as part of a University course
- You have attended training or read or viewed documentation
- You have taken 6 weeks off work and read the specification from cover to cover

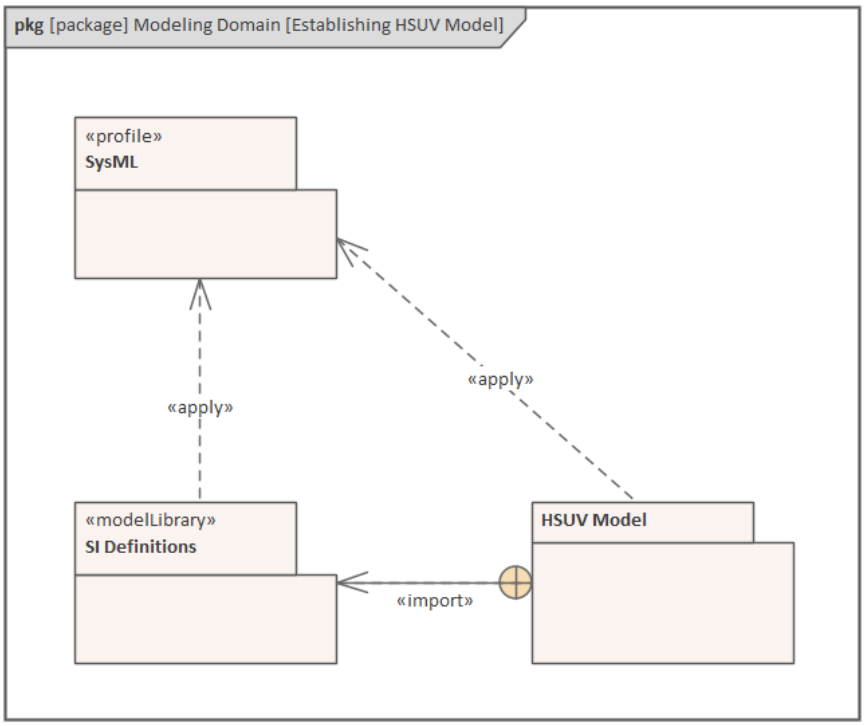
If the last one is true you will probably have a deep knowledge of the language but might be in need of some recreational leave as it is a fairly dense document and you will have needed to dip your toes into the Unified Modeling Language specification as well. It is more likely that a larger number of readers will have little or no exposure to the language, and this first example is intended to give you a quick and high level view of what can be expected when working in Enterprise Architect to model a Model Based Systems Engineering Project using the SysML. It is based on the example of a hybrid vehicle that appears in the Sample Problem Annex in the specification

Package Overview (Structure of the Sample Model)

The Package diagram demonstrates a way of visualizing the contents of the repository; when the diagram contents are viewed in Enterprise Architect's Browser window the structure can be navigated. There are also important structural and namespace relationships that can be seen on Package diagrams and these help to clarify the important high-level relationships between groups of elements in the repository.

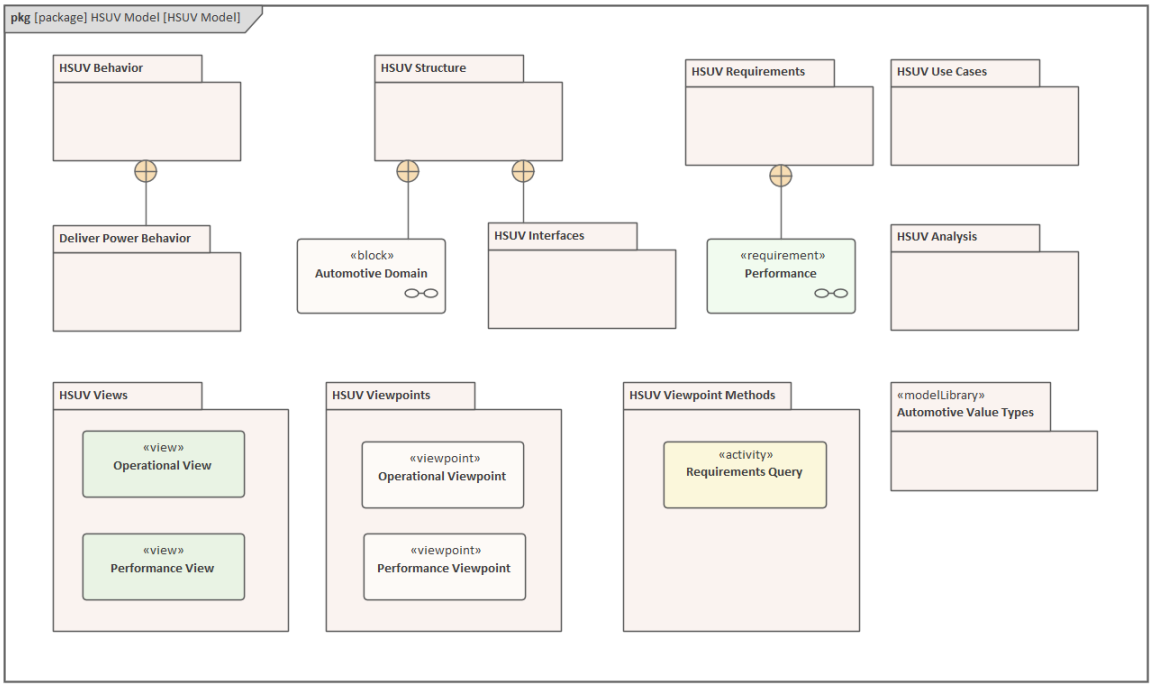
Package Diagram - Applying the SysML Profile

As shown in this diagram, the HSUV Model is a Package that represents the user model. The SysML Profile is applied to this Package in order to include stereotypes from the profile. The HSUVModel might also require model libraries, such as the SI Units Types model library. The model libraries are imported into the user model as indicated.

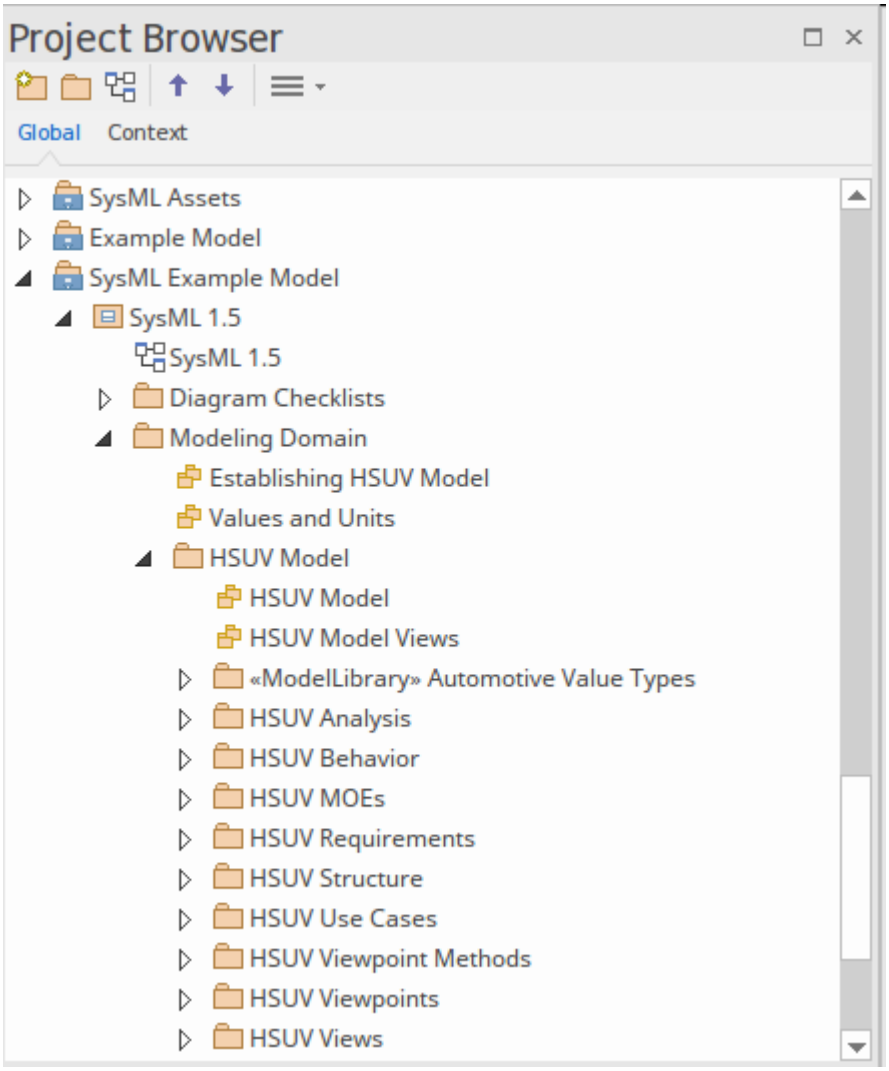


Package Diagram - Showing Package Structure of the Model

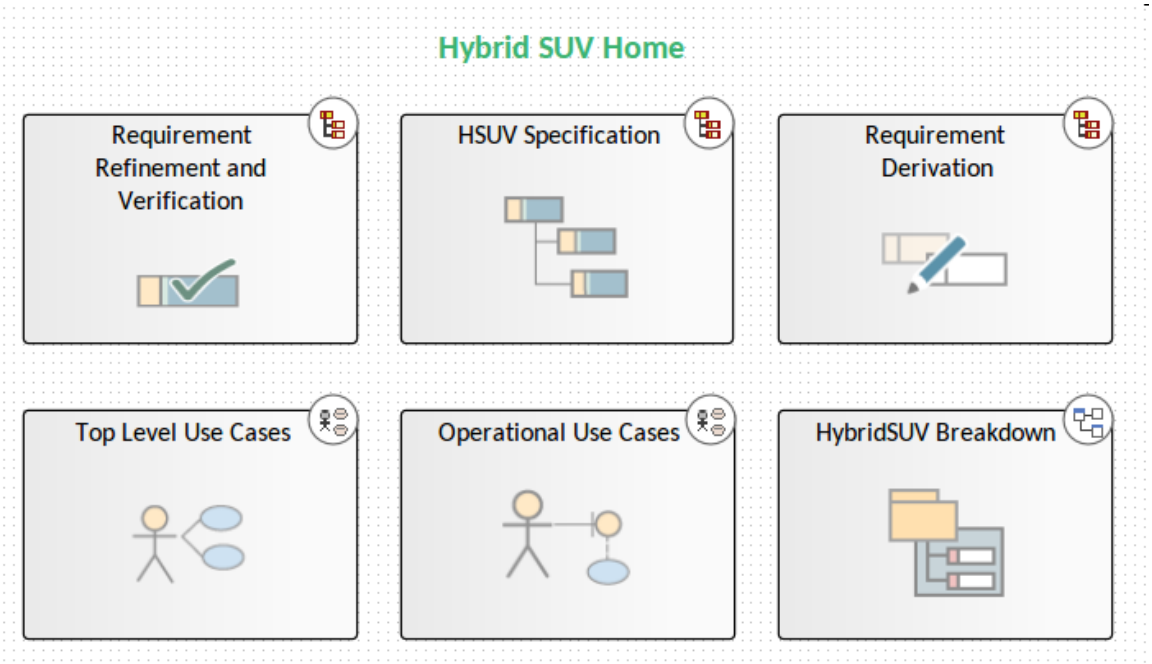
This Package diagram shows the structure of the model used to define the HybridSUV system. The diagram provides a useful way of viewing the high level containers (Packages) used to structure the repository and define the structural, behavioral and requirements of the HybridSUV system.



The relationships between the Operational View and the Performance View and the rest of the user model are explicitly expressed using the «import» relationship. The Packages that appear in the diagram are defined and can also be visualized in a hierarchical view using the Browser window.



While the Browser window provides an important mechanism for navigating through the repository, there is a wide range of other views including - in this case - a diagram. Enterprise Architect also provides a convenient way of creating user-defined diagrams that can act as an alternative way of navigating the repository. This mechanism allows Systems Engineers and others to create any number of Navigation Cells to provide audience-tailored access to the model, shielding the user from needing to know or understand how to traverse the model.



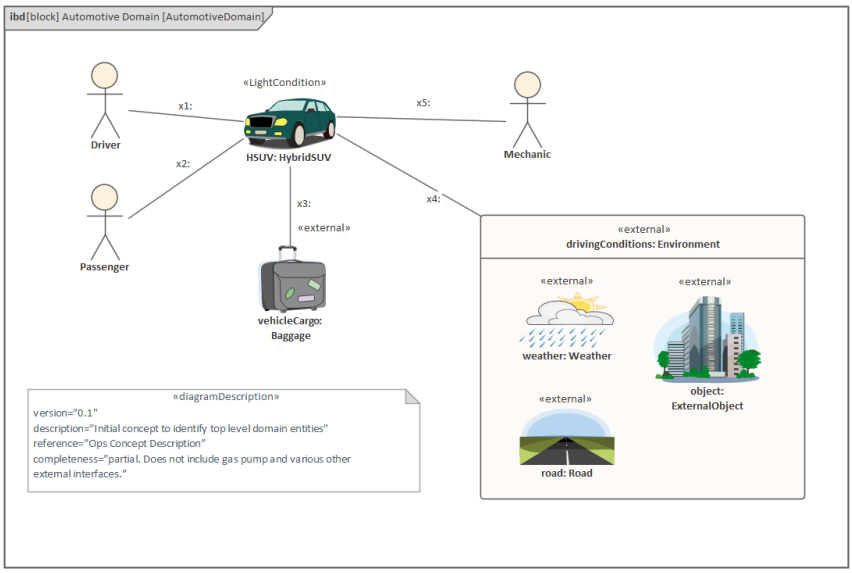
These are particularly useful when viewing the models through a Web Browser.

Setting the Context (Boundaries and Use Cases)

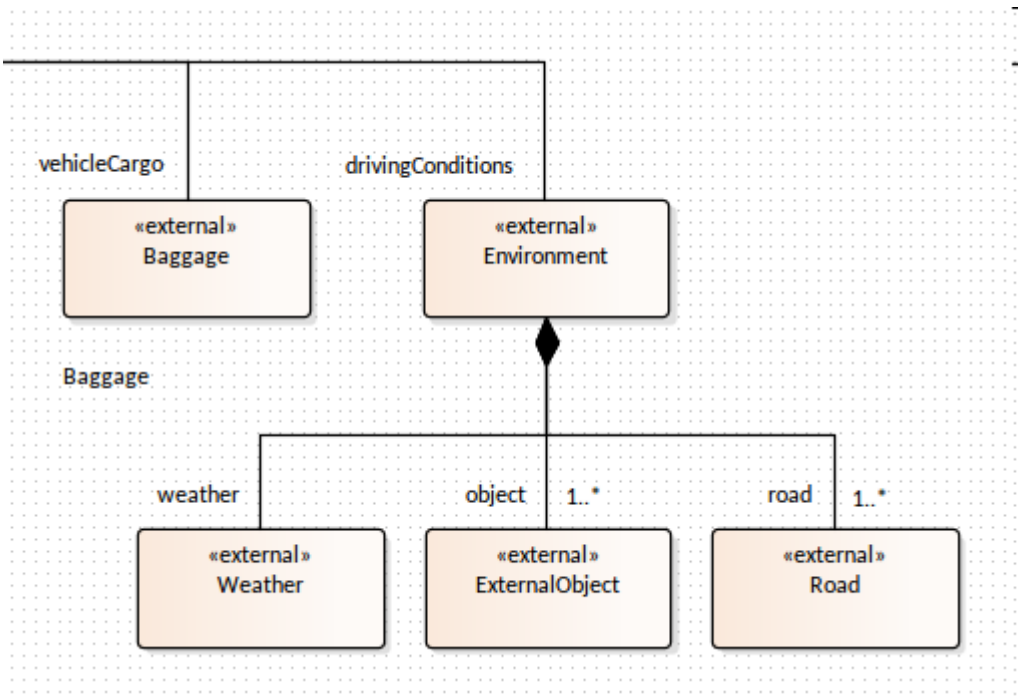
The context of a system is critical for all stakeholders to gain an understanding of the system in its environment. The Use Case diagram is one of the simplest but most descriptive diagrams in the SysML tool kit. Its power lies in the fact that it relates entities that reside outside the system (Actors) to the benefits they want from the system (Use Cases) without articulating how the system will deliver the value. The Use Cases can be written at a descriptive level, but if more detail is required Enterprise Architect's Scenario Builder can be used to specify the steps for each scenario, in a tool that takes the grind out of documenting Use Cases. Behavior diagrams such as Activity diagrams and Use Case documents can also be automatically generated from the tool.

Operational Domain Model - Setting Context

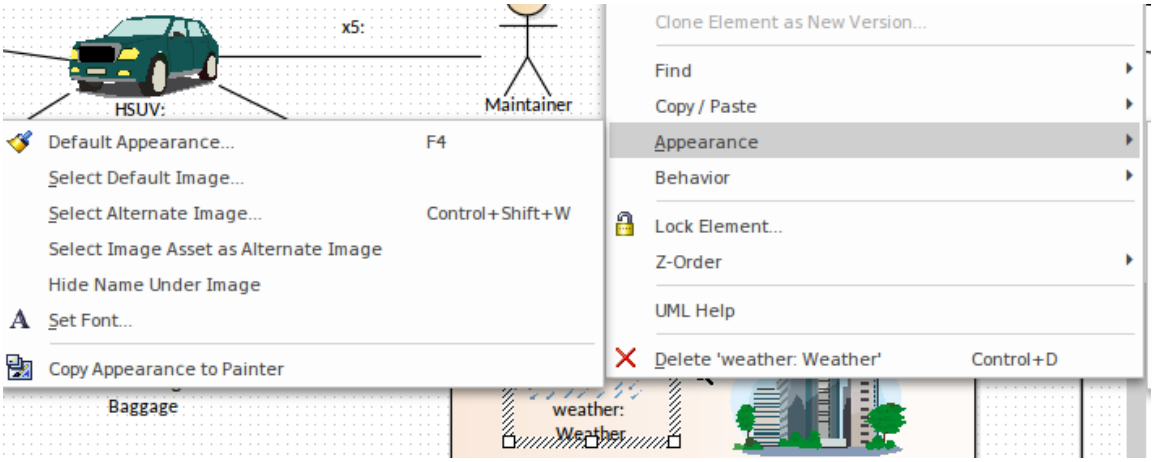
This Internal Block diagram depicts a user-defined usage of the diagram type, which depicts the system in the context of its environment. The «system» and «external» stereotypes are user-defined and not specified in SysML, but help the System Engineer describe the system of interest relative to its environment.



Enterprise Architect also allows the conventional symbols of the SysML to be replaced by more appealing and meaningful images that assist in the acceptance of the diagram by non-technical audiences.

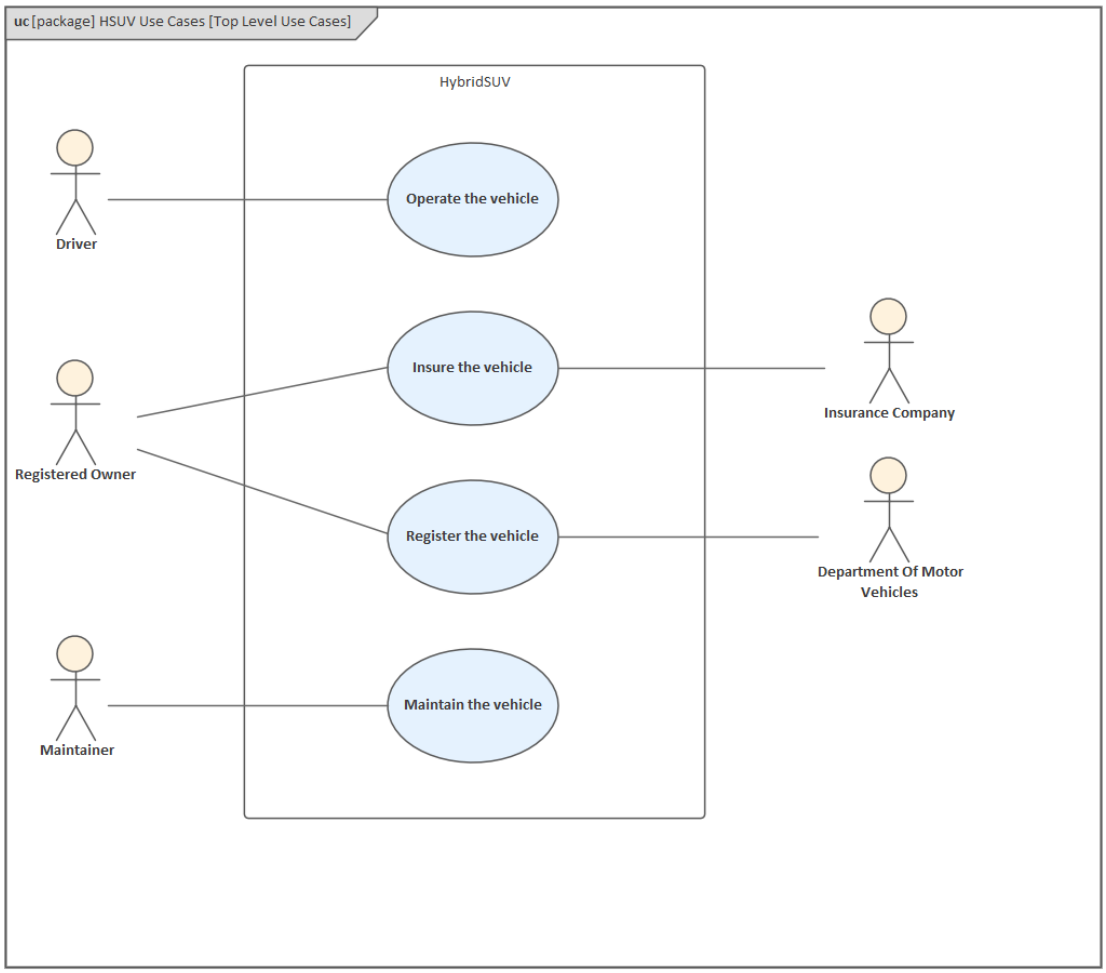


This change of element appearance can be applied at a (default) global level or at a diagram specific level allowing alternative presentations to be created for different audiences.

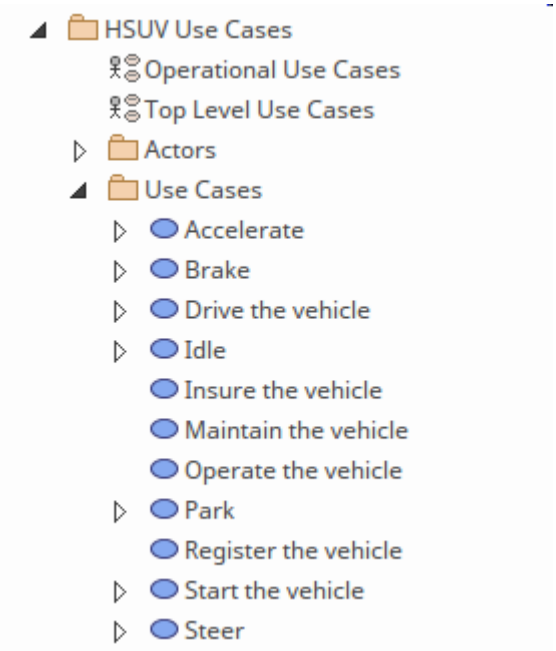

x5:
HSUV:
Maintainer
Default Appearance... F4
Select Default Image...
Select Alternate Image... Control+Shift+W
Select Image Asset as Alternate Image
Hide Name Under Image
Set Font...
Copy Appearance to Painter
Clone Element as New Version...
Find
Copy / Paste
Appearance
Behavior
Lock Element...
Z-Order
UML Help
Delete 'weather: Weather' Control+D
Baggage
weather:
Weather

Use Case Diagram - Top Level Use Cases

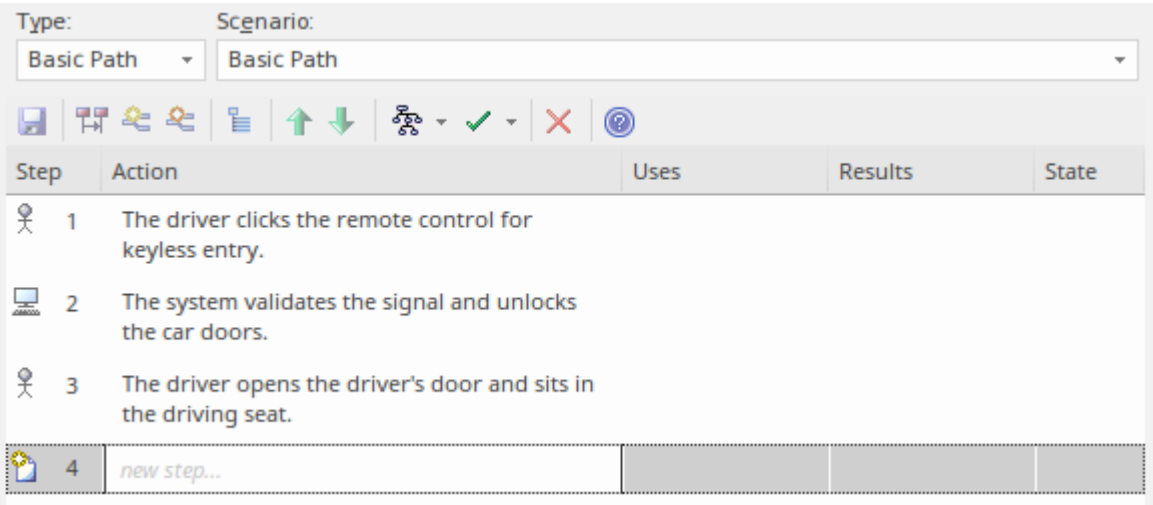
The Use Case diagram describes the HybridSUV as a system that is utilized by a number of external roles. The diagram helps to set the context of all the roles (human and other systems) who interact with or get value from the system. The Use Case diagram, while appearing simplistic, provides a mechanism to ensure that all potential system interactions are defined and understood. The system itself is represented in the diagram by a Boundary element, which acts as container for the Use Cases, with the Actors lying outside the Boundary. In this diagram there are a number of external roles - other than the Driver - who will interact with the HybridSUV system, including the Registered Owner, Maintainer, Insurance Company and Department of Motor Vehicles.



The Use Cases appear in the Browser window and can be conveniently grouped into Actors and Use Cases. Any number of Use Case diagrams can then be defined that allow the Systems Engineer to visualize the Use Cases.



Enterprise Architect also provides a number of helpful and unique tools to assist the Systems Engineer to efficiently describe the Use Cases and define Scenarios that detail the steps representing the interaction between the Actor and the System. Once these have been defined the tool can automatically generate behavioral diagrams directly from the model.

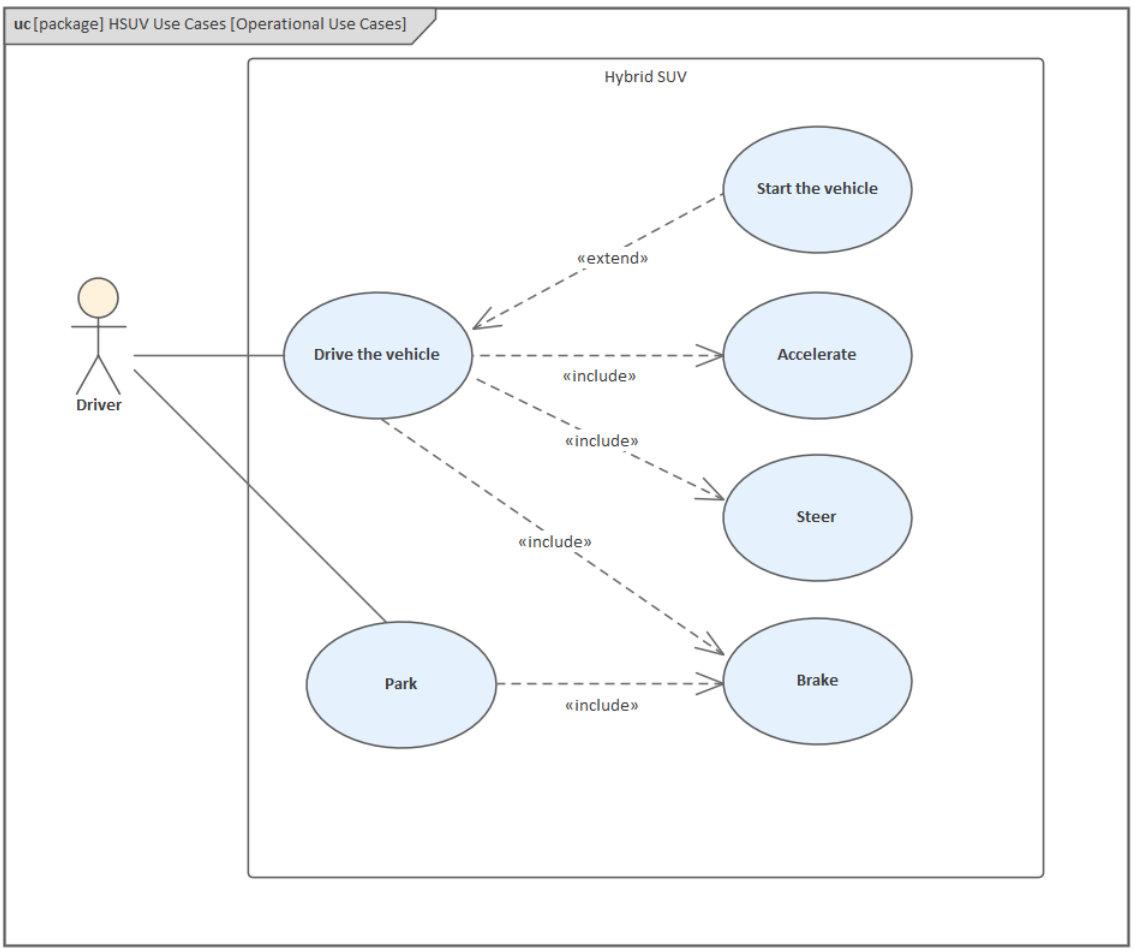


Once the steps have been generated as model elements then the traceability can be added:

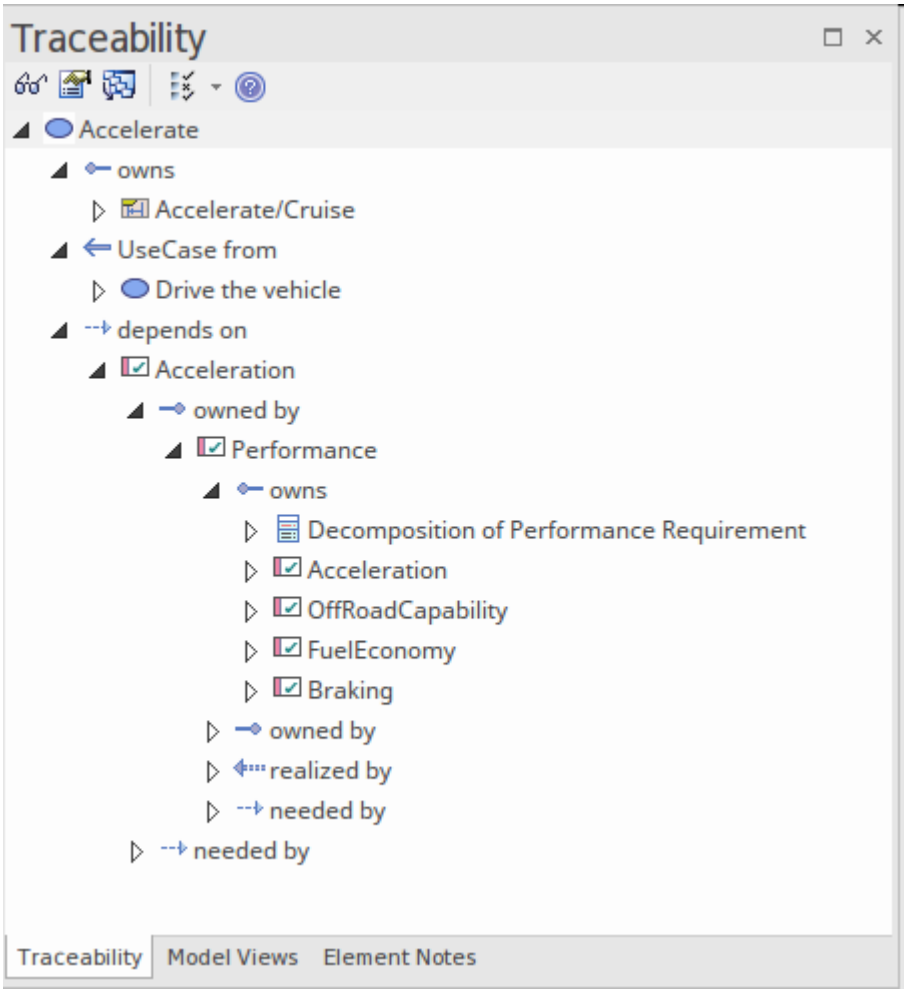
- Actor steps can be traced to Human Machine Interface Models
- System steps can be traced to Component Models.

Use Case Diagram - Operational Use Cases

Goal-level Use Cases associated with 'Operate the Vehicle' are depicted in the next diagram. These Use Cases help flesh out the specific kinds of goals associated with driving and parking the vehicle. The diagram focuses on the Driver of the vehicle as the central Actor. Higher level Use Cases such as maintenance, registration, and insurance of the vehicle are defined under a separate set of context Use Cases.

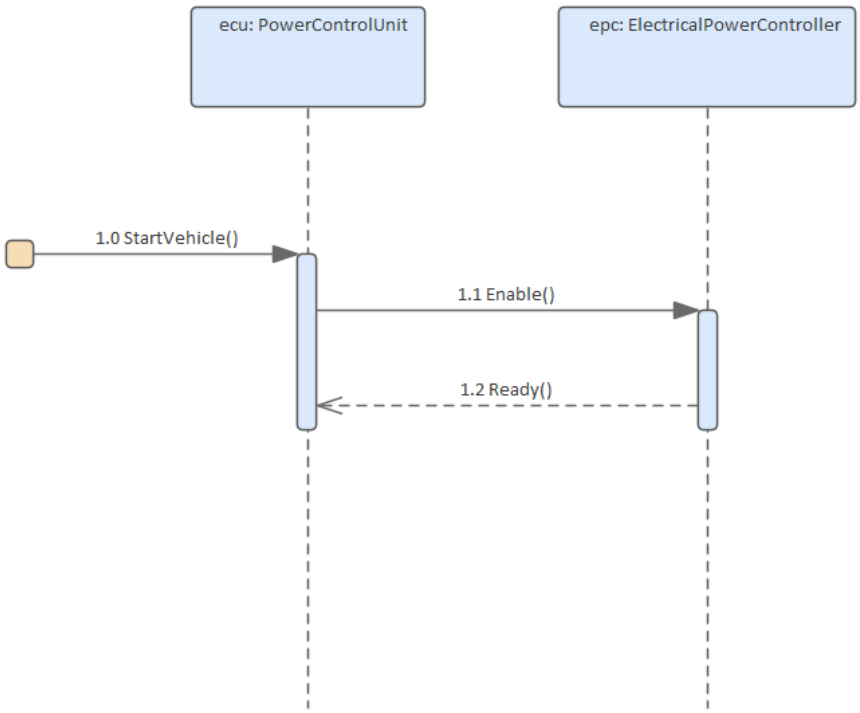


In addition to the extensive features described in the previous section for articulating the scenarios and their detailed steps, there are a number of other features provide by the tool that assist the System Engineer while working with Use Cases. One of the most useful of these features is the Traceability window, which provides a compelling visualization of what a given Use Case is connected to and in turn what the connected element is related to. As different Use Cases (or any other elements) are selected in the Browser window or a diagram, the window refreshes to show the selected element's connections.


Traceability
Accelerate
owns
Accelerate/Cruise
UseCase from
Drive the vehicle
depends on
Acceleration
owned by
Performance
owns
Decomposition of Performance Requirement
Acceleration
OffRoadCapability
FuelEconomy
Braking
owned by
realized by
needed by
needed by
Traceability
Model Views
Element Notes

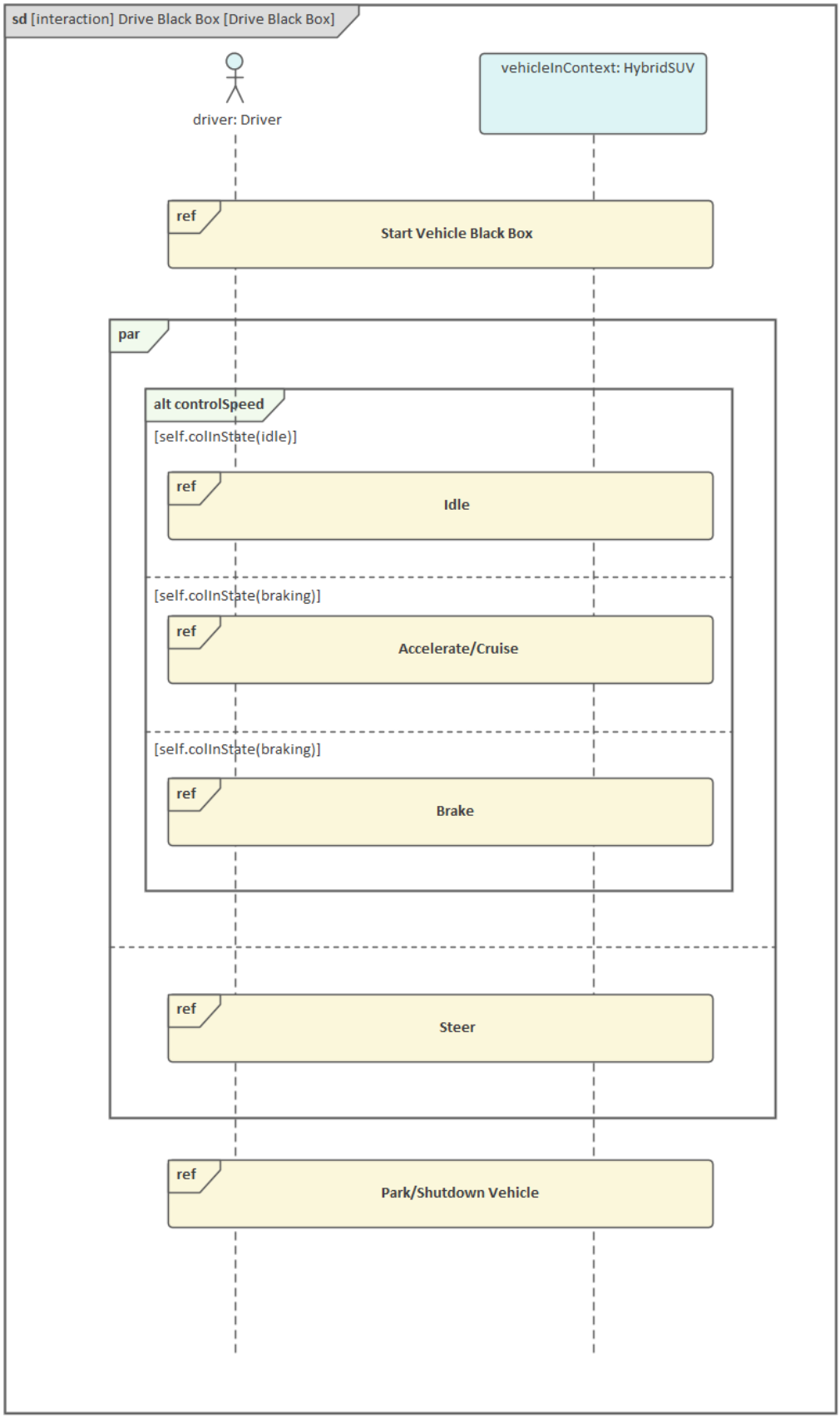
Elaborating Behavior (Sequence and StateMachine Diagrams)

These topics and sections show how behavior can be represented and elaborated using Sequence and StateMachine diagrams.

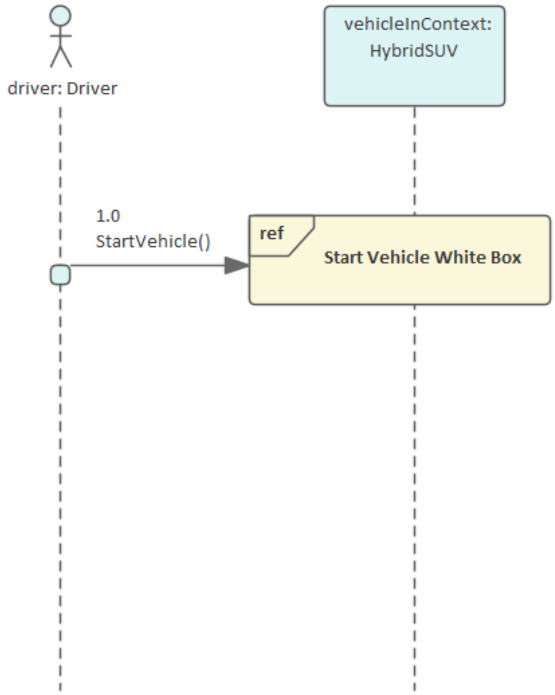


Sequence Diagram - Drive Black Box

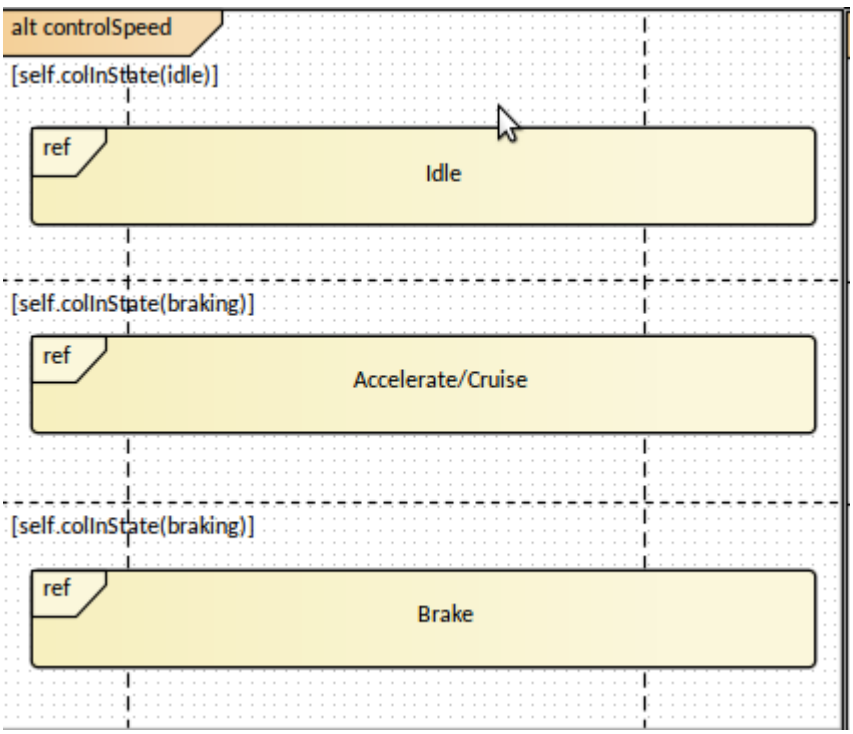
The Sequence diagram is a type of Interaction diagram and describes the way the Driver interacts with the vehicle in a particular context. The elements that participate are, by convention, listed from left to right on the horizontal access of the diagram, and time proceeds vertically in the diagram. The dotted lines that emanate from the objects are called lifelines and represent the time the elements are in existence.



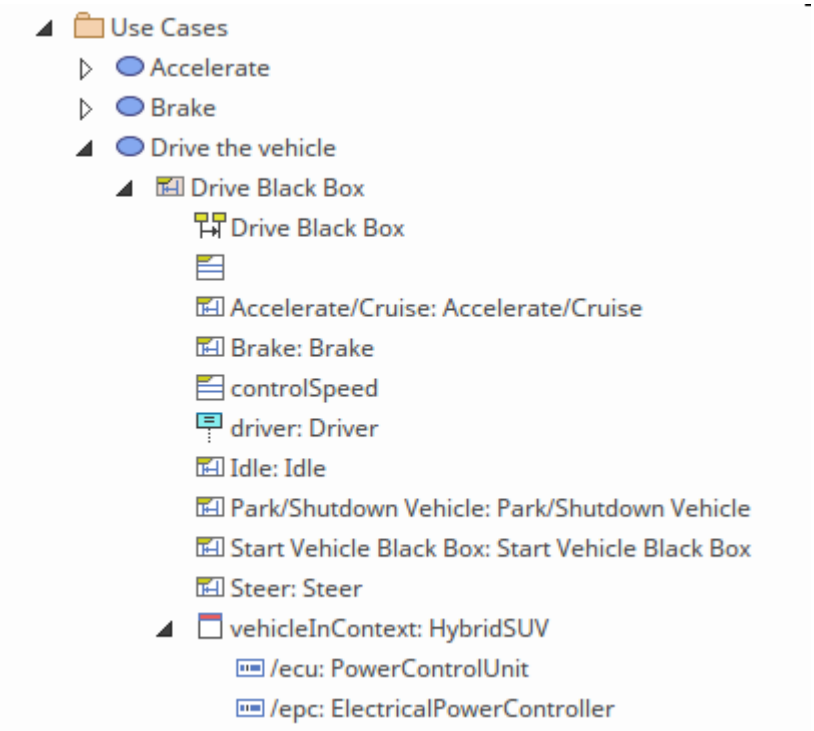
This diagram uses a type of element called a Reference, which acts as a place holder for another Sequence Diagram. Enterprise Architect conveniently allows these diagrams to be opened by double-clicking on the 'ref' element in the diagram.



The diagram utilizes a Parallel (par) Combined Fragment to specify that the Control Speed and Steering interactions occur at the same time (in parallel). The diagram also uses a Combined Fragment with a designation of Alternative (alt) which specifies a number of (alternative) ways that the driver can control the speed.

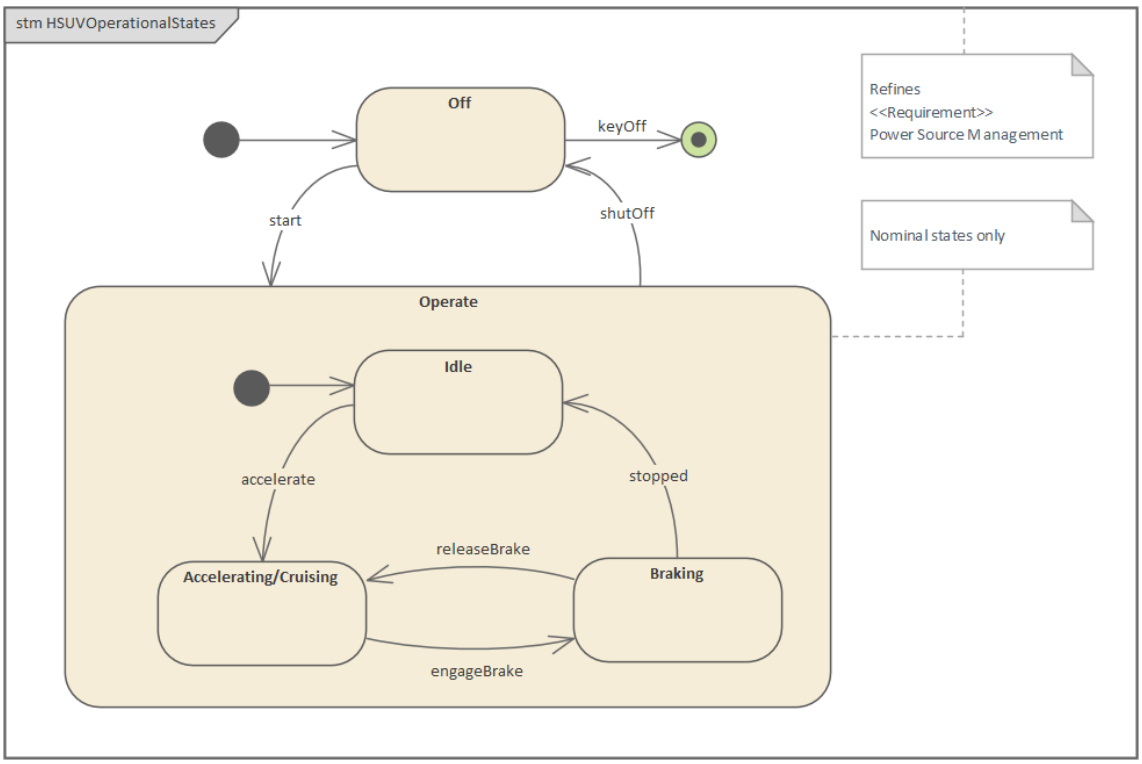


Enterprise Architect allows the Sequence Diagram to be defined as a child of the 'Drive the Vehicle' Use Case making it easy for a modeler to access the diagram and view it in the context of the goal the Driver is wanting to achieve with respect to the vehicle.

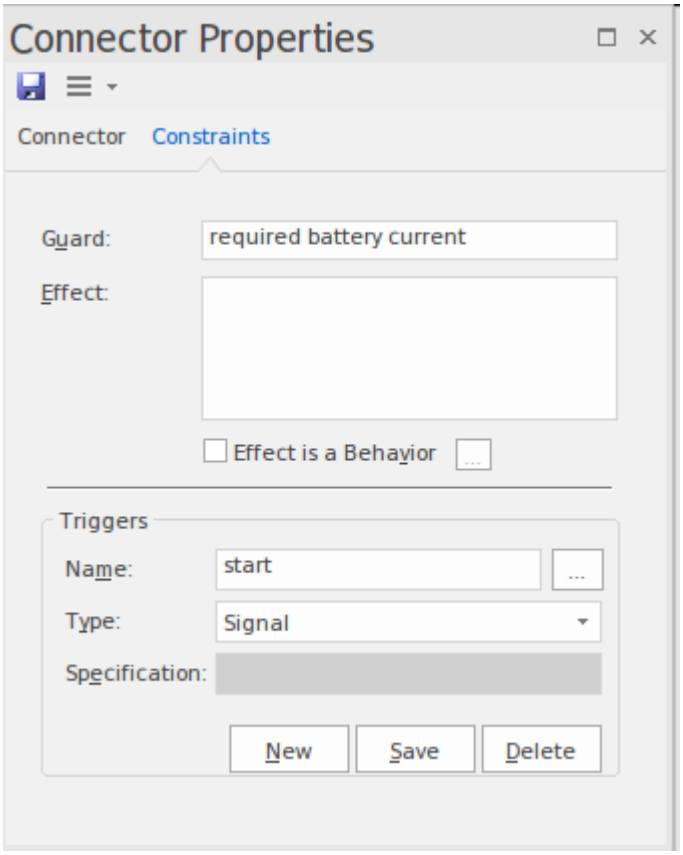

Use Cases
Accelerate
Brake
Drive the vehicle
Drive Black Box
Drive Black Box
Accelerate/Cruise: Accelerate/Cruise
Brake: Brake
controlSpeed
driver: Driver
Idle: Idle
Park/Shutdown Vehicle: Park/Shutdown Vehicle
Start Vehicle Black Box: Start Vehicle Black Box
Steer: Steer
vehicleInContext: HybridSUV
/ecu: PowerControlUnit
/epc: ElectricalPowerController

StateMachine Diagram - HSUV Operational States

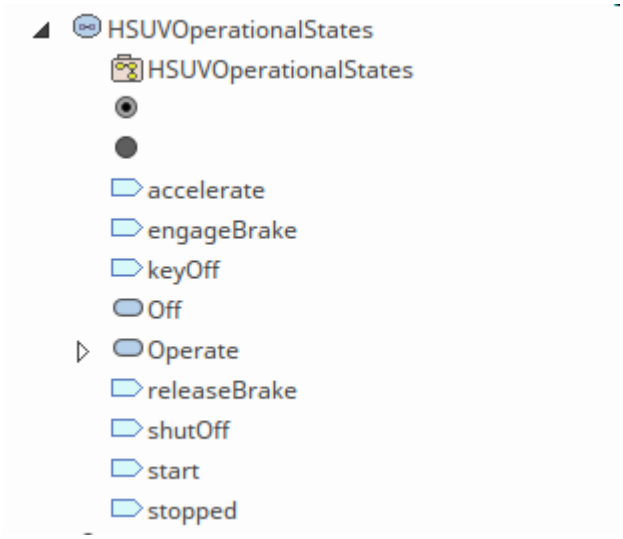
This StateMachine diagram is used to describe the discrete states that the Hybrid SUV exhibits during a specified part of its lifetime. A system or part of a system can have a wide range of states depending on the observer's perspective, so a modeler must always specify the perspective or view that is being used for the diagram. The diagram then articulates the important and relevant conditions within the specified lifetime of the entity.



The message in the Sequence diagram Start Vehicle is the trigger that will cause the vehicle to transition from the Off state to the Operate (on) state. Enterprise Architect allows these transitions to be defined in detail.

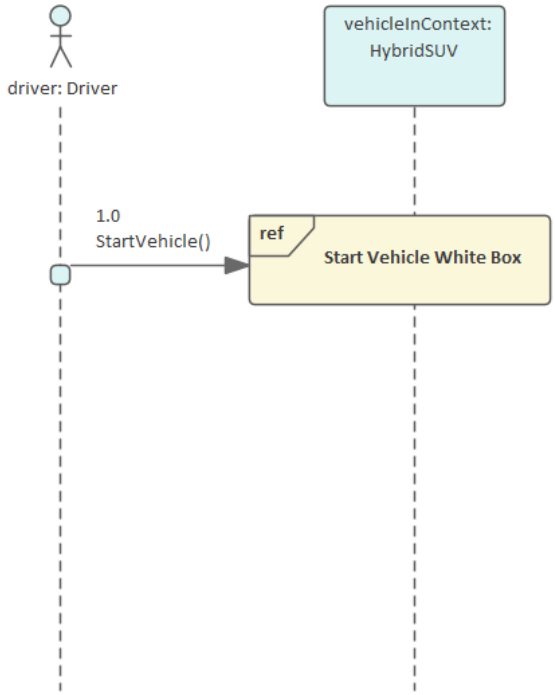


The States appear in the Browser window and are conveniently grouped together under the StateMachine node.

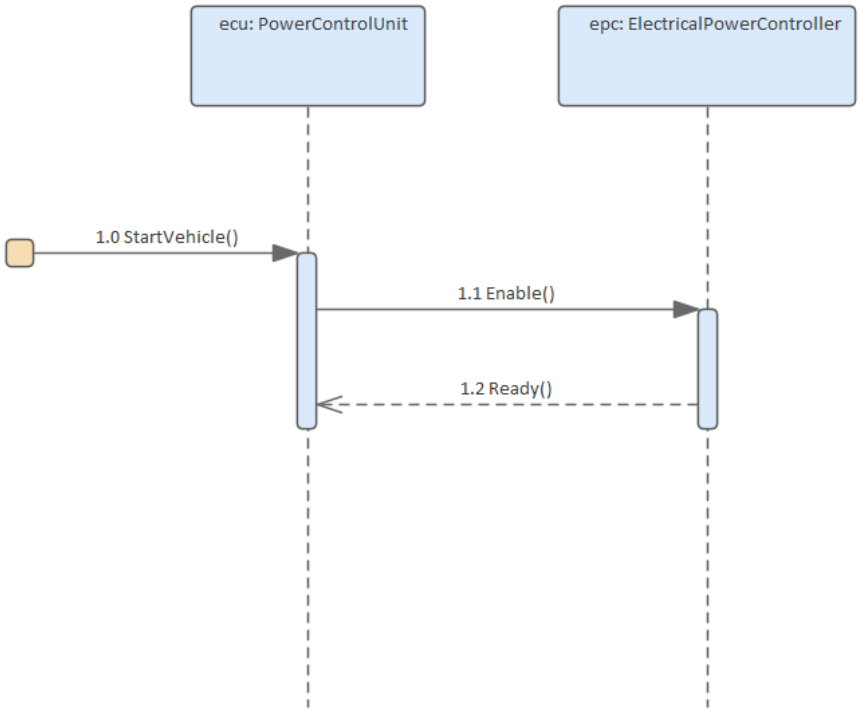


Sequence Diagram - Start Vehicle Black Box and White Box

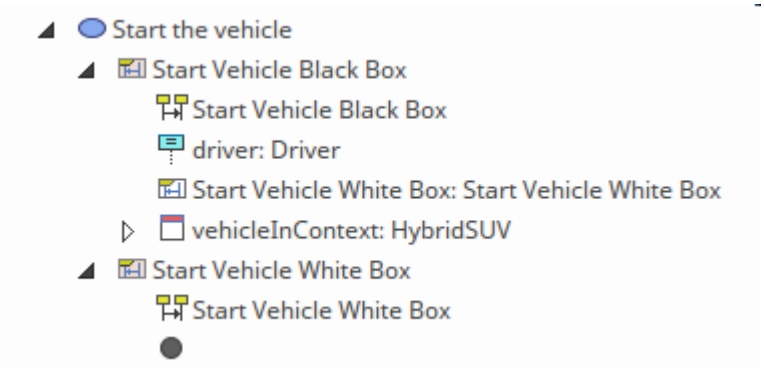
This Sequence diagram describes the interaction of the Driver starting the vehicle. It makes use of the Interaction Use element indicated by the (ref) to make reference to another Sequence diagram. The diagram is intrinsically simple but these diagram can be augmented with a number of other elements and connectors which elaborate the details of other more complex interactions.



Enterprise Architect provides a convenient mechanism allowing the modeler to click-through to the referenced diagram in this case the Start Vehicle White Box diagram.



Elements in Enterprise Architect can appear in multiple diagrams allowing for expressive narratives to be built up in a model and providing a mechanism for modelers to create multiple views of the same element. The elements on a diagram can be located in the Browser window showing their structural relationship to other parts of the model. In this case both the Black and White Box views of the Start Vehicle interaction are located as children of the Start Vehicle Use Case making it easy to relate them to each other.



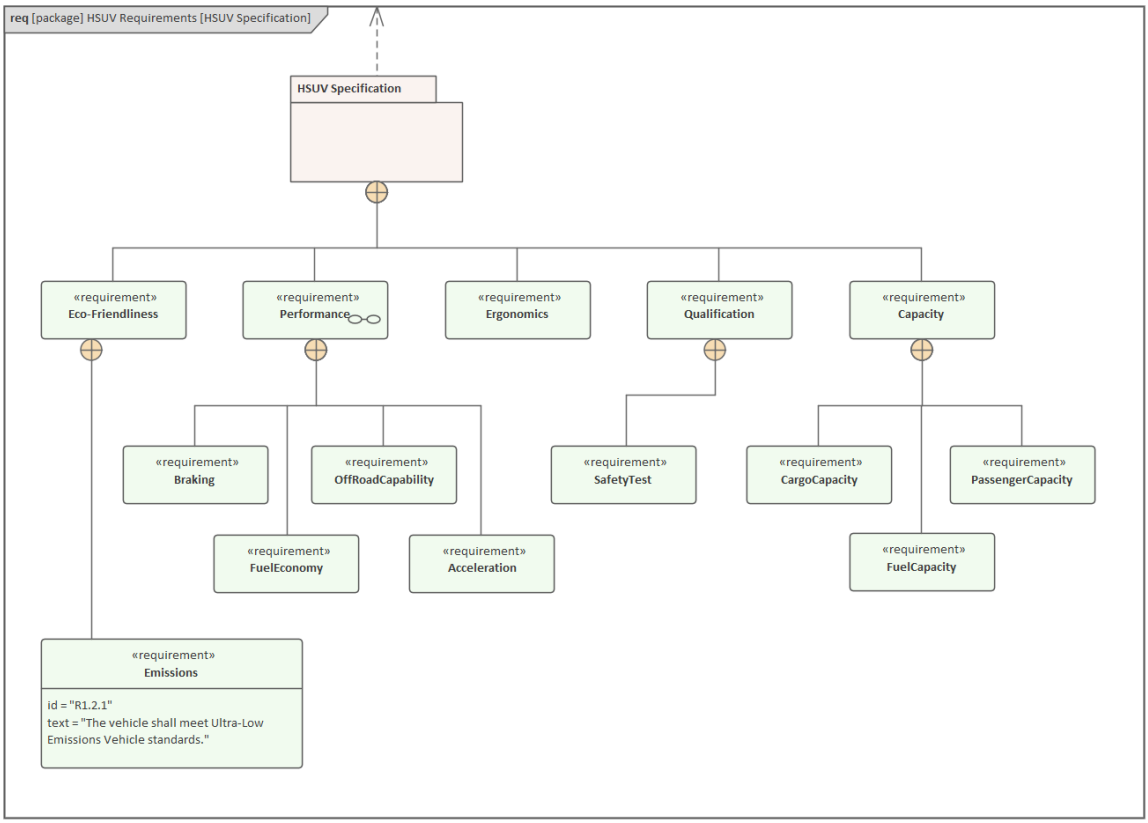
Establishing Requirements (Requirements Diagrams and Tables)

The purpose of this part of the model is to demonstrate the visual modeling of requirements. The vehicle system specification contains many text based requirements which have been recreated in Enterprise Architect. There are a range of requirements that have been modeled including the requirement for the vehicle to pass emissions standards, which is expanded for illustration purposes.

Enterprise Architect provides a wide range of tools for creating, developing, analyzing, managing and testing requirements. It also has integrations to the DOORS requirements management tool.

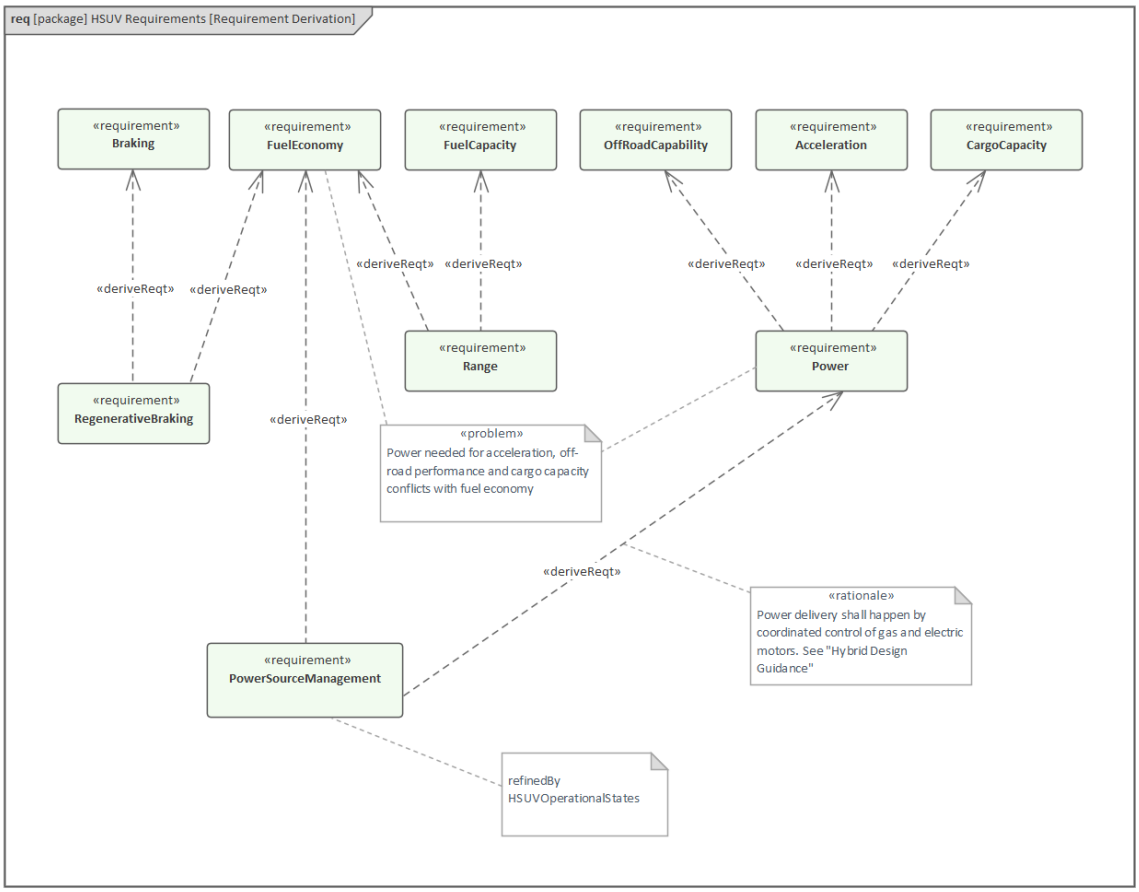
Requirement Diagram - HSUV Requirement Hierarchy

This diagram demonstrates the visualization of requirements in a hierarchy using the containment (cross-hatch) connector to show the parent child relationship. The higher level requirements act as a type of grouping or containment system and cover a range of high level concerns which are broken down into lower level and presumably measurable statements.



Requirement Diagram - Derived Requirements

This requirements diagrams shows that a number of requirements have been derived from other requirements. This relationship implies that some analysis has been conducted and the derived requirement represent a need that can meet the intention of the original requirement. For example Regenerative breaking wasn't a requirement but it has been derived from the need to have both Braking and Fuel Economy as expressed in the requirements at the arrow end of the relationship.



Requirement Diagram - Acceleration Requirement Relationships

This diagram demonstrates a number of requirement relationships which would typically appear on a requirements diagram after some analysis and modeling had been done. The focal element is the Acceleration requirement and the diagram shows how a number of other elements are related to this requirement. The refine relationship is introduced as a way of relating a similarly named Use Case to the Acceleration requirement. We have another derived requirement which is subsequently satisfied by a Block. A 'Max Acceleration' Test Case is also shown on the diagram and related to the central requirement by a Verifies relationship. The diagram also demonstrates that elements other than requirements can be added and help to make the diagram more expressive.

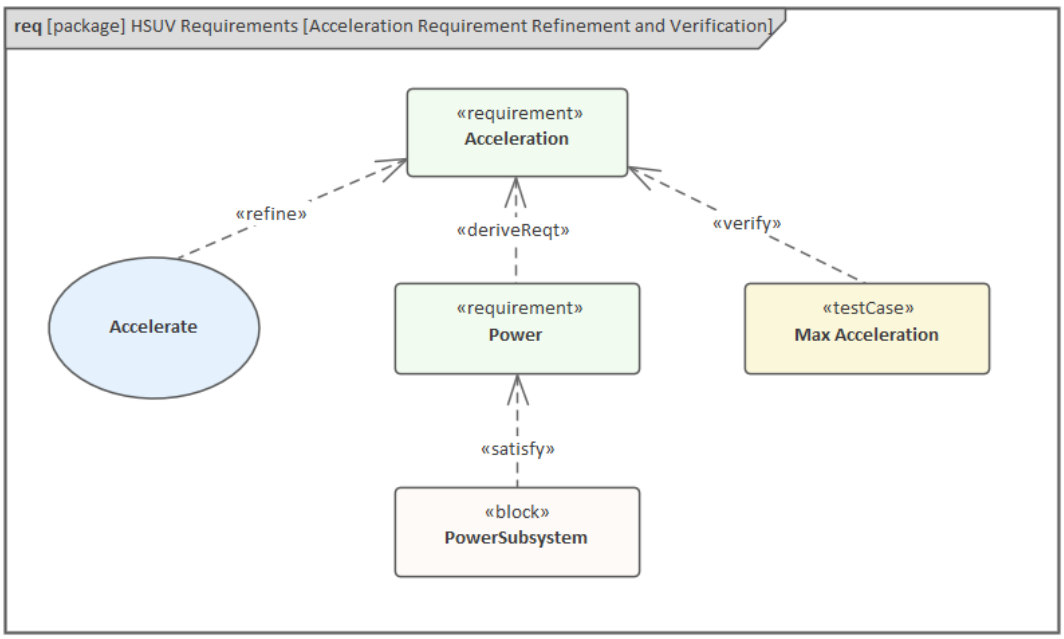
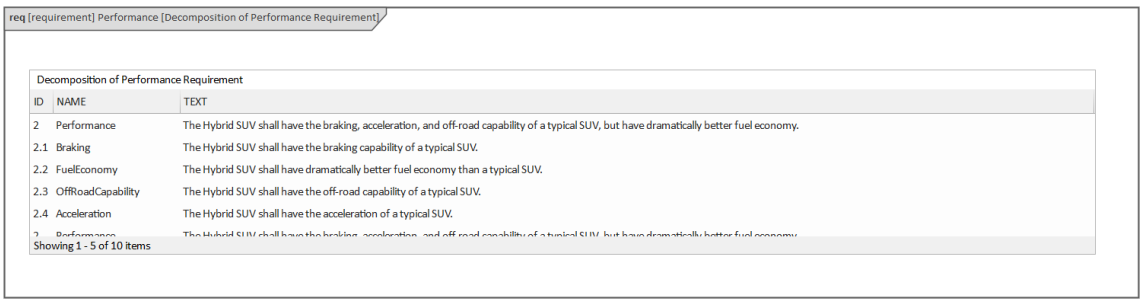


Table - Requirements Table

These examples demonstrate the way that Requirements can be displayed in a tabular form as an alternative to the graphical representation in diagrams. This is a welcomed presentation style for a range of stakeholders who are more accustomed to working with spreadsheets. The first table lists the requirements with their IDs and textual statements. The second table lists the source and target requirements that participate in the derive relationship.

Enterprise Architect also provides a range of tools and ways of visualizing requirements (and other elements) including List Views, Kanban Boards, Specification Views, Gantt Charts, Graphs and more.

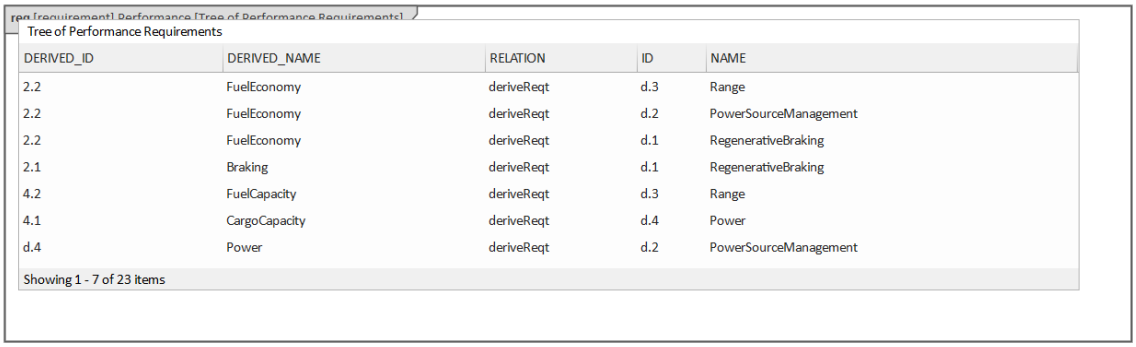


req [requirement] Performance [Decomposition of Performance Requirement]

Decomposition of Performance Requirement

ID	NAME	TEXT
2	Performance	The Hybrid SUV shall have the braking, acceleration, and off-road capability of a typical SUV, but have dramatically better fuel economy.
2.1	Braking	The Hybrid SUV shall have the braking capability of a typical SUV.
2.2	FuelEconomy	The Hybrid SUV shall have dramatically better fuel economy than a typical SUV.
2.3	OffRoadCapability	The Hybrid SUV shall have the off-road capability of a typical SUV.
2.4	Acceleration	The Hybrid SUV shall have the acceleration of a typical SUV.
2	Performance	The Hybrid SUV shall have the braking, acceleration, and off-road capability of a typical SUV, but have dramatically better fuel economy.

Showing 1 - 5 of 10 items



req [requirement] Performance [Tree of Performance Requirements]

Tree of Performance Requirements

DERIVED_ID	DERIVED_NAME	RELATION	ID	NAME
2.2	FuelEconomy	deriveReqt	d.3	Range
2.2	FuelEconomy	deriveReqt	d.2	PowerSourceManagement
2.2	FuelEconomy	deriveReqt	d.1	RegenerativeBraking
2.1	Braking	deriveReqt	d.1	RegenerativeBraking
4.2	FuelCapacity	deriveReqt	d.3	Range
4.1	CargoCapacity	deriveReqt	d.4	Power
d.4	Power	deriveReqt	d.2	PowerSourceManagement

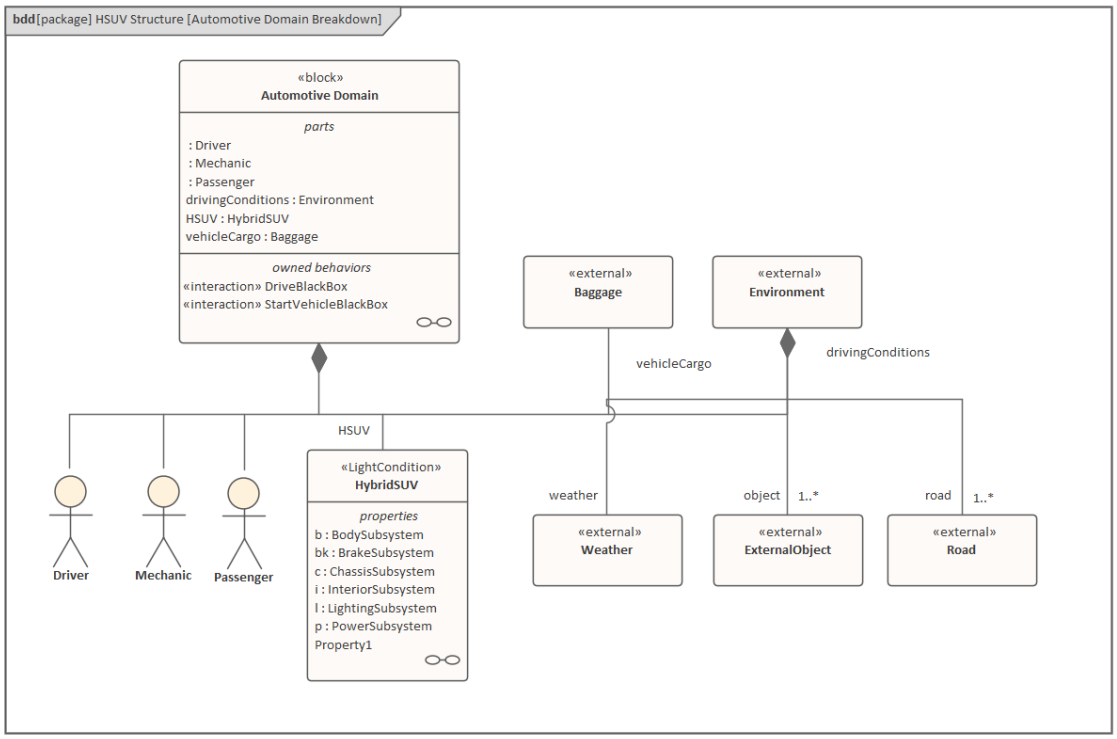
Showing 1 - 7 of 23 items

Breaking Down the Pieces (Block Definition Diagrams, Internal Block Diagrams)

Blocks are the fundamental units of structure and can be used on both Block Definition and Internal Block Diagrams to describe structural aspects of a system. The Block definition diagram is often a starting point for many engineers wanting to gain knowledge of a system and see how it is structured. The Block itself is made up of structure and this is represented by Features, this includes Parts which are themselves typed by other blocks for example a wheel assembly could have a disc caliper Part. There are also Value properties, that are items that have quantity and these represent physical and other measurable dimensions for example a car could have weight and color and have a 0-100 km acceleration time of 5 seconds. There are also interaction points that show the points that a block can interact with its environment.

Block Definition Diagram - Automotive Domain

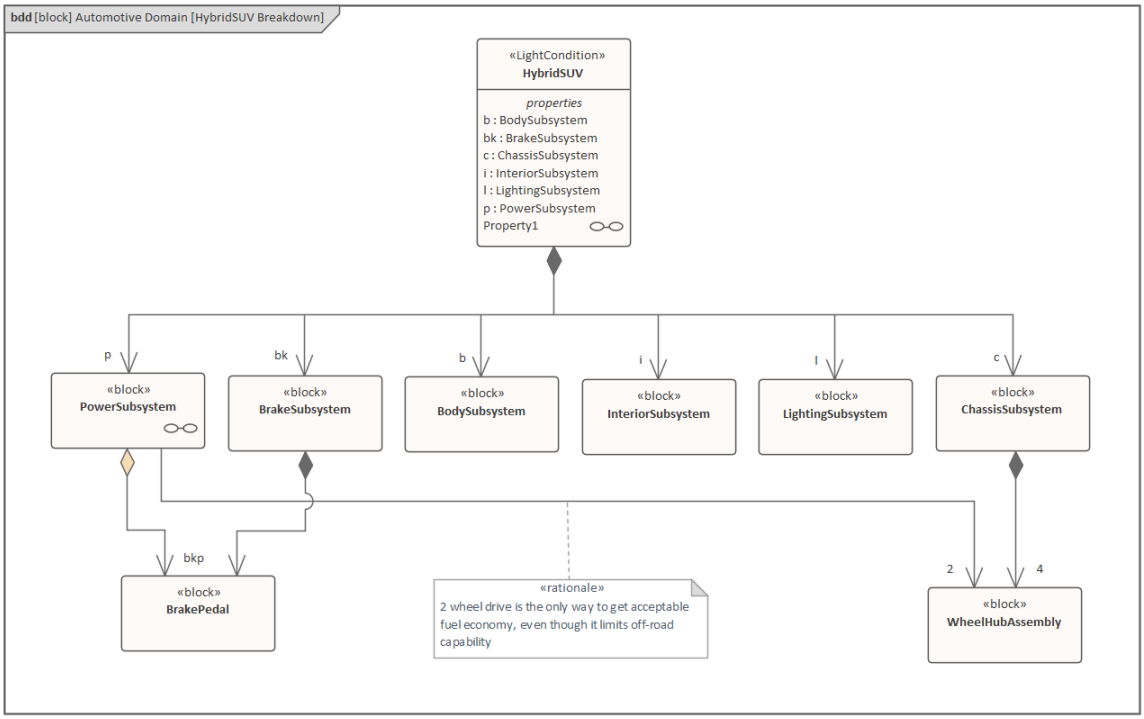
This diagram shows the use of a Block Definition diagram to describe the parts that make up the Automotive domain. The domain includes people who while performing a role will interact with the system including a Driver, Passenger and Mechanic, so there could be a number of specific people performing the role of Driver and the Mechanic when performing a test after repairing the braking subsystem would also shift their role from Mechanic to Driver.



Block Definition Diagram - Hybrid SUV

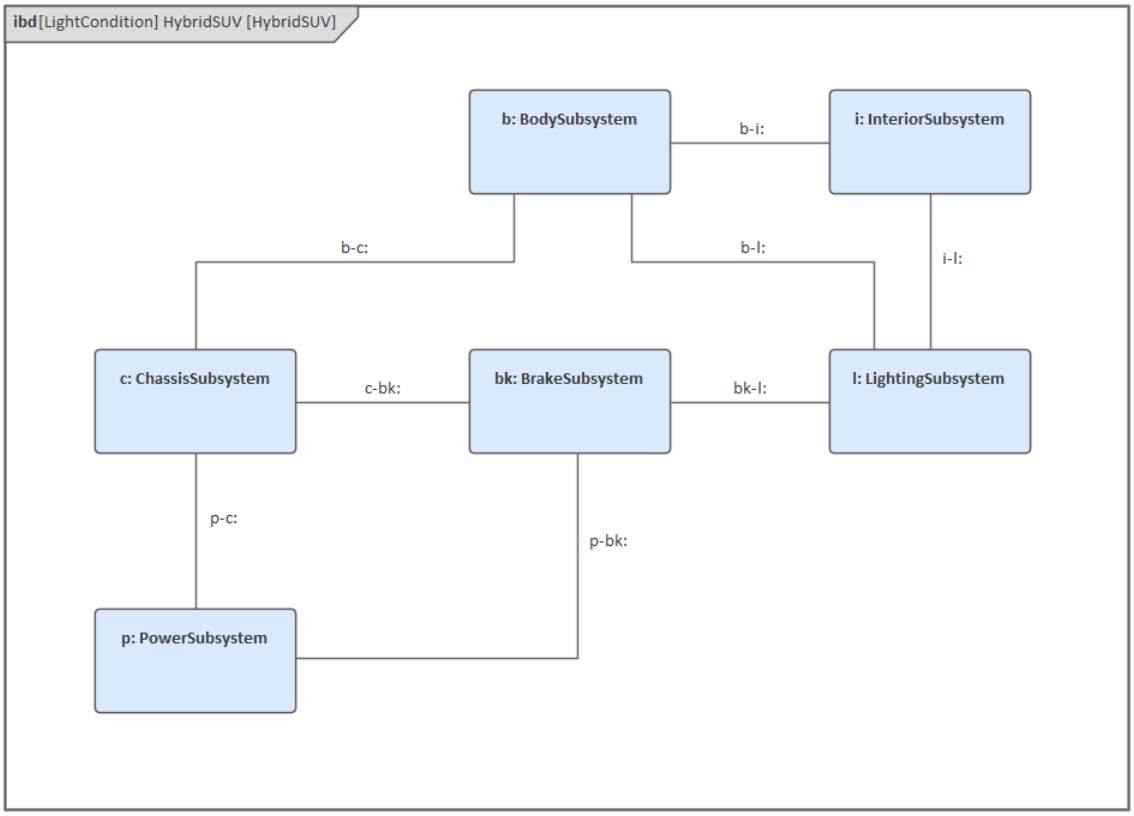
All systems that require engineering analysis and design will be of a level of complexity that will require that the system be broken down into a number of parts to help reduce the complexity and facilitate the project management. The first level of breakdown is most typically called a subsystem and in the case of the Hybrid SUV or for that matter any other automobile the sub-systems would include blocks such as Power, Brake, Lighting and Chassis.

These sub-systems would themselves in turn be broken down into a number of constituent parts for example the Braking Subsystem could be broken down into disc assemblies and hydraulic parts.



Internal Block Diagram - Hybrid SUV

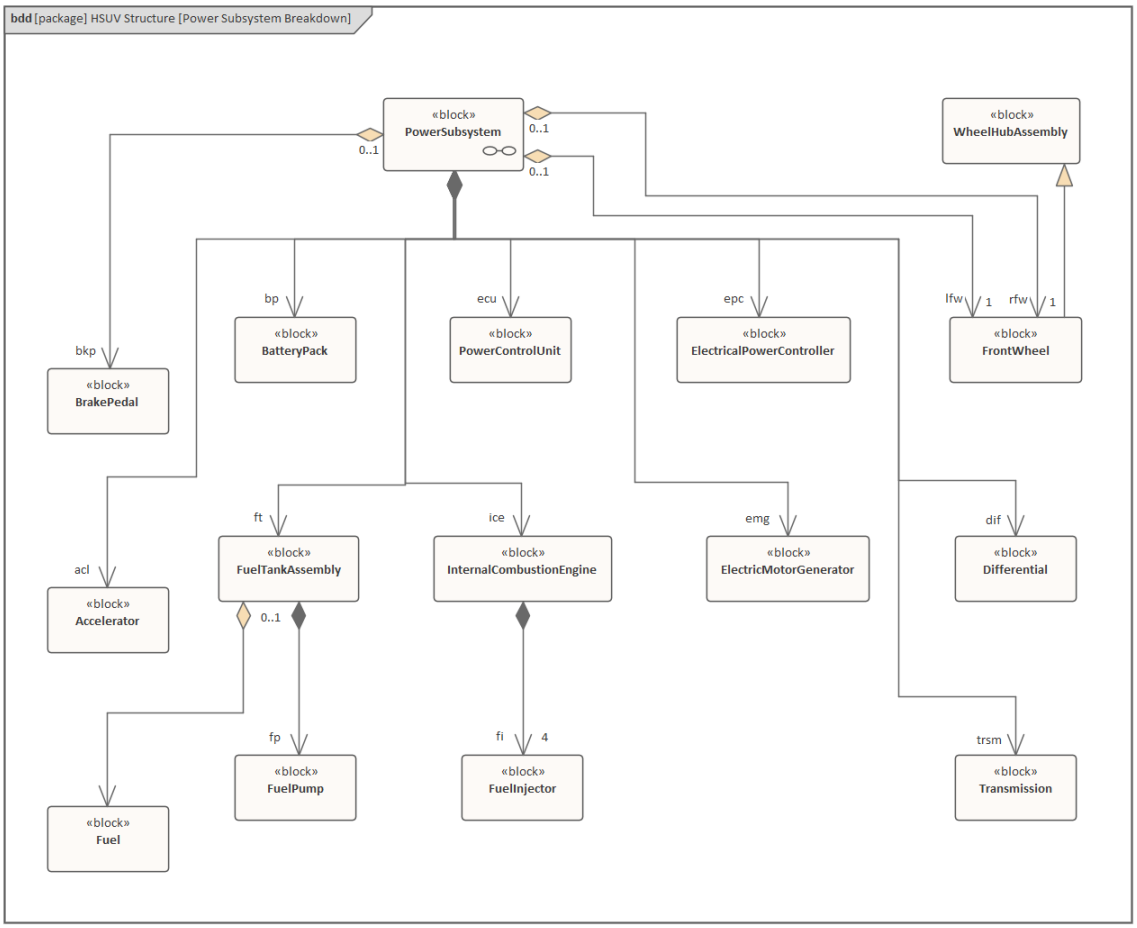
The Internal Block diagram provides a mechanism to describe how the parts are related or connected to each other in the context of the whole or owning block. In our example of the Hybrid SUV we can see a connection between the Power sub-system and the Braking subsystem presumably to model power assisted breaking. So, while the Block Definition diagram show the structure in terms of composition, the Internal Block is able to look inside the block and see how it is 'wired' together.



We will see in a later section how a specialized form of the Internal Block diagram, namely the Parametric diagram is used to model systems of mathematical equations.

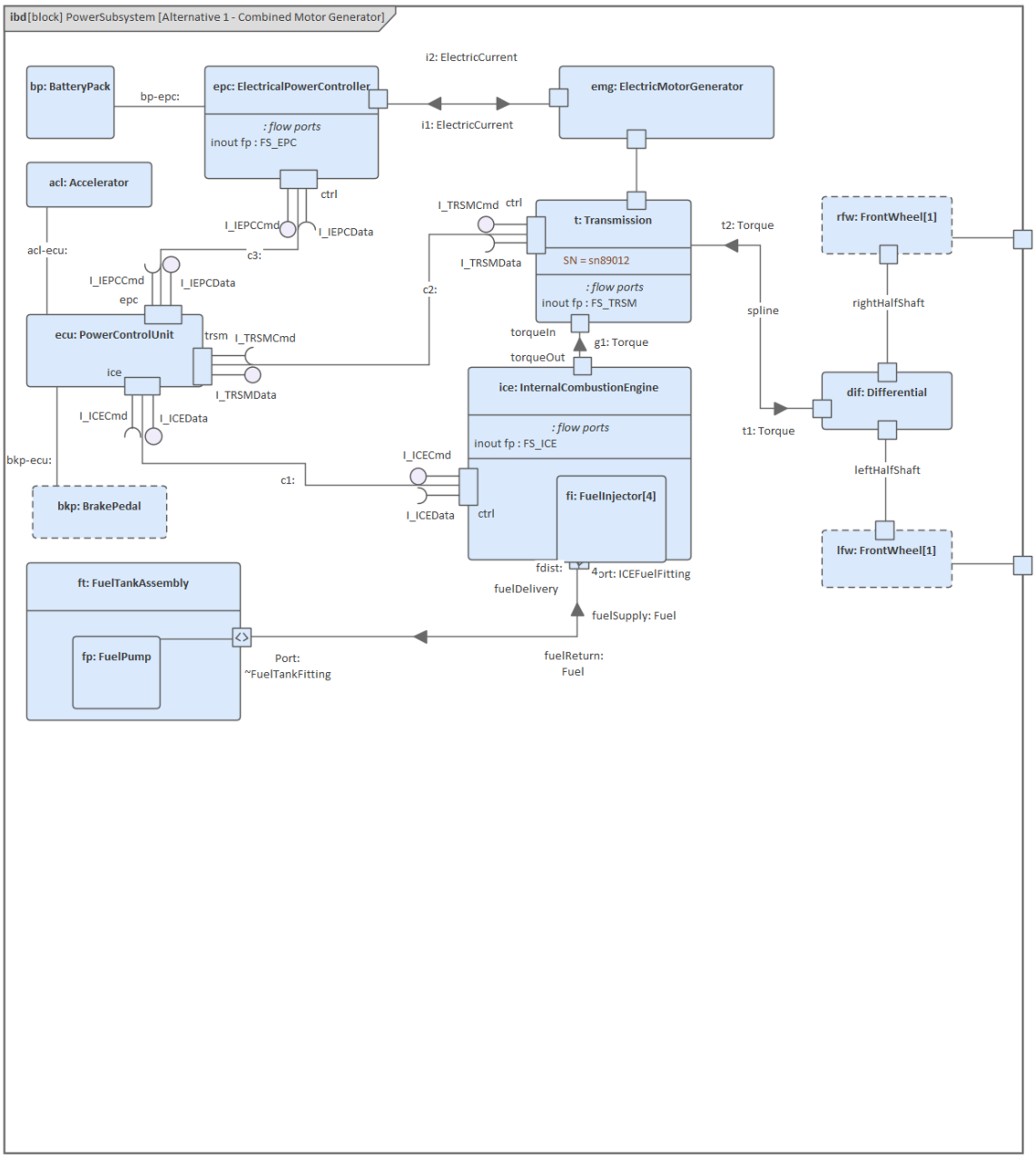
Block Definition Diagram - Power Subsystem

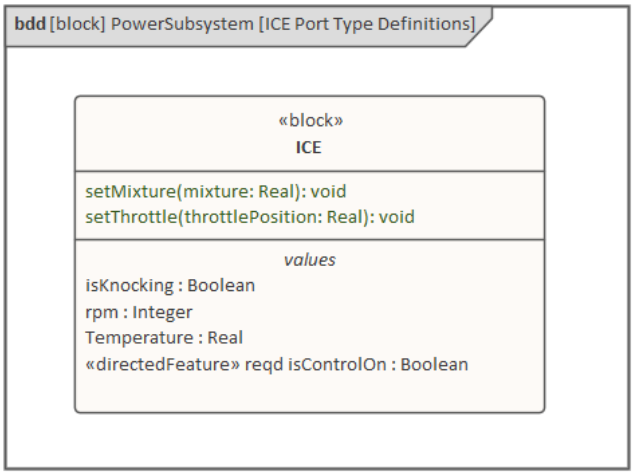
In the previous two sections we have seen how a system can be broken down into a number of sub-systems and that this breakdown can be represented on a Block Definition diagram. We also saw how the subsystems can be related to each in the context of the overall system and how these could be represented on an Internal Block diagram. We will now look at how one of these sub-systems (the Power Subsystem) can be broken down into a number of constituent parts again using the Block definition diagram.



Internal Block Diagram for the Power Subsystem

In an analogous way to how we showed the connections between subsystems, using an Internal Block diagram, we can do the same thing to represent the way that the parts of a subsystem are connected together. So we see again how the two diagram types, Block Definition diagram and an Internal Block diagram, can be used in tandem to describe the structure of a system and how we can move down a part hierarchy to a point where the complexity is understood and does not require further modeling.




bdd [block] PowerSubsystem [ICE Port Type Definitions]
«block»
ICE
setMixture(mixture: Real): void
setThrottle(throttlePosition: Real): void
values
isKnocking : Boolean
rpm : Integer
Temperature : Real
«directedFeature» reqd isControlOn : Boolean

Defining Ports and Flows

The diagrams in the topics of this section show how items that flow can be modeled, using Ports, Flows and Flow Specification on Block Definition, Internal Block and Parametric diagrams. Most physical systems will have items that flow, which can be an important part of how the system works. We could consider a number of examples, including:

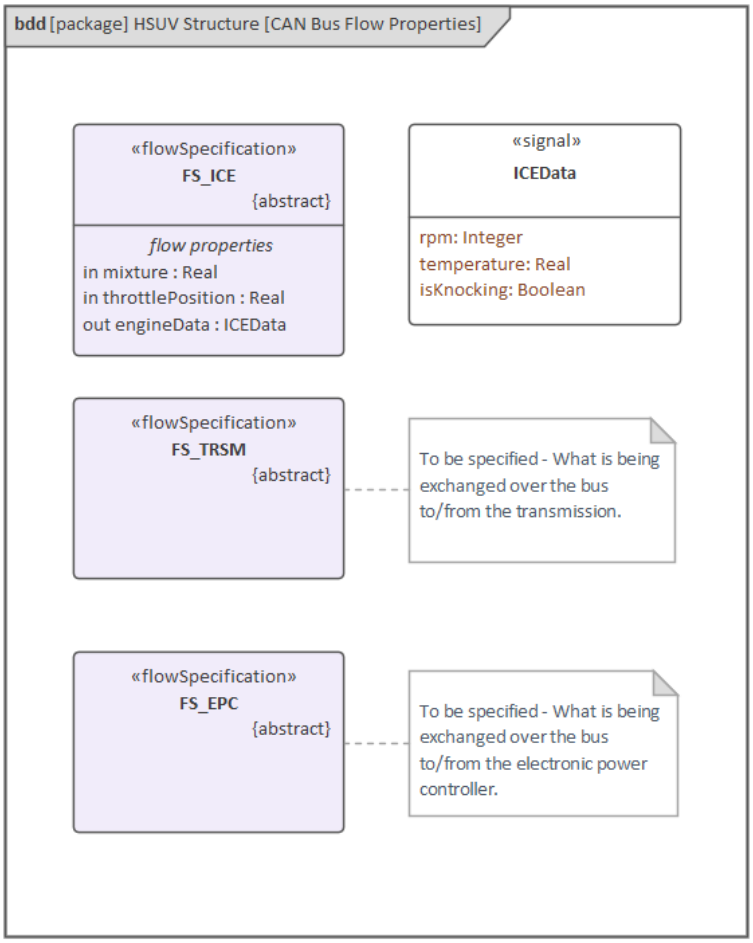
- A desalination plant - where both salt and fresh water and power flow through the system
- A production line where assemblies, parts, power and robot control instructions flow through the system
- An urban transport system where trains, trams, buses, ferries and passengers flow through the system
- An aircraft where fuel, air, control signals, hydraulic fluid, passenger, flight attendants flow through the system

The diagrams start by defining a Controller Area Network (CAN) bus architecture and show how a number of flow specifications can be used to define the way that items flow between parts of the Power Subsystem. The flow of fuel is modeled using a Block Definition diagram that exhibits Flow Ports (deprecated in SysML version 1.5) that show the logical 'conduit' allowing fuel to flow from the Fuel Assembly and the Internal Combustion Engine. Internal Block diagrams take this further, and finally a Parametric diagram is used to show how a mathematical equation for fuel flow rate, defined in a Constraint, can be used to model the equation. Simulated plots are then visualized using Enterprise Architect's simulation capabilities.

Block Definition Diagram - ICE Flow Properties

This diagram shows the first (unfinished) steps in the definition and refinement of a bus architecture. The modeler has used flow specifications to model the way that items will flow though the vehicle; for example, a flow specification has been defined for:

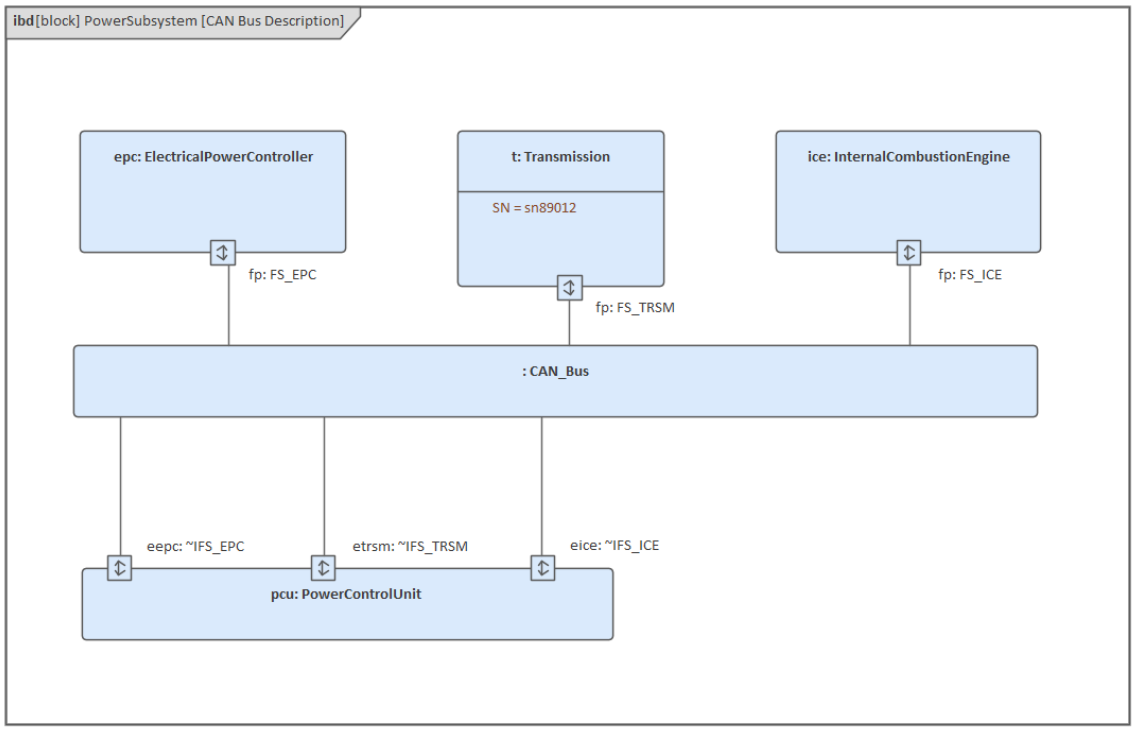
- The Internal Combustion Engine FS_ICE
- The Transmission System
- The Electronic Power Controller



The diagram will be refined in subsequent iterations of the process, and Ports and Flows will be used to model the items that flow through and between the various subsystems.

Internal Block Diagram - CAN Bus

This diagram continues the refinement of the bus architecture, using an Internal Block diagram to show how the various systems are integrated into the Controller Area Network (CAN) Bus. This CAN bus architecture is a central device for controlling and integrating various parts of the Hybrid SUV sub-systems.

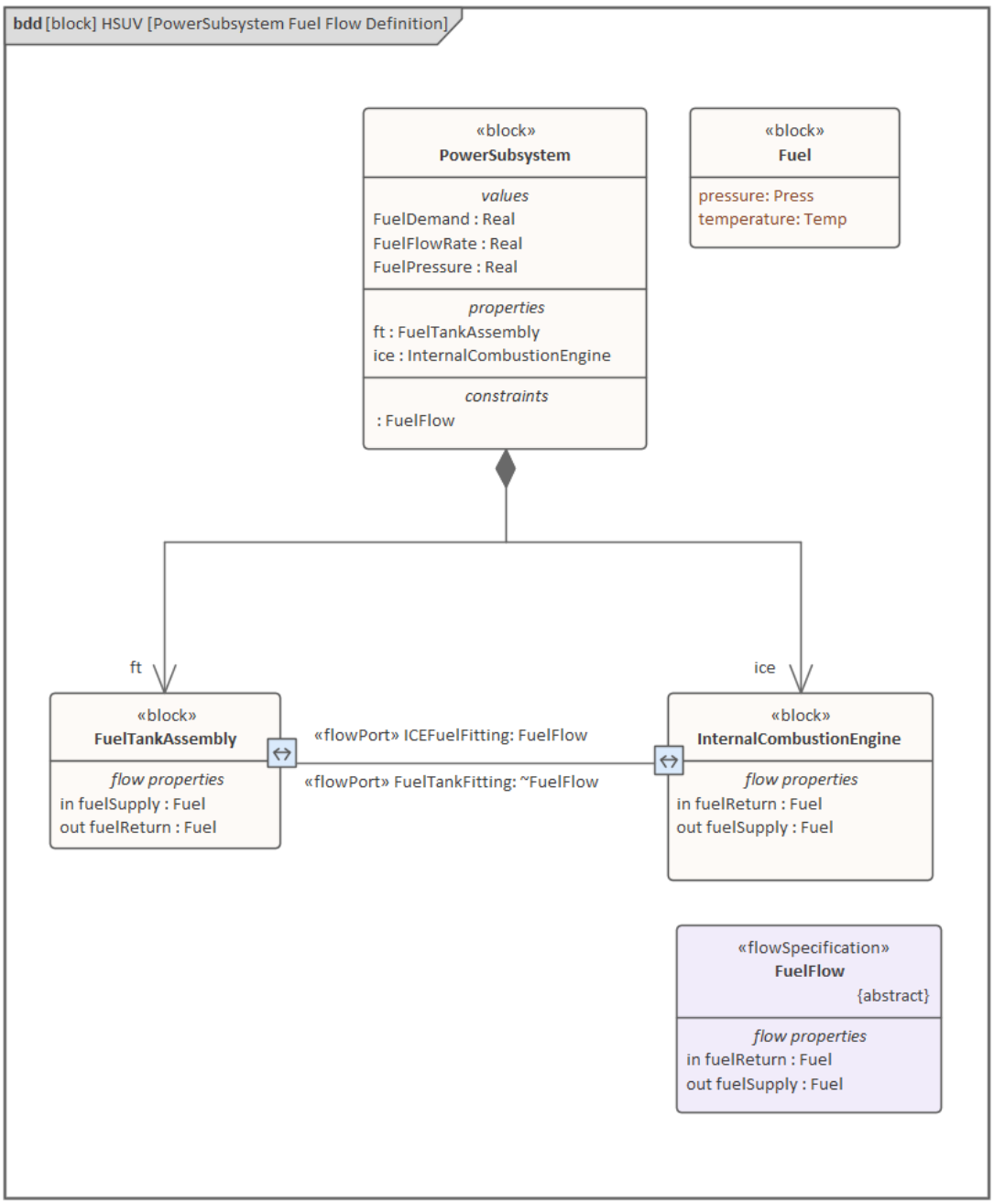


Block Definition Diagram - Fuel Flow Properties

This Block Definition diagram continues modeling how the fuel flows from the Fuel Tank Assembly to the Internal Combustion Engine, elaborating the definition of Fuel Flow. Fuel itself is modeled as a Block and has two value properties that define the important physical characteristics, namely:

- Fuel Temperature
- Fuel Pressure

The Power Subsystem Block is broken down into two of its important parts, namely the Fuel Tank Assembly and the Internal Combustion Engine. The two parts have Flow Ports defined and a connector has been drawn between the two Ports indicating that the item 'Fuel' can flow from the Fuel Tank to the Engine.

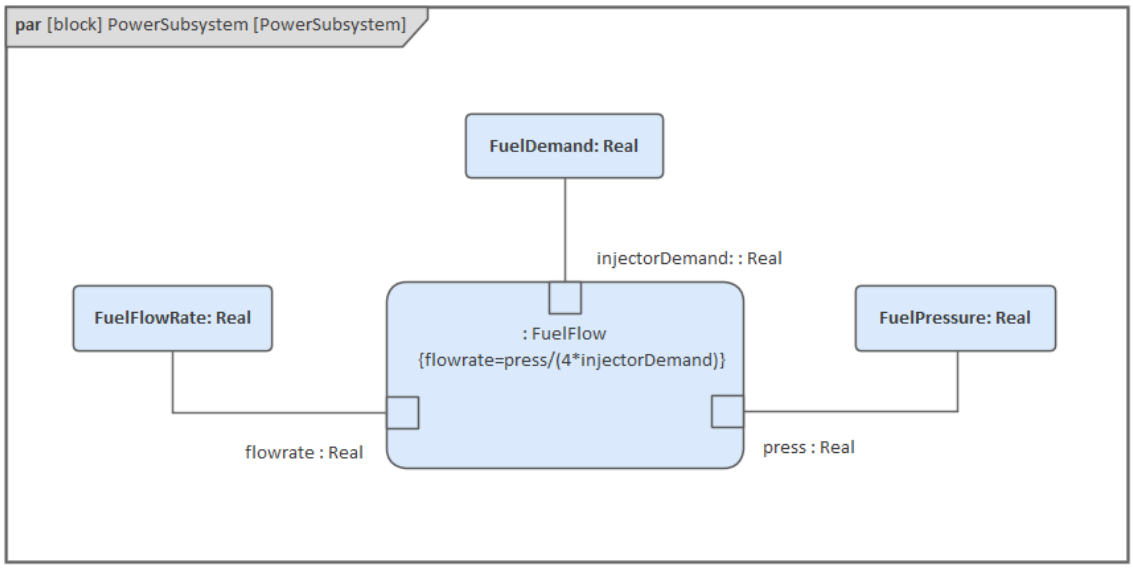


Parametric Diagram - Fuel Flow

This Parametric diagram demonstrates how mathematical equations can be modeled using ConstraintProperties and parameters that are bound to the perimeter of the ConstraintProperty. In this diagram we see that the flow rate is related to both the Fuel Demand and the Fuel Pressure, using the equation:

{flowrate=press/(4*injectorDemand)}

The constraint is modeled in the Constraint Block and can be used in a number of different contexts, using ConstraintProperties on Parametric diagrams. Enterprise Architect has an advanced simulation facility that uses either OpenModelica or Simulink to create plots of modeled equations.



Analyze Performance (Constraint Diagrams, Timing Diagrams, Views)

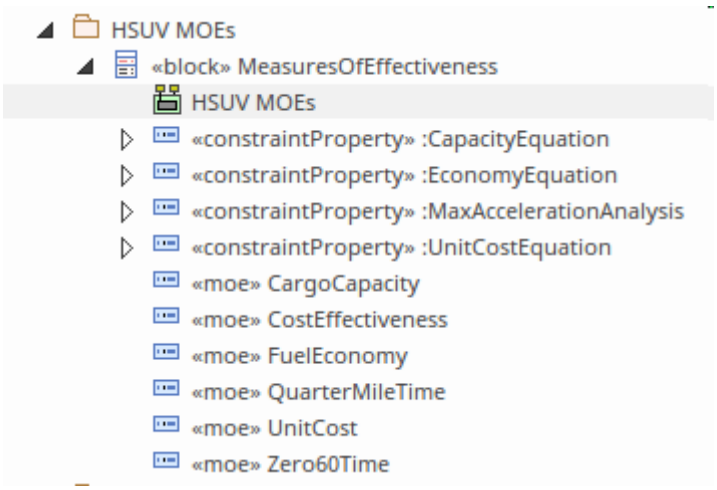
The diagrams in this section of the example are largely Package diagrams that describe Viewpoints and Views used to address stakeholder concerns. The topics also introduce Measures of Effectiveness (MOEs) that can be used in trade studies to evaluate candidate solutions and architectures. The concepts of Viewpoint and View are articulated in ISO-42010 (formerly IEEE-1471) and the SysML Viewpoint and View constructs are consistent with this standard. Typical examples of Views include operational, manufacturing, or security, and these are then related to model elements.

The Viewpoint and View model are best thought of as a narrative or description model, which helps clarify and explain a system model. A Viewpoint and View model exposes elements of one or more system models. More specifically, a Viewpoint is a particular frame from which to view the system models and is a specification of rules for constructing a View to address a set of concerns that are of significance to stakeholders. For example a performance architect will have different concerns to a safety architect. The View is intended to visualize the system from the specified Viewpoint. This provides a mechanism for stakeholders to specify aspects of the system model that are important to them from their Viewpoint, and then represent those aspects of the system in a specific View.

The Viewpoint describes the point of view or lens through which a group of stakeholders look at a system model and, by framing the concerns of the stakeholders along with the method for producing a View, their concerns can be addressed. The method describes:

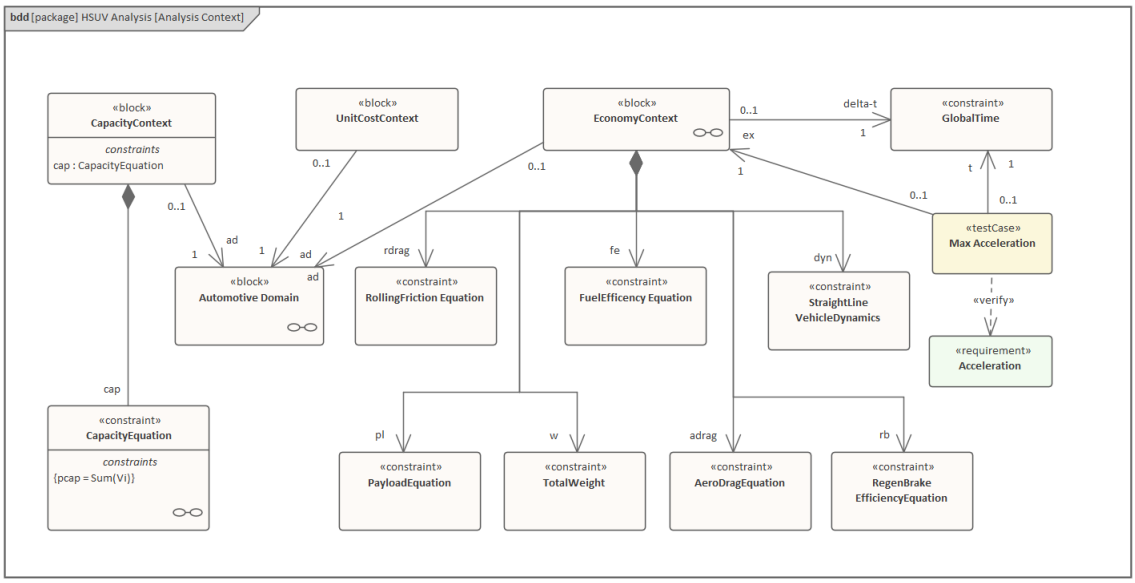
- The expectation of what stakeholder(s) want to see exposed from the model
- How the stakeholder wants the information to be structured and presented
- In what kind of artifact the stakeholder wants to consume the information.

In other words, the process is the set of rules that describe how the View should express the information from the model to address the stakeholder concerns. When the Views and Viewpoints are modeled in Enterprise Architect the relationship to modeling elements can be defined.



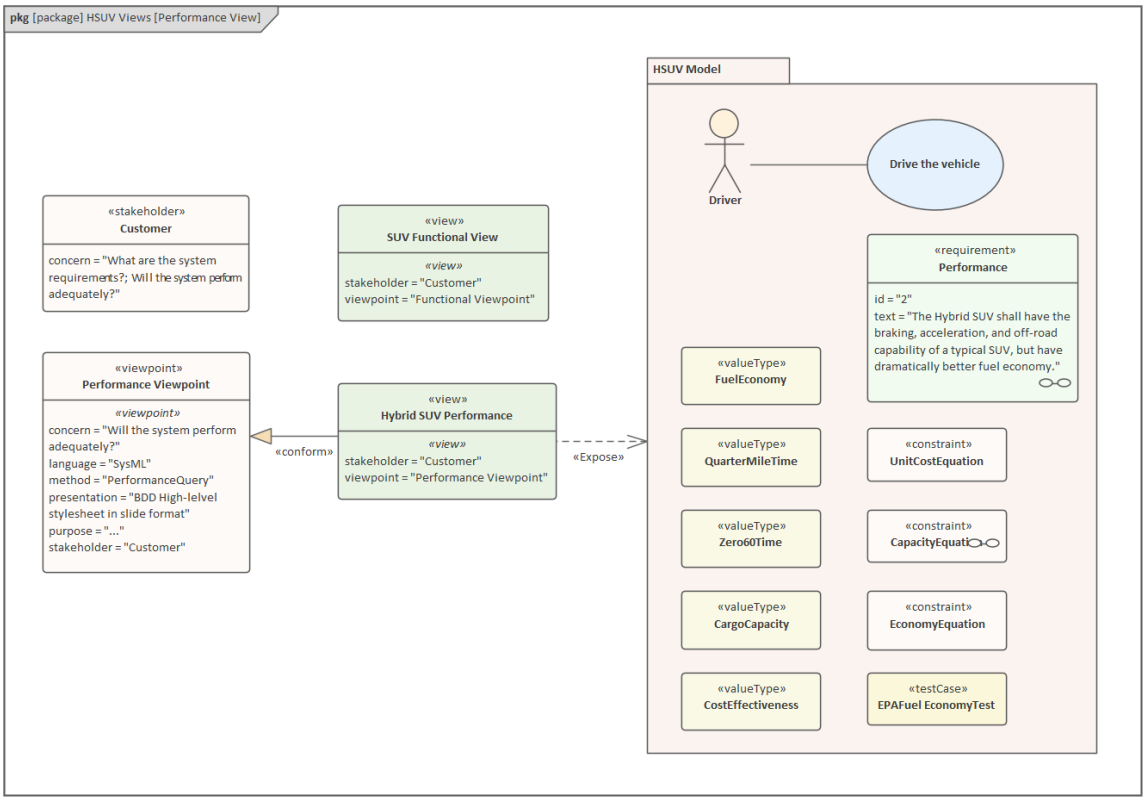
Block Definition Diagram - Analysis Context

One of the key reasons for modeling a system is to be able to perform analysis on the models, which provides a cost effective way to gain insights into how the built system will perform in situ. Performing analysis on a model is cheaper and more convenient than building prototypes. This Block Definition diagram defines the various model elements that will be used to conduct the analysis in this example. It depicts each of the ConstraintBlocks and related equations that will be used for the analysis, and the key relationships between them. There are two types of element present on the diagram - Blocks and Constraints. The diagram also shows the Verify relationship between a Requirement and a Test Case.



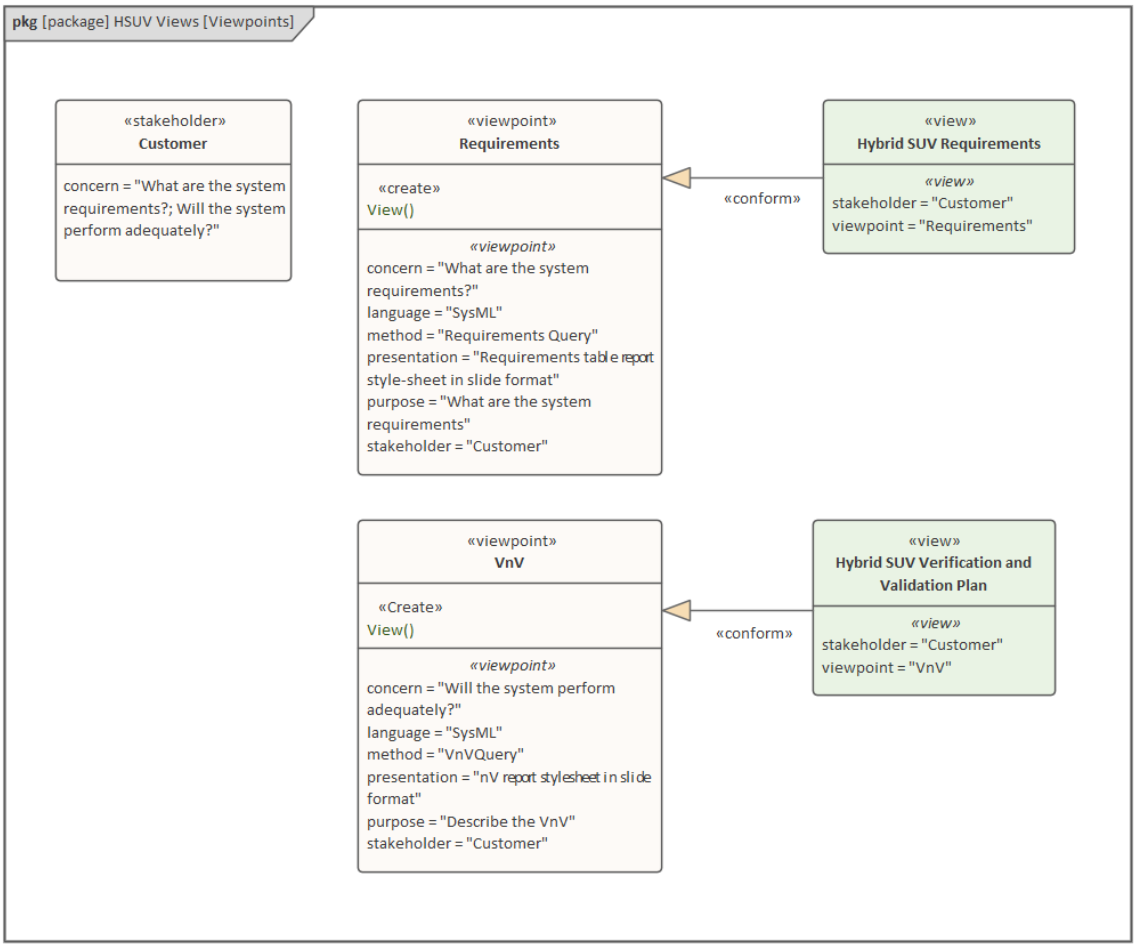
Package Diagram - Performance View Definition

The SysML allows a team to define their own viewpoints; in this example we see a user-defined Performance Viewpoint, and the elements that populate the HSUV-specific Performance View. The Performance View itself might contain a number of diagrams depicting the elements it contains. We can see in the diagram that a number of views have been defined including the Hybrid SUV Performance view and SUV Functional View. Each view has a Stakeholder defined and a View Point. The expose relationship has been used to relate the Performance View to the SUV model and the conform relationship shows that the Performance View is conformant with the Performance Viewpoint.



Package Diagram - Viewpoint Definition

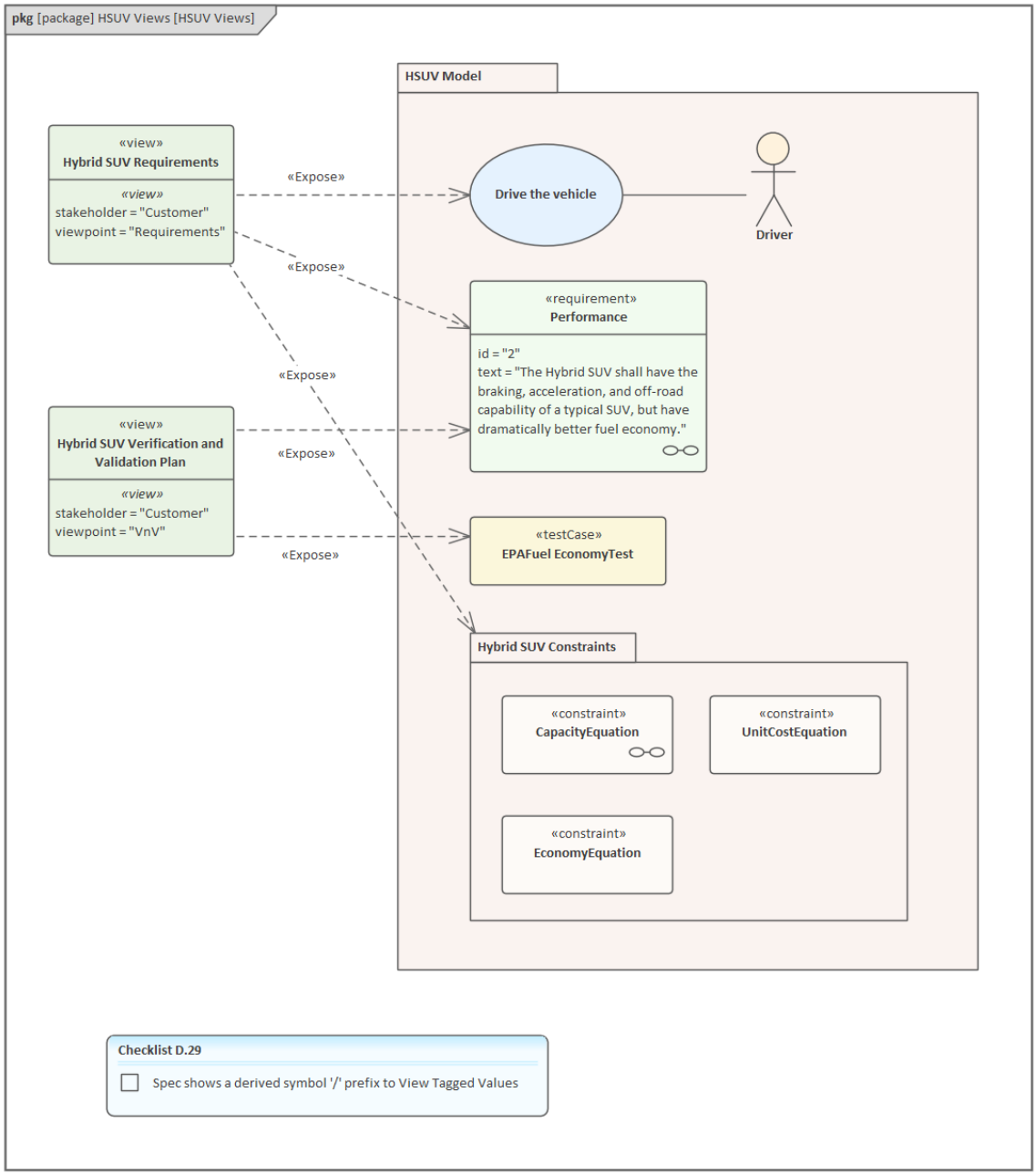
This Package diagram shows the Requirements and Verification and Validation (VnV) viewpoint definitions with relationships to stakeholders, concerns and views. The stakeholder and viewpoint share the same concern via comments that are shown textually as values of the concern property. The comments could be shown graphically with annotation relationships to stakeholders and viewpoints, if needed. Note that the value of the stakeholder property is an instance of the stereotype not the class to which the stereotype is applied.



Package Diagram - View Definition

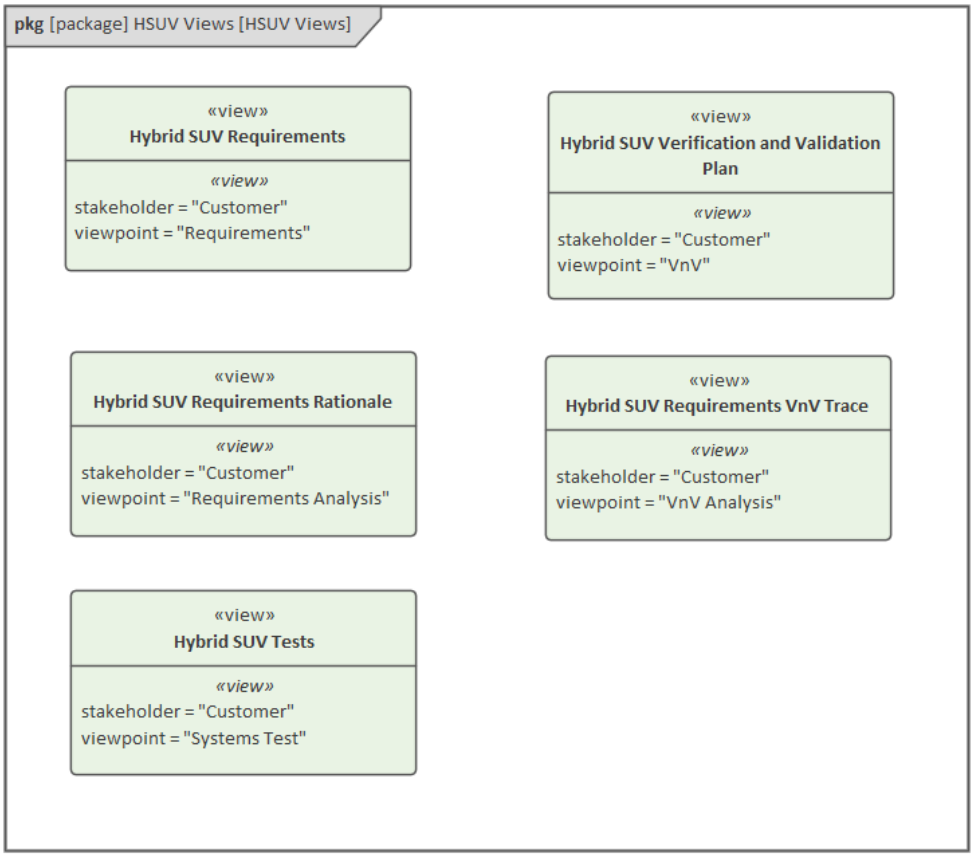
This diagram shows the use of the Expose relationship, which is a graphical device to indicate the elements (including Packages) that are part of the view. The Requirements and the Verification and Validation Views have outgoing Expose relationships that target a number of elements in the model. This provides a useful way of indicating the elements involved in the view; for example, it can be seen from the diagram that the Hybrid SUV Requirements view exposes the Drive Vehicle Use Case, a Performance Requirement and a Package containing a group of SUV constraints.

Using some of the feature rich visualization tools, it would also be possible to visualize which Views a given element participated in, for example the Drive Vehicle Use Case could appear in a number of different Views.



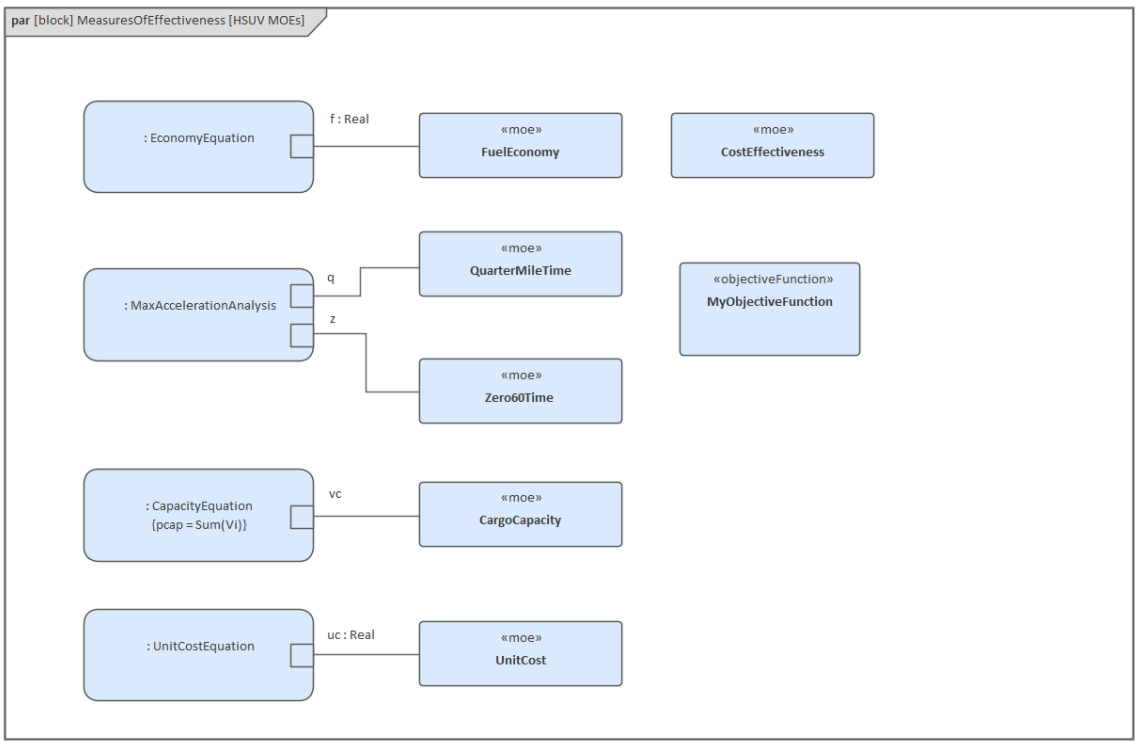
Package Diagram - View Hierarchy

This Package diagram shows how views, or for that matter any other elements with the same stereotype, can be collected into a Package and presented visually.



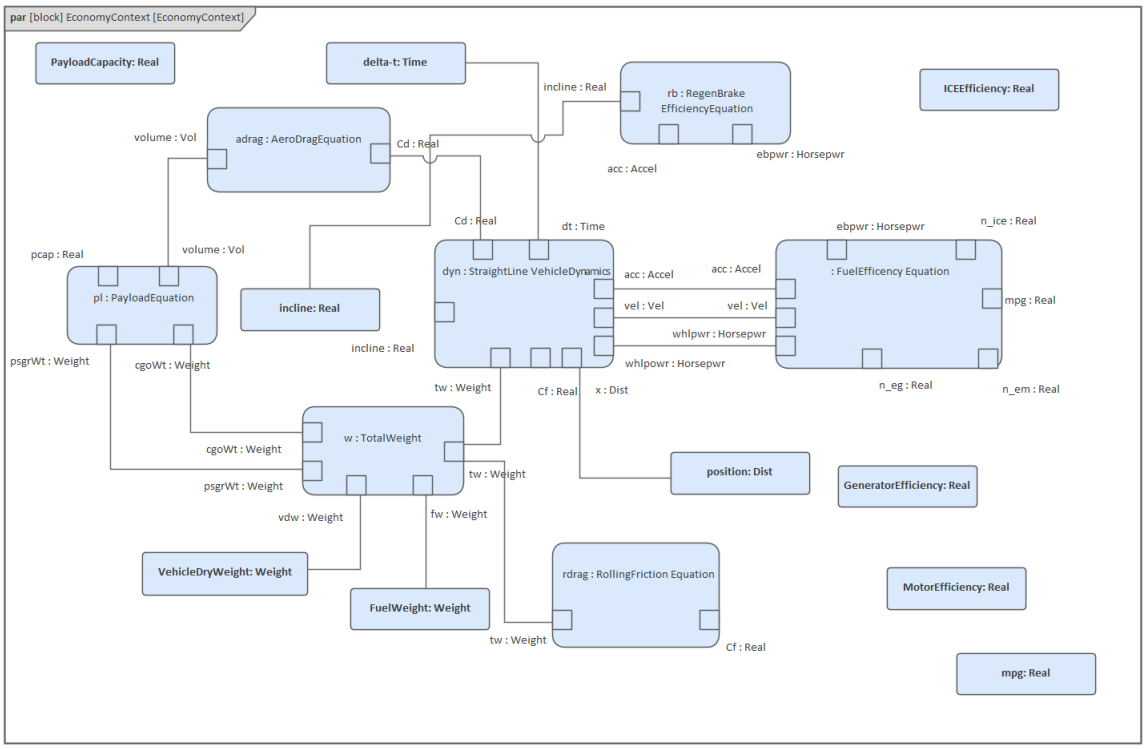
Parametric Diagram - Measures of Effectiveness

Measure of Effectiveness is a mechanism to evaluate a solution by setting up a set of measures that will allow the engineering team to evaluate two or more solutions to a problem. This technique is usually called a trade study and the Measures of Effectiveness (MOEs) are calculated for two or more solutions and compared using a utility (objective) function. The MOE is a user defined stereotype and not formally part of the SysML core language; it relies on the stereotype extension mechanism that permits the language grammar to be extended. This Parametric diagram shows how the overall cost effectiveness of the HSUV will be evaluated. It shows the particular MOEs for one alternative for the HSUV design, and can be reused to evaluate other alternatives.



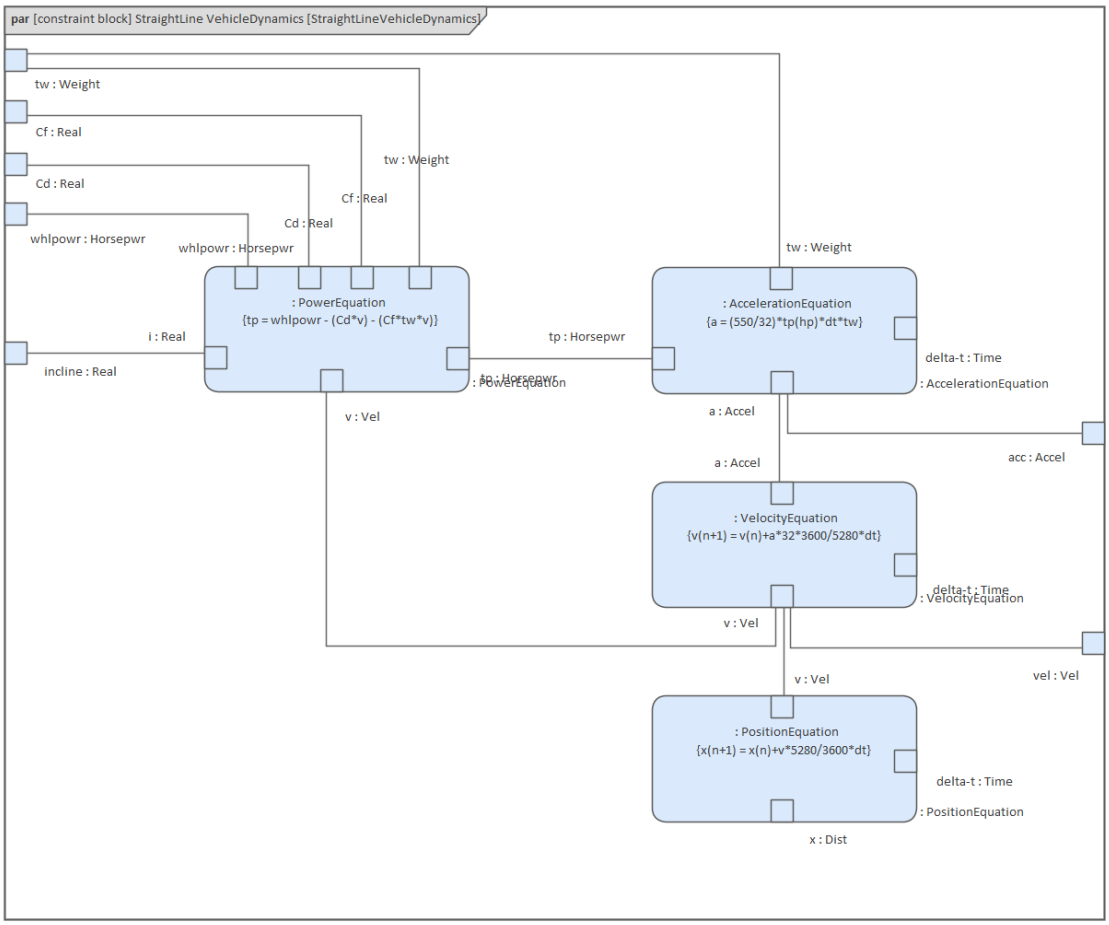
Parametric Diagram - Economy

This Parametric diagram is used to model the fuel economy as it has been defined as an important high level requirement for the SUV and so needs to be assessed to ensure the final product will perform adequately. The Parametric diagram contains constraint properties (instances of the constraint blocks). There are a number of equations that contribute to the evaluation of the overall economy including: Aero Drag Equation, Regenerative Break Efficiency Equation, Straight Line Vehicle Dynamics, Rolling Friction equation and the Fuel Efficiency equation. The constraint properties contain boundary mounted parameters that are connected to other parameters by binding connectors.



Parametric Diagram - Dynamics

In this Parametric diagram the Constraint Block - Straight Line Vehicle Dynamics - from the previous example has been expanded to show how it can be modeled with a number of constraint properties. The Straight Line Vehicle Dynamics constraint is represented by the diagram frame and the constituent equations that contribute to the overall equation are modeled in the diagram as constraint properties. Each constraint on which the constraint properties are based has a constraint equation defined, which is shown in curly braces {} on the diagram; for example, the Acceleration Equation is defined within the Constraint Block as {a = (550/32)*tp(hp)*dt*tw}. Binding Connectors are used to relate the parameters (variables) in one equation to the parameters (variables) in another equation.



(Non-Normative) Timing Diagram - 100hp Acceleration

Enterprise Architect has a power capability to generate plots of Parametric diagrams using its OpenModelica or Simulink integration. One of the great benefits of modeling physical systems is to be able to analyze the way a system would behave in a real world context, without the need to build expensive prototypes or to have to perform the test on the built system itself. The ability to model mathematical equations that govern the way a system will operate, and to create models of these as constraints using Block Definition and Parametric diagrams, provides the precursors to model simulation.

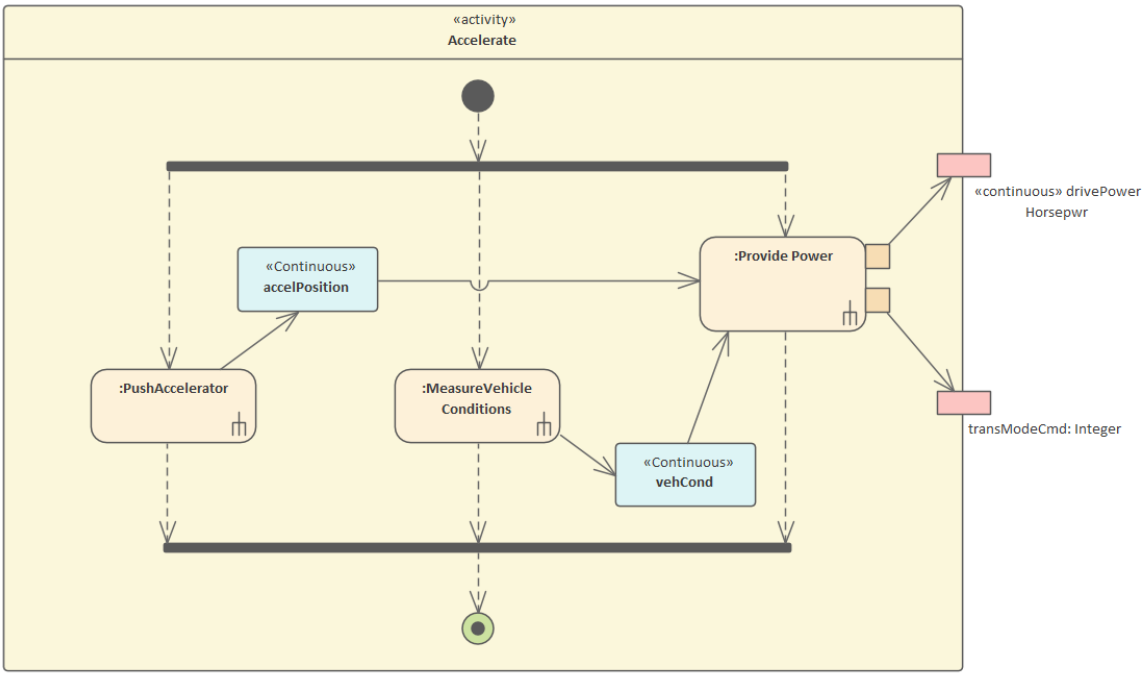
Enterprise Architect harnesses the power of an open tool called OpenModelica, which is underpinned by the Modelica language to generate plots and graphical representations of equations in motion.

Defining, Decomposing, and Allocating Activities

The examples in the topics of this section use Activity diagrams that describe behavioral aspects of the Hybrid SUV, using Actions that are responsible for defining the work, which is ultimately carried out by instances of Blocks. There are also a number of Internal Block diagrams that demonstrate the way that allocations can be represented.

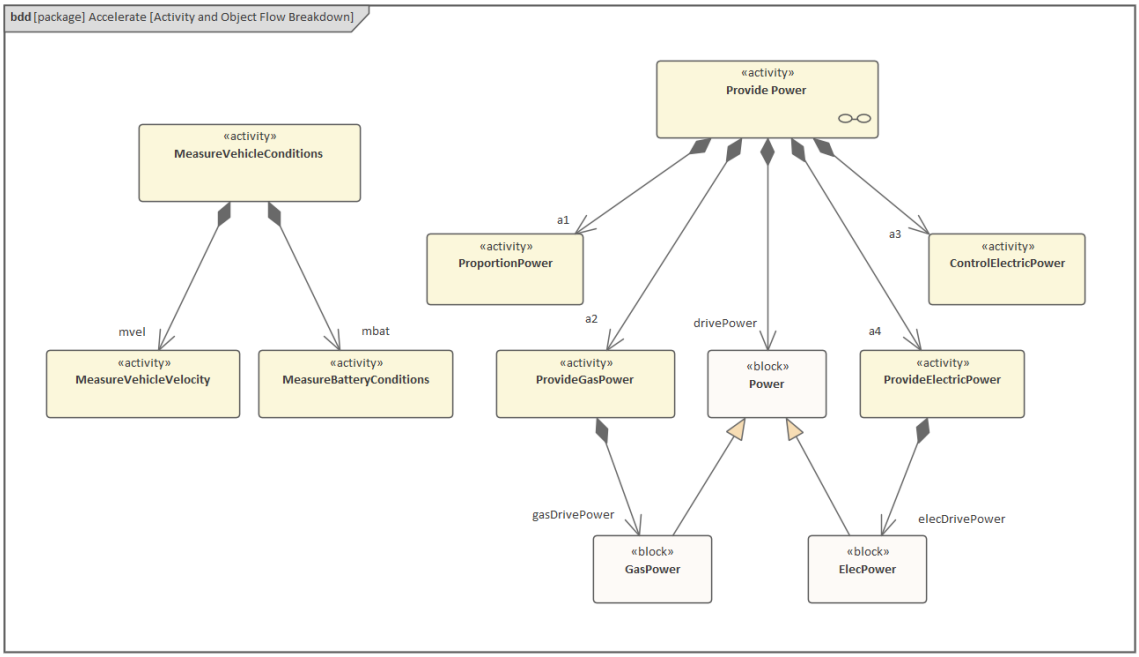
Activity Diagram - Acceleration (top level)

This Activity diagram shows the top level behavior of an Activity representing the acceleration of the HSUV. It is the intent of the System Engineer in this example to allocate this behavior to parts of the Power Subsystem. It is quickly found, however, that the behavior as depicted cannot be allocated, and must be further decomposed. The stereotypes on the object nodes between actions in the figure apply to parameters of the behaviors or operations called by the actions



Block Definition Diagram - Acceleration

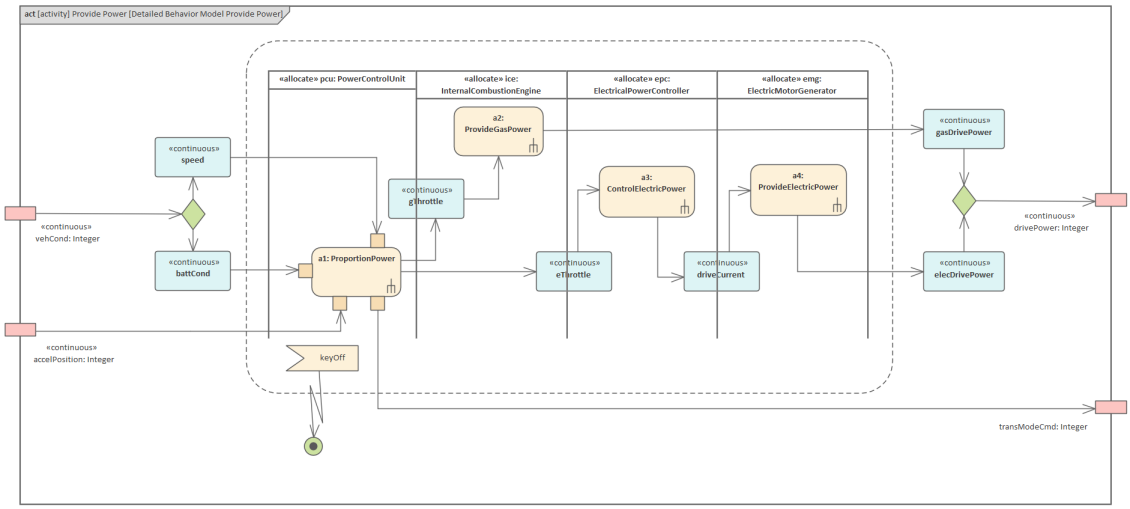
This Block Definition diagram shows the decomposition of the Provide Power Activity from the diagram in the previous topic. It is important to note that this is a functional decomposition and, as such, it defines structural relationships between the Activities in the hierarchy and so should be modeled on a Block Definition diagram.



Activity Diagram (EFFBD) - Acceleration (detail)

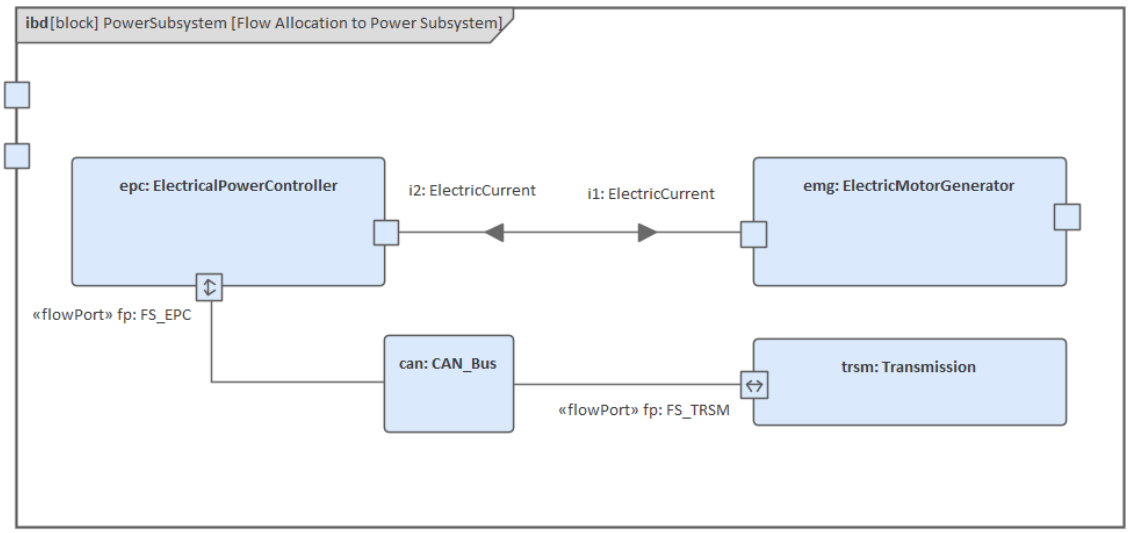
This Activity diagram has the Provide Power Activity as a diagram frame, which includes Actions invoking the decomposed Activities and Object Nodes from the previous diagram. It also uses Allocate Activity Partitions, which are oriented vertically in the diagram. These partitions include the Power Control Unit, the Internal Combustion Engine, the Electric Power Controller and the Electric Power Generator, which are used to show which part of the system is responsible for the Actions defined in the diagram. There is also an allocation callout to explicitly allocate activities and an object flow to parts in the Power Subsystem Block.

The modeling engineer has used incoming and outgoing object flows for the ProvidePower Activity. This was done to distinguish the flow of electrically generated mechanical power and gas generated mechanical power, and to provide further insight into the specific vehicle conditions being monitored.



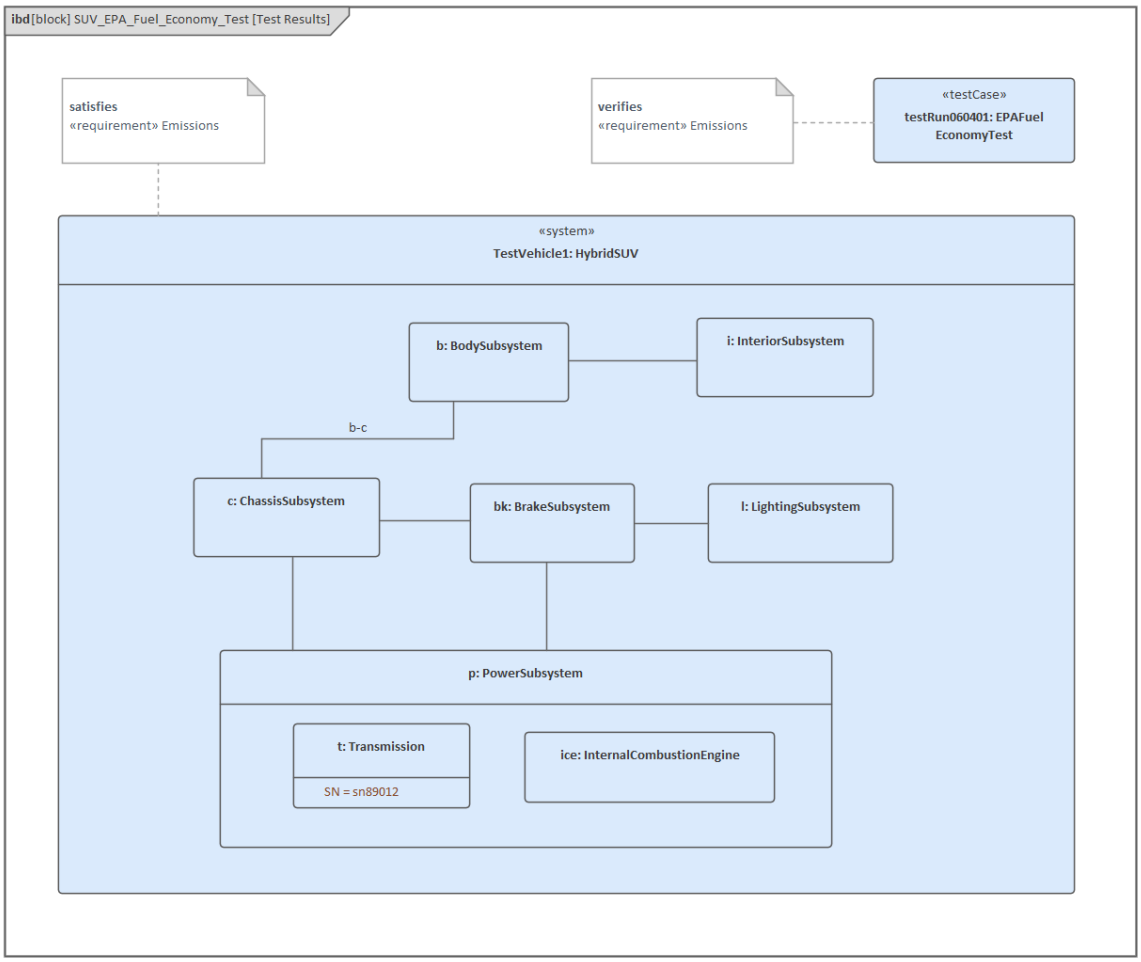
Internal Block Diagram - Power Subsystem Behavioral and Flow Allocation

This partially-completed Internal Block diagram elaborates on some of the allocation relationships shown in the previous example. Here we see how Blocks that have been added to the diagram as properties communicate with each other, and we can see the flow of items from one instance of a Block to another. Specifically, the Electric Power Controller is connected to the Electric Motor Generator and we can see electrical current is flowing between the two properties.



Internal Block Diagram: Property Specific Values - EPA Fuel Economy Test

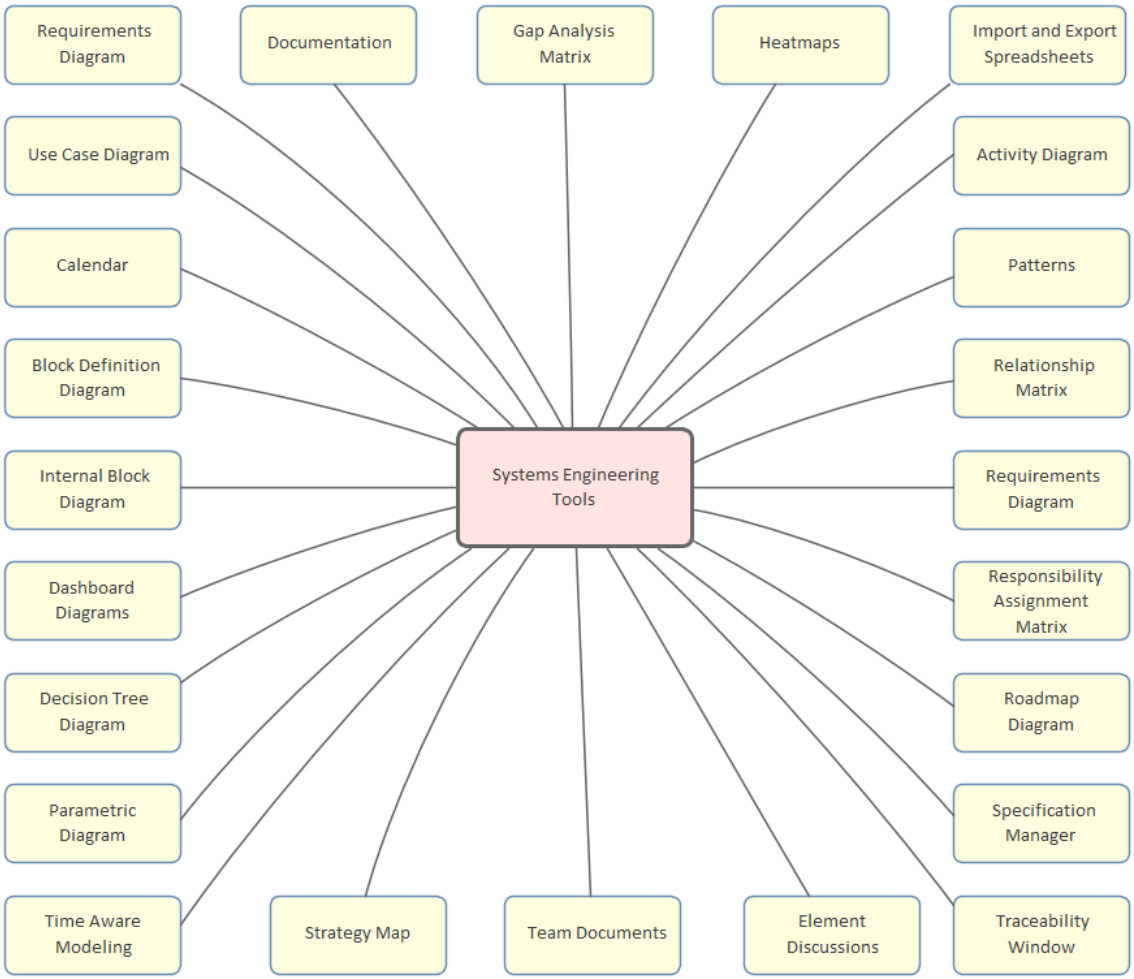
This Internal Block diagram demonstrates the way Test Cases can be modeled in the context of the representation of sub-systems. The tests have been conducted for a specific instance of the Hybrid SUV that presumably has come off the production line and has a specific VIN (vehicle identification number). The test that has been conducted is the EPA fuel economy test. Serial numbers of specific relevant parts can also be indicated to identify any problems that instances of the parts might have.



Meet the Systems Engineering Tools

Enterprise Architect is a sophisticated and flexible Model Based System Engineering platform that can be used as both a repository and a tool for managing engineering projects. It can be used across the entire life cycle, from setting up a Systems Engineering program or practice, through planning, managing, developing and documenting engineering endeavors, to the governance of implementation projects that consume the designs and architectural output. The tool can be used with any processes and any number of languages of representation including SysML, UML, ArchiMate or BPMN. There is a wide range of facilities and tools that allow the engineer to work using their preferred methods, such as word processor views, spreadsheet views, diagrams, Relationship Matrices and a range of other core and extended features.

This Mind Map shows the landscape of the key Systems Engineering tools that can be used to set up and maintain any number of Model Based Systems Engineering initiatives. While these are the primary tools, there is a range of other tools that individual teams or engineers will find useful; these can be explored through the User Guide.

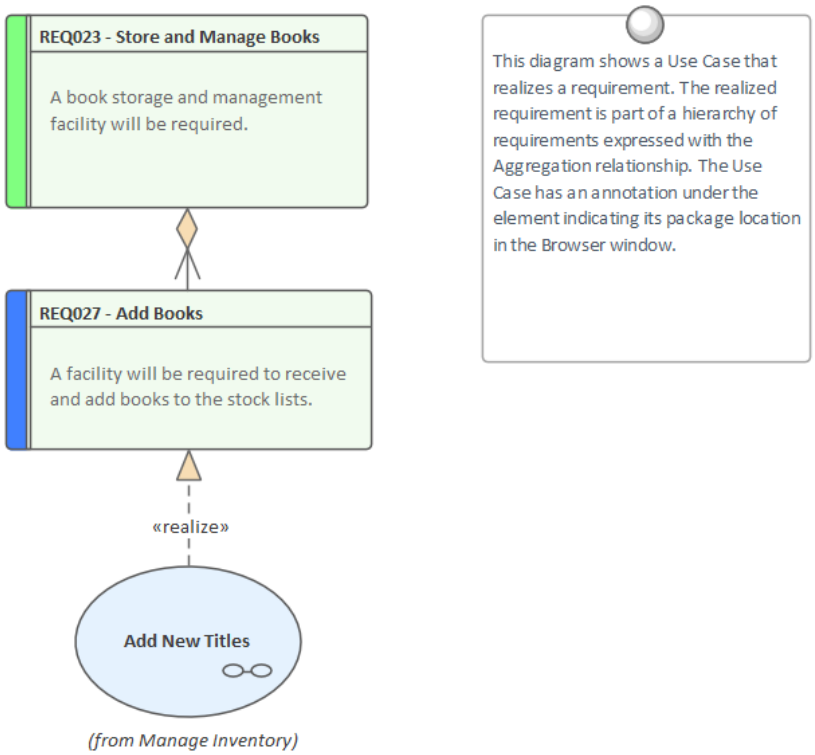


Requirements Diagram

Getting to Know the Requirements Diagram

Introducing the Requirements Diagram

The Requirements diagram provides a visual representation of how Requirements are related to each other and to other elements in the model, including Business Drivers, Constraints, Business Rules, Use Cases, User Stories, design Components and more. The diagram is one of Enterprise Architect's extended diagram types. It provides an appealing graphical representation of Requirements, that will be a welcome change for Requirements Analysts who are accustomed to working with text based tools.



Where to find the Requirements Diagram

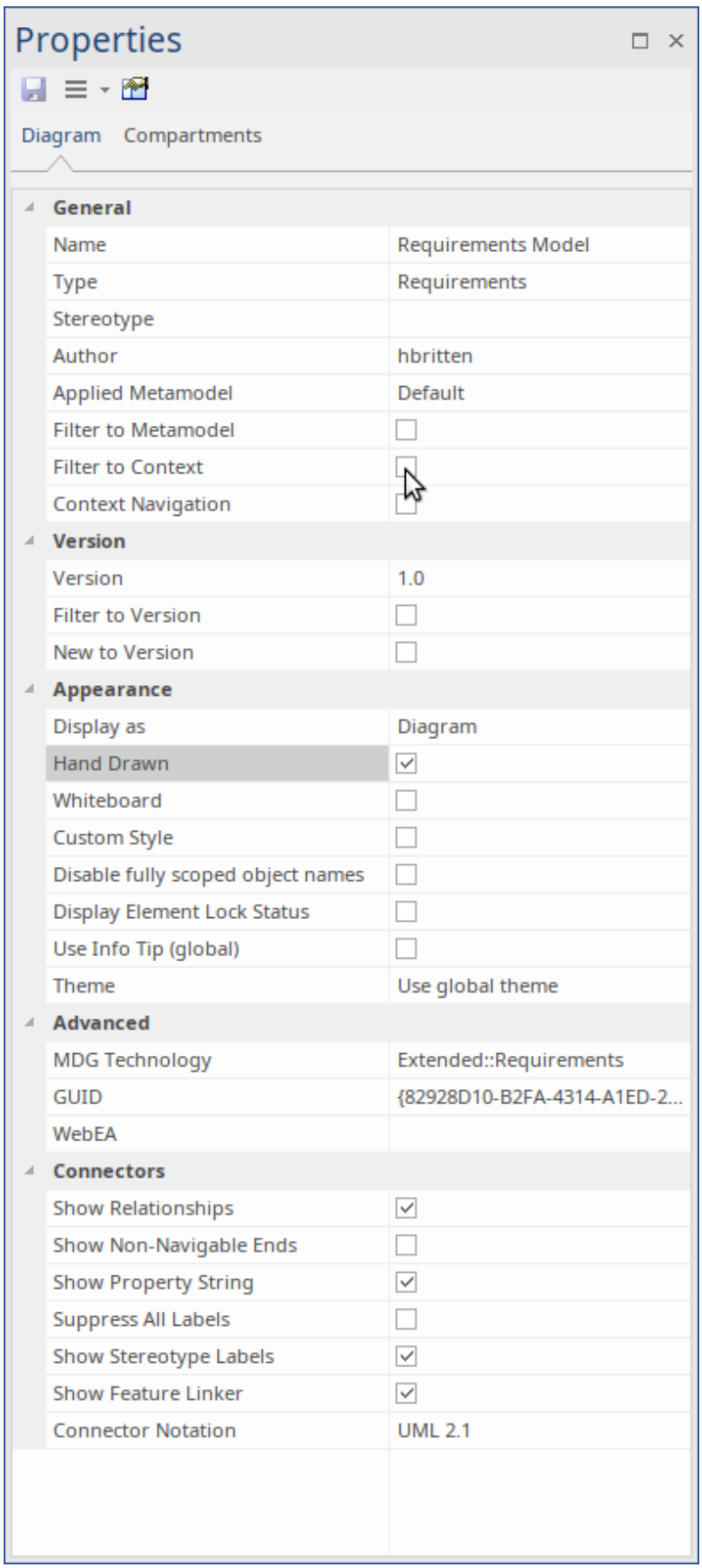
Browser window Context Menu : Add Diagram : | Manage | Show All Perspectives | Extended | Requirements

Usage of the Requirements Diagram

One usage is to show how Requirements are connected together in a hierarchy or, even more importantly, how Requirements are connected to other elements. The experienced modeler will define and manage the Requirements in the Specification Manager and then use the Requirements diagram to show how each Requirement is related to upstream process elements such as Business Drivers, and downstream process elements such as Use Cases, User Stories, User Experience designs and solution Components.

Options for the Requirements Diagram

The appearance of a diagram can be changed to suit the audience, and details can be included, suppressed or altered to ensure the diagram meets its main objective of communication. There is a wide range of options, ranging from creating a Hand Drawn style of diagram to filtering diagram content.



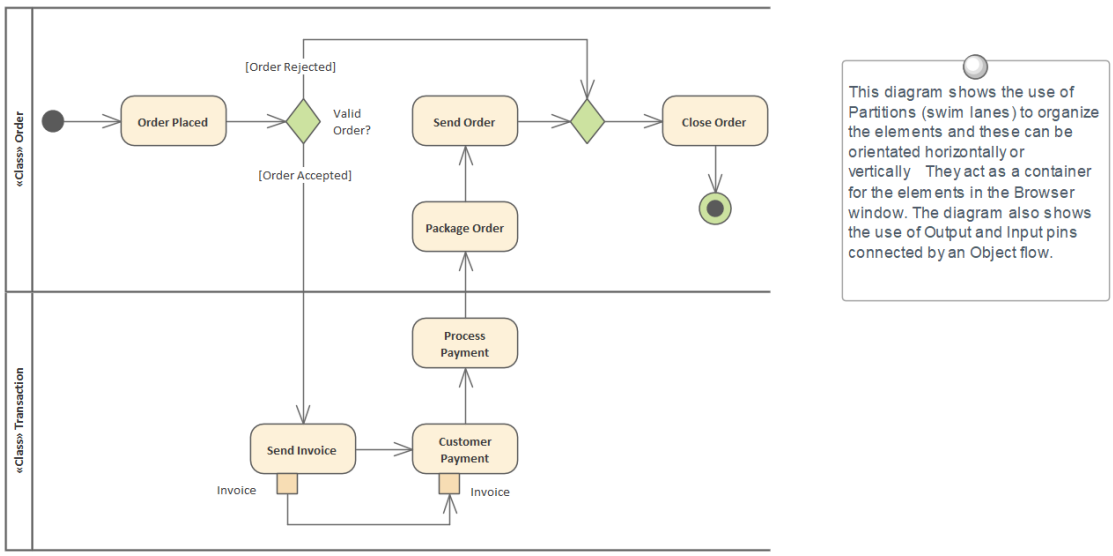
Learn more about the Requirements Diagram

Working In Diagrams

Activity Diagram

The Activity diagram is one of the Unified Modeling Language (UML) Behavioral diagrams that can be used to model a process or algorithm as a sequence of steps. It is a more sophisticated version of its close cousin the Flowchart diagram. Activity diagrams can be used to model Business Processes as a UML alternative to the BPMN Business Process diagram; they have the same ability to create a hierarchy of Activities in the Browser window.

Activity Diagram showing the use of Partitions



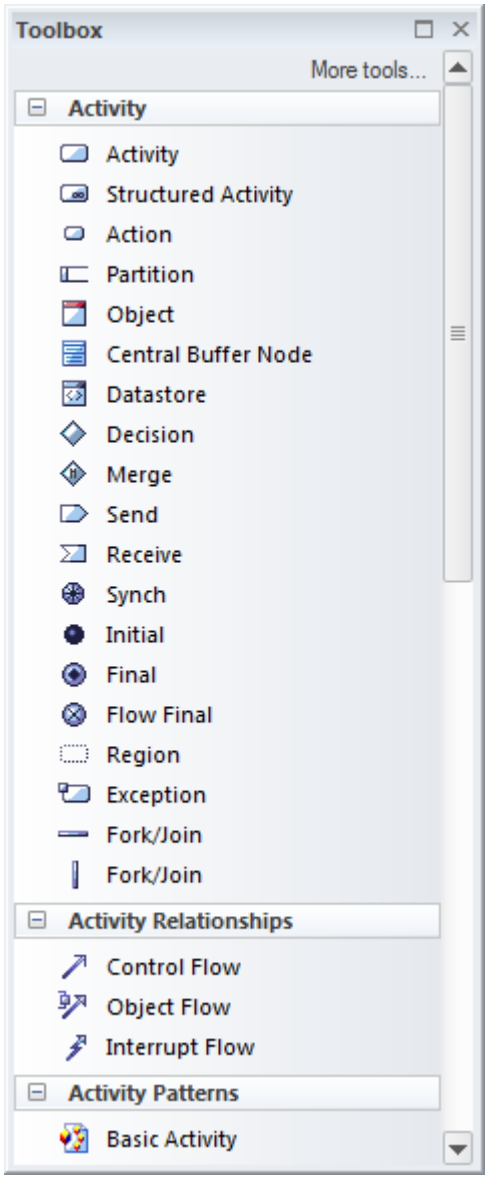
The elements can be given a name and detailed descriptions can be added to the Notes. By connecting the Activities, Decisions and Forks with connectors (Control Flows) a sequence of elements can describe the business process. A process hierarchy can be constructed by nesting Activities in the Browser window and using the child diagram functionality to enable drill down from the value chain level to the lowest level processes.

Getting to know the Activity Diagram

Where to find the Activity Diagram	Ribbon: Design > Diagram > Add Diagram > UML > Behavioral > UML Behavioral > Activity Browser window Toolbar : New Diagram icon > UML > Behavioral > UML Behavioral > Activity Browser window context Menu | New Diagram... > UML Behavioral > Activity
Usage of the Activity Diagram	The Activity diagram can be used to model any business or technical activity or notion that has a series of steps. This includes business and technical processes and also computer algorithms. The steps are connected by Control Flow relationships that show the sequencing of the steps. Decisions and Merges can be used to model choice and to further control the flow through the Activity. Forks and Joins can be added to split and reunite the flow of control and objects added to show how data is supplied and consumed.
Options for the Activity Diagram	Activity diagrams can be drawn at different levels of formality, from a Basic Flow Chart style of diagram used to represent a simple Business Process to a sophisticated Action-based diagram that can be used to model a complex system. There is a toolbox that contains a range of elements, relationships and Patterns for creating the models.

The Activity diagram (like any diagram) can be viewed as an Element List, which makes working with element properties easier.

Diagram Filters can also be used when presenting the diagrams, to draw attention to parts of the diagrams, and the diagrams can be presented in hand drawn or whiteboard style by changing the properties of the diagram.



Learn more about the Activity Diagram

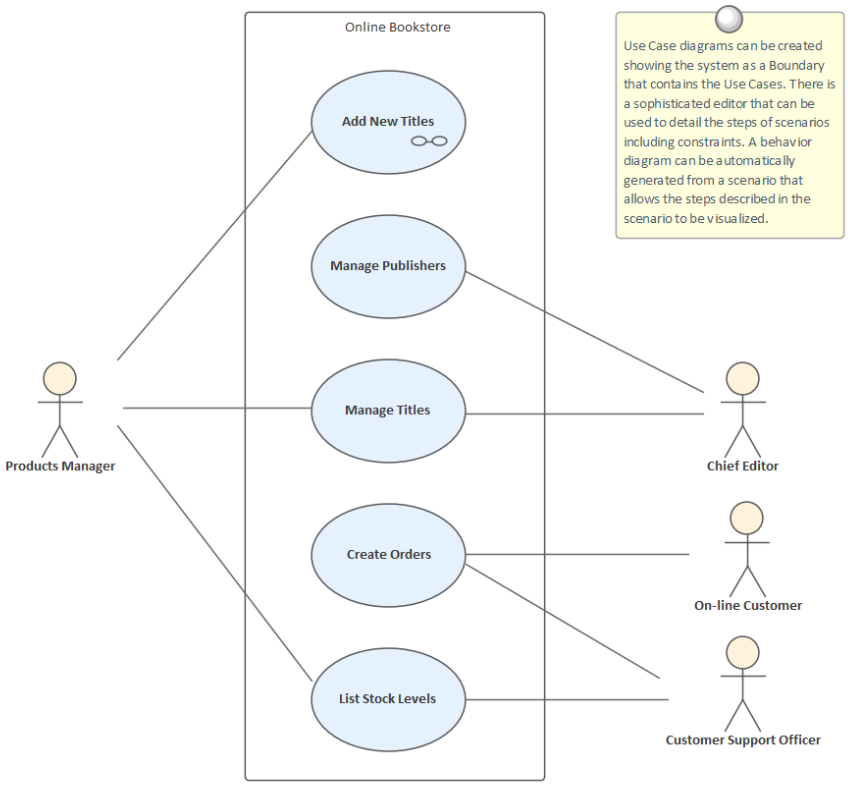
Activity Diagram

Use Case Diagram

Getting to know the Use Case diagram

Introducing the Use Case Diagram

The Use Case diagram is one of the Unified Modeling Language (UML) Behavioral diagrams that can be used to describe the goals of the users and other systems that interact with the system that is being modeled. They are used to describe the functional requirements of a system, subsystem or entity and present a simple but compelling picture of how the system will be used.



They are typically used in conjunction with higher level Business and Stakeholder Requirements and are often supplemented with a set of Non Functional Requirements.

Where to find the Use Case Diagram

Ribbon: Design > Diagram > Add Diagram> UML Behavioral > Use Case

Browser window Toolbar : New Diagram icon > UML Behavioral > Use Case

Browser window context menu | New Diagram... > UML Behavioral > Use Case

Usage of the Use Case Diagram

The Use Case diagram is used to describe the goals that users or other systems want to achieve from interacting with the system. They always describe the goal from the Actors' perspective, the details of the Use Case will describe the goal with more precision.

Use Cases will often act as the basis for the definition of Test Cases.

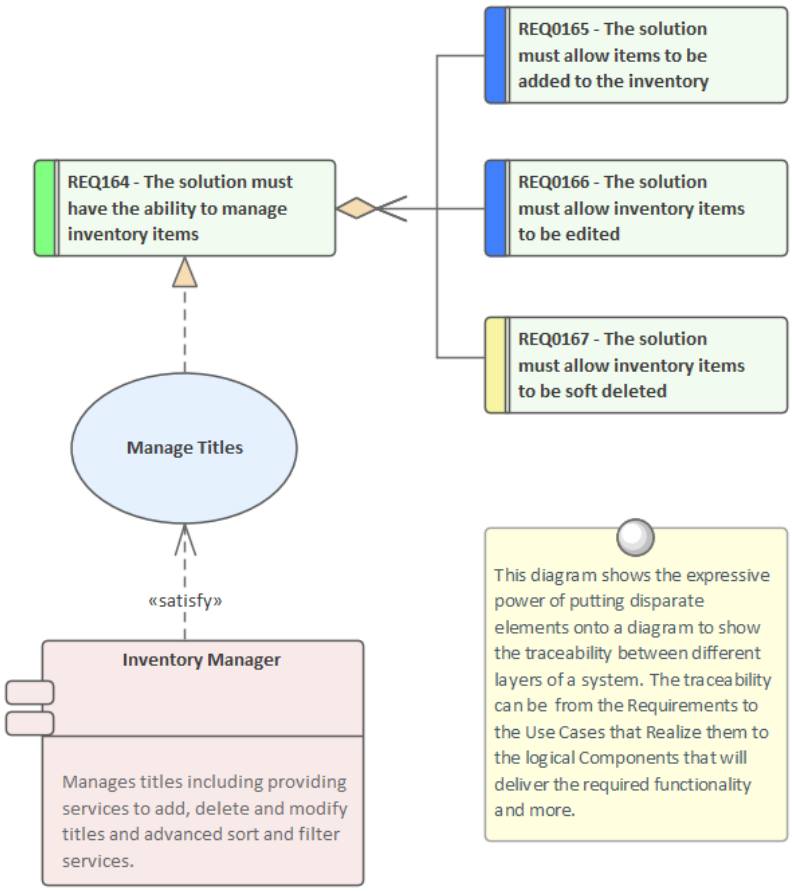
Options for the Use Case

Any number of Use Case diagrams can be created to represent different parts of a

Diagram

system or Packages of Use Cases. The diagrams can be kept simple or they can be structured by the application of a number of additional connectors such as Include, Extend and Generalization relationships.

A system Boundary can be included that is used to name the system, subsystem or entity under discussion; the Actors lie outside the Boundary and the Use Cases inside.



Use Case diagrams can be used to show how the Use Case are related to other elements in the system, including up-stream elements such as Requirements and down-stream elements such as Components.

The Use Case diagram (as for any diagram) can be viewed as an Element List, which makes working with the element's properties easier.

Diagram Filters and Diagram Layers can also be used when presenting the diagrams, to draw attention to parts of the diagrams and the diagrams can be presented as hand drawn or in a whiteboard style by changing the properties of the diagram.

Learn more about the Use Case Diagram

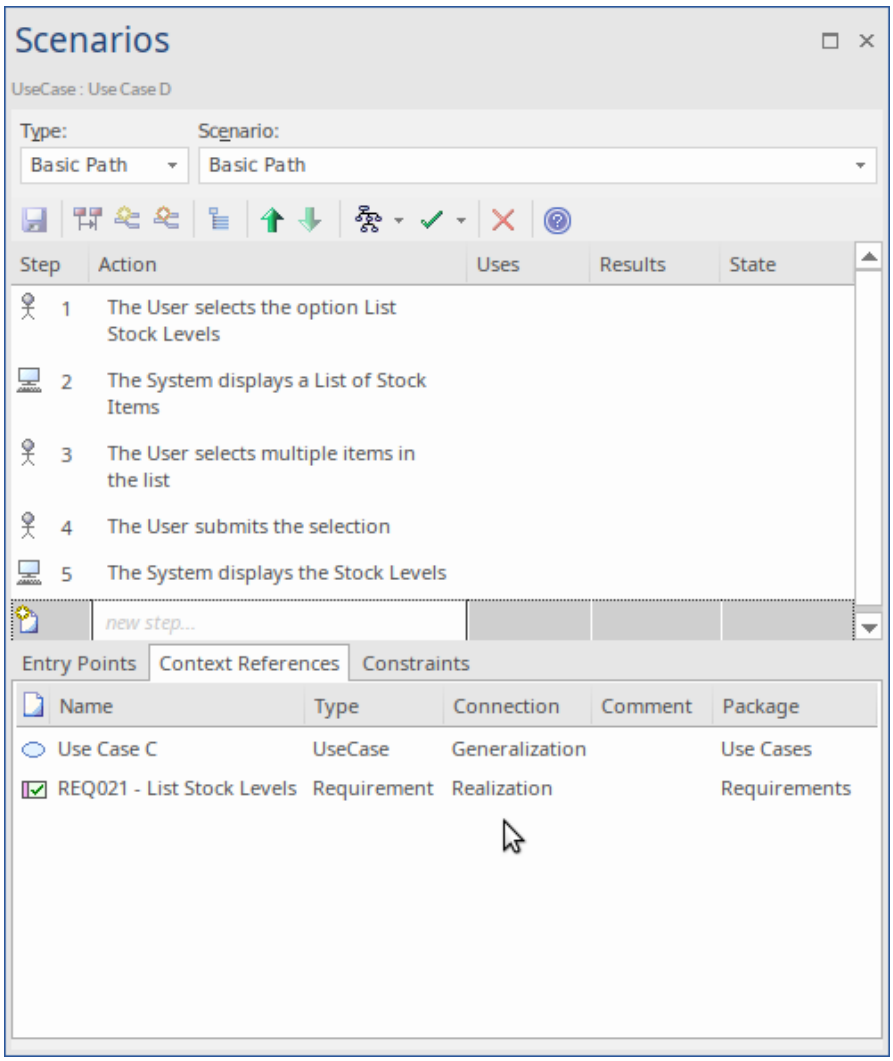
Use Case Diagram

Scenario Builder

Getting to Know the Scenario Builder

Introducing the Scenario Builder

The Scenario Builder is used to define the details of a Use Case including defining detailed descriptions, creating one or more Scenarios and defining pre-conditions, post-conditions and other constraints. The detailed steps of a Use Case can be recorded and linked to other elements in the model and these can then be generated out as a diagram providing a visual representation of the Use Case and its Scenarios. The diagram and the text can be synchronized and individual steps can then be traced to other elements such as Components that will realize the Requirement specified in the Use Case.



Where to find the Scenario Builder

Start > Application > Edit > Responsibilities > Structured Scenarios

Design > Element > Editors > Responsibilities > Structured Scenarios

Element Context Menu: Properties | Responsibilities > Scenarios | right click | Add New : Structure Editor

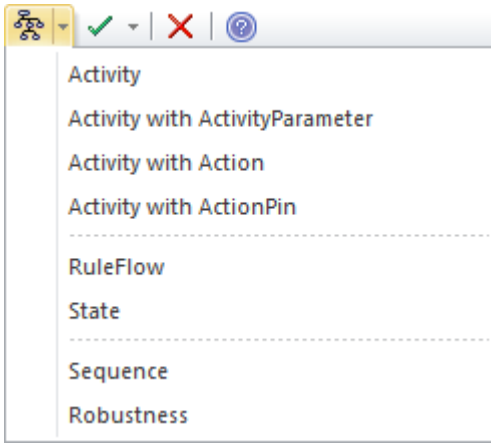
Usage of the Scenario Builder

To define the details of a Use Case and its scenarios and constraints, which can be used to replace the traditional text-document based approach to defining Use

Cases. This ensures that the Use Case diagram and the textual details of the Use Cases and its Scenarios and Constraints are all contained in the same model and can be traced. If the Use Cases are required in a document format for contractual or process reasons, a Use Case Report can be generated automatically from the models using the in-built documentation engine.

Options for the Scenario Builder

The Scenario Builder can be viewed as a tabbed or a docked window or in an element's Properties window. The steps of a Use Case including its Scenarios can be automatically generated into a number of different diagram types available from the Generate Diagram toolbar icon.



Learn more about the Scenario Builder

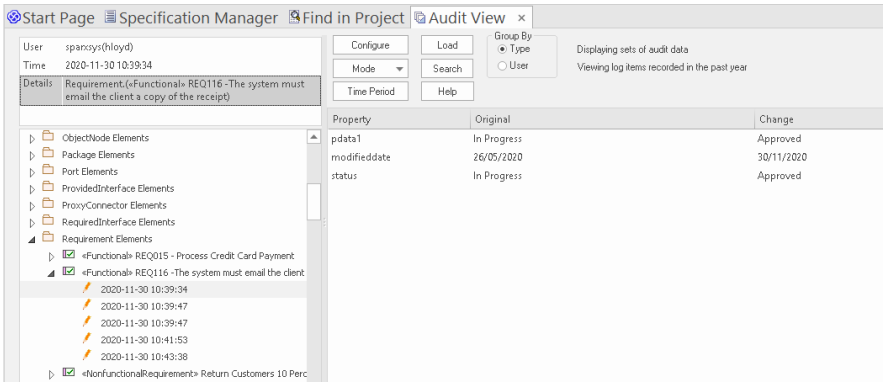
Scenarios

Auditing

Getting to know Auditing

Introducing Auditing

The Auditing feature can keep track of the changes to Requirements including what was changed, when it was changed and by whom. Auditing is by default disabled and must be enabled before the changes to requirements will be recorded. Once enabled it is a passive tool that silently records the changes to elements. It does not replace Version Control or Baselines and in contrast to these tools it can not be used to return to a previous state of the model. Change management, governance and quality control are all aided by the use of Auditing.



Where to find Auditing

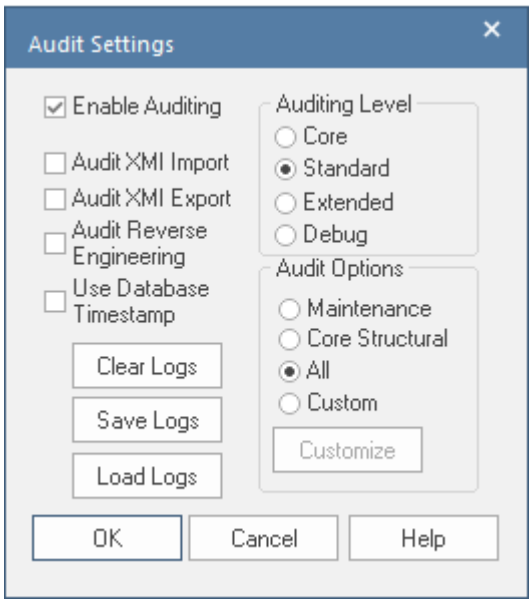
Ribbon: Settings > Model > Auditing

Use of Auditing

Auditing can be used to track what was changed in a model, who changed it and when. There are a number of modes and a repository administrator can use the settings to specify what is recorded in the audit. While a baseline can be used to show the difference between a model and a snapshot at a point in time, the Auditing tool records each individual change; it can not, however, be used to revert to a previous state.

Options for Auditing

There is a wide range of settings to configure auditing, starting with enabling or disabling the settings that determine which elements have an audit trail and the level of detail recorded. Audit logs can be exported from the repository to increase performance.



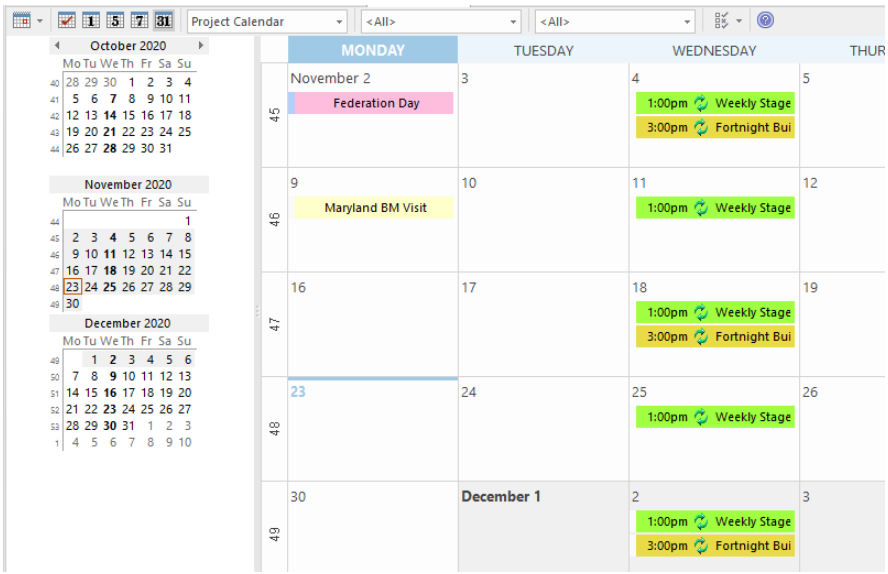
Learn more about Auditing Auditing

Calendar

Getting to know the Calendar

Introducing the Calendar

The Calendar is a fully featured mechanism for recording the important events in an initiative and displaying other information such as resource allocation. There are day, week and month views and the display can be set to show Calendar entries, Project Tasks and Resource Allocation. When a resource has been allocated - for example to analyze a set of requirements - a user can drill through from the Calendar to the requirements' location in the Browser window.



There are also fully configurable Event Types, Categories and colors. The work of a Business Analyst will involve a wide range of events including things like: workshops, interviews, focus groups, collaborative games, brainstorming sessions, reviews, observations and meetings. All of these events can be conveniently recorded and managed in the Calendar. When resources have been allocated to elements and tasks have been assigned to individuals these can be displayed in the Calendar.

Where to find the Calendar

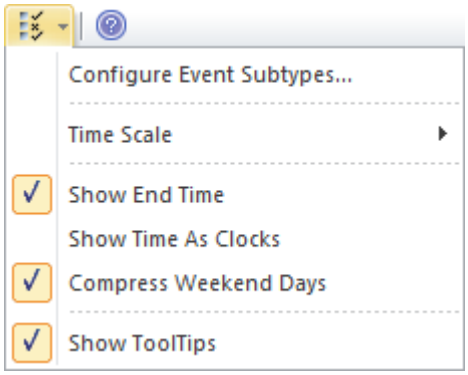
Ribbon: Start > Collaborate > Calendar

Usage of the Calendar

The Calendar can be used to schedule and view events such as meetings, milestones, reviews, workshops and more. It can be used to view the allocation of resources to elements in the repository such as who is analyzing a set of requirements. It can also be used to view Project Tasks. An analyst can conveniently click through to the elements in the Browser window or the Project Tasks.

Options for the Calendar

The Calendar has a number of options including the ability to create recurrent events. There is an options toolbar icon that allows aspects of the Calendar's appearance to be configured.



Learn more about the Calendar The Model Calendar

Block Definition Diagram

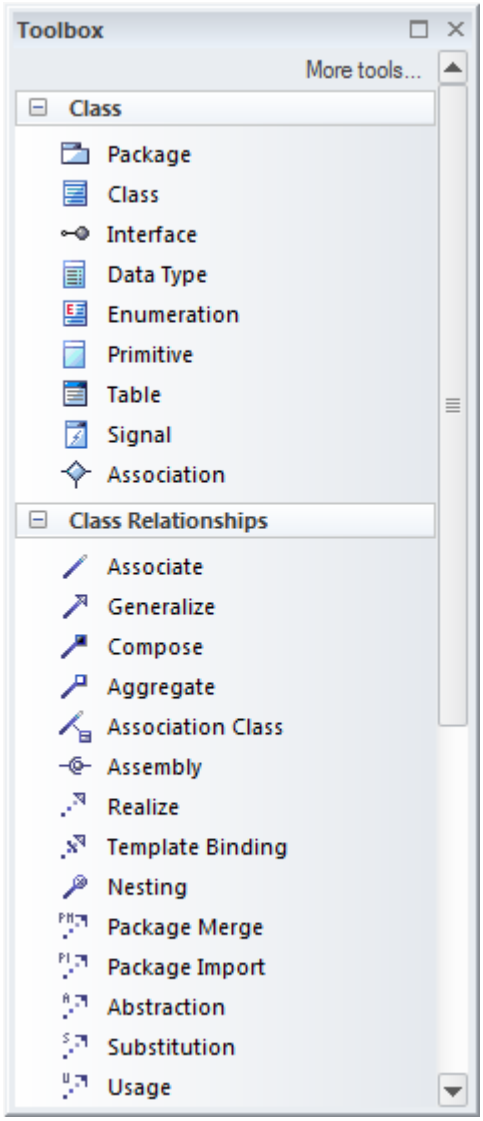
Getting to know the Block Definition Diagram

Introducing the Block Definition Diagram	The Block Definition diagram is one of the Systems Modeling Language (SysML) Structural diagrams that can be used to model a wide range of things. It is a general purpose diagram for modeling entities in the business and technical domains, including terms and concepts, Business Rules, and Capabilities in XML and Database Schemas.
Where to find the Block Definition Diagram	Ribbon: Design > Diagram > Add Diagram : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Block Definition > <required type> Browser window Toolbar : New Diagram icon : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Block Definition > <required type> Browser window Context Menu | New Diagram : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Block Definition > <required type>
Usage of the Block Definition Diagram	The Block Definition diagram can be used whenever a logical or structural representation of a system is required. It has applicability for modeling both business and engineering concepts. It is the fundamental diagram for modeling the structure of a system or subsystem or one of its components.
Options for the Block Definition Diagram	The Block Definition (like any diagram) can be viewed as an Element List, which makes working with the element's properties easier. Diagram Filters can also be used when presenting the diagrams to draw attention to parts of the diagrams and the diagrams can be presented as hand drawn or in a whiteboard style by changing the properties of the diagram.
Learn more about the Block Definition Diagram	Using Blocks to Model Structure and Constraints

Internal Block Diagram

Getting to know the Internal Block Diagram

Introducing the Internal Block Diagram	The Internal Block diagram is used in conjunction with the Block Definition Diagram, but it is typically used to show the internal structure of a Block including its parts and how they work together to deliver the behaviors specified by the block or that have been allocated to it.
Where to find the Internal Block Diagram	Ribbon: Design > Diagram > Add Diagram : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Internal Block > <required type> Browser window Toolbar : New Diagram icon : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Internal Block > <required type> Browser window Context Menu | New Diagram : Perspective = Systems Engineering > SysML Select From = SysML 1.5 Diagram Types = Internal Block > <required type>
Usage of the Internal Block Diagram	The Internal Block diagram is used to model the internal structure of a block including its parts and the relationship between those parts.
Options for the Internal Block Diagram	The Internal Block diagram (like any diagram) can be viewed as an Element List, which makes working with the element's properties easier. Diagram Filters can also be used when presenting the diagrams to draw attention to parts of the diagrams and the diagrams can be presented as hand drawn or in a whiteboard style by changing the properties of the diagram.

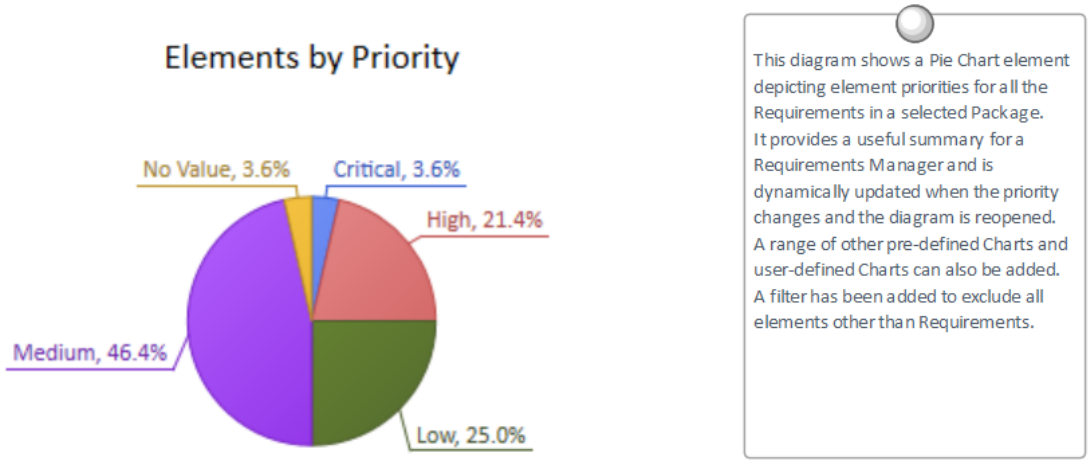


Learn more about the Internal Block Diagram

Using Properties and Parts to Model Block Usage

Dashboard Diagrams

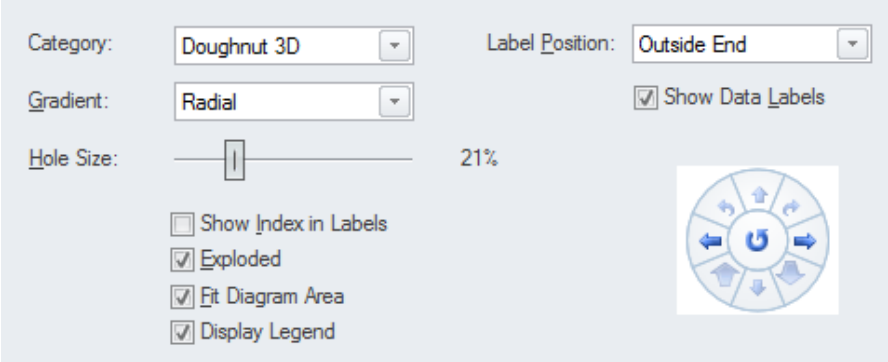
Dashboard diagrams allow you to create high quality Charts and graphs to display repository information in a visually compelling way. This diagram is an example of creating a Dashboard d**iagram** in Sparx Systems Enterprise Architect; it illustrates the ratio of Requirement Priorities in a Pie Chart.



Enterprise Architect provides a Toolbox page of pre-configured Charts and graphs, but you are free to create and save any number of Charts, sourcing data from anywhere in the repository. The Charts and graphs provide valuable summary information that assists in the management of Requirements. High level reporting and project status can be easily tracked and documented using the numerous Charts and report elements available, which tightly link in with the model content and status.

Getting to Know Dashboard Diagrams

Where to find Dashboard Diagrams	Browser window | Right-click on Package | Add Diagram : | Manage | Show All Perspectives | Extended | Dashboard
Usage of Dashboard Diagrams	Dashboard diagrams present rich yet easily understood views of information - such as the status of Requirements in a particular release of the system - that can be opened inside the model or conveniently copied directly into management or project team presentations. They are useful for planning an iteration such as an Agile sprint to view how ready the Requirements are for the implementation team; for example, to view what percentage of the Requirements have been approved and are of high priority.
Options for Dashboard Diagrams	The standard Charts and graphs available from the Toolbox can be configured in a number of ways, including changing the source, applying filters or modifying the appearance of the Chart as indicated in this diagram, available from the Chart's Properties window using the 'Appearance' section.



Learn more about Dashboard Diagrams

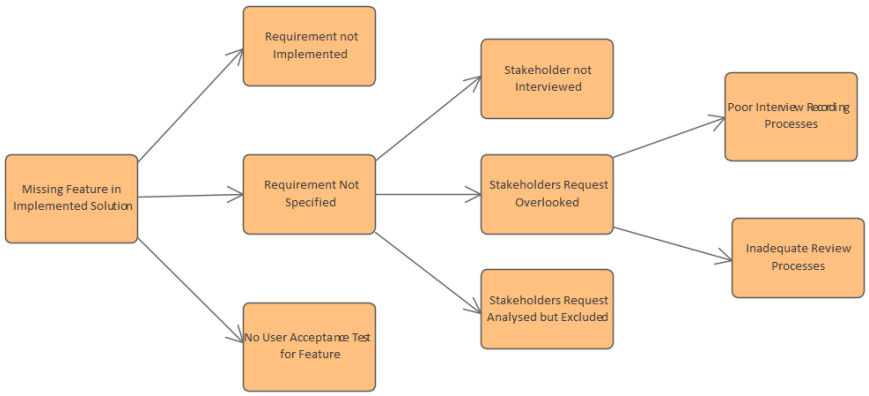
- Standard Charts

Decision Tree Diagram

Getting to know the Decision Tree Diagram

Introducing the Decision Tree Diagram

Decision Trees are an effective way of graphically representing a number of options and provide a mechanism to investigate the possible outcomes and benefits of choosing those options. They can also assist the analyst to form a balanced picture of the risks and benefits associated with each possible course of action. They are a close cousin of the Decision Table but have the benefit of being graphical. Enterprise Architect has a purpose-built diagram allowing complex decisions to be modeled and displayed including probabilities and uncertainty.



Where to find the Decision Tree Diagram

Ribbon: Design > Diagram > Add Diagram > Strategic Modeling > Decision Tree

Browser window Toolbar : New Diagram icon > Strategic Modeling > Decision Tree

Browser window context menu | New Diagram... > Strategic Modeling > Decision Tree

Usage of the Decision Tree Diagram

Decision Trees can be used to help in decision making processes, particularly when the decision involves a complex set of conditions that have different likelihoods of occurrence. They can be used for strategic or operational decision analysis and can help to formalize the basis of decision making particularly when it is imperative that actions that are taken are based on formal analysis or have expensive consequences. A Decision Tree can be used to present a graphical picture of a Decision Table for stakeholders who are more comfortable viewing diagrams rather than tables and documents.

Options for the Decision Tree Diagram

Decision Trees can be drawn with varying levels of formality from simple trees with a series of decisions resulting in outcomes to more formal trees that involve uncertainty with probability values assigned or formulaic expressions with input parameters. The 'Decision Tree' toolbox page contains a range of elements that can be used, and two Patterns that can be used to create a diagram giving the analyst a starting point.

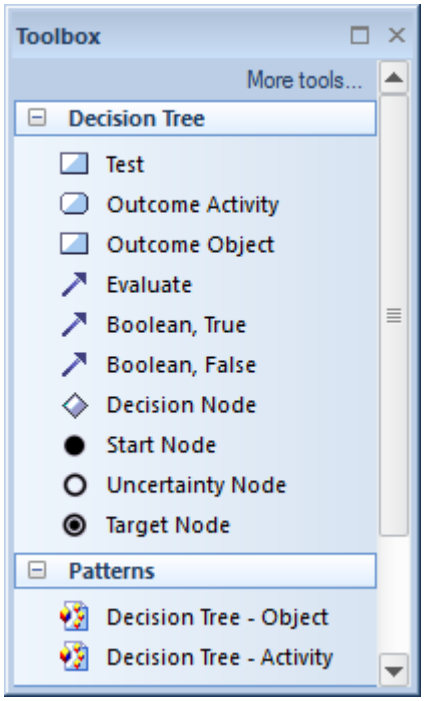


Diagram Filters can also be used when presenting the diagrams to draw attention to parts of the diagrams and the diagrams can be presented as hand drawn or in a whiteboard style by changing the properties of the diagram.

Learn more about the Decision Tree Diagram

Decision Tree

StateMachine Diagram

Getting to know the StateMachine Diagram

Introducing the StateMachine Diagram

The StateMachine diagram is one of the Systems Modeling Language (SysML) Behavior diagrams that can be used to model a wide range of things. It is a general purpose diagram for modeling entities in the business and technical domains, including terms and concepts, Business Rules, and Capabilities in XML and Database Schemas.

Where to find the StateMachine Diagram

Ribbon: Design > Diagram > Add Diagram : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = StateMachine > <required type>

Browser window Toolbar : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = StateMachine > <required type>

Browser window Context Menu | New Diagram : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = StateMachine > <required type>

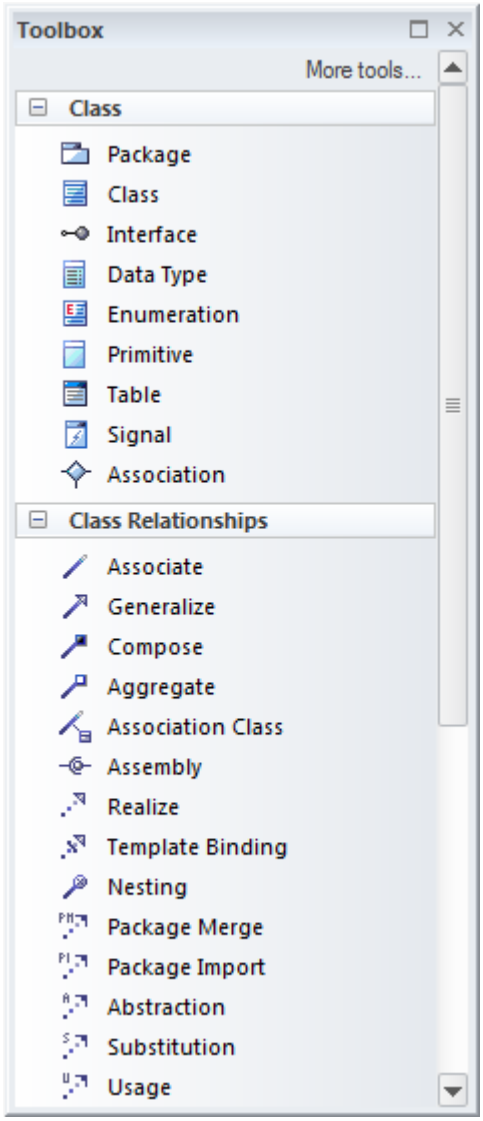
Usage of the StateMachine Diagram

The Class diagram can be used whenever a logical or structural representation of a system is required. It has applicability for modeling both business and technical concepts and can be used to model information and structures such as XML and database schemas.

Options for the StateMachine Diagram

The Class diagram (like any diagram) can be viewed as an Element List, which makes working with the element's properties easier.

Diagram Filters can also be used when presenting the diagrams to draw attention to parts of the diagrams and the diagrams can be presented as hand drawn or in a whiteboard style by changing the properties of the diagram.



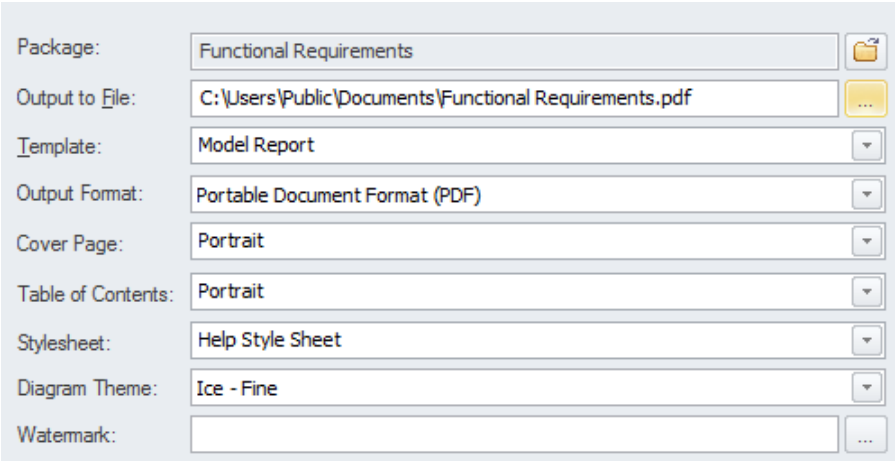
Learn more about the StateMachine Diagram StateMachines

Documentation

Getting to know Documentation

Introducing Documentation

The Documentation features can be used to automatically generate a wide range of documentation directly from the models. These can be document-based such as PDF and Docx format or HTML-based. Flexible templates can be used to completely tailor the documents that are generated including company logos, tables of content, tables of element information and diagrams. Ad-hoc reports can also be created from a number of tools such as the Glossary and the Search Window.



Where to find Documentation

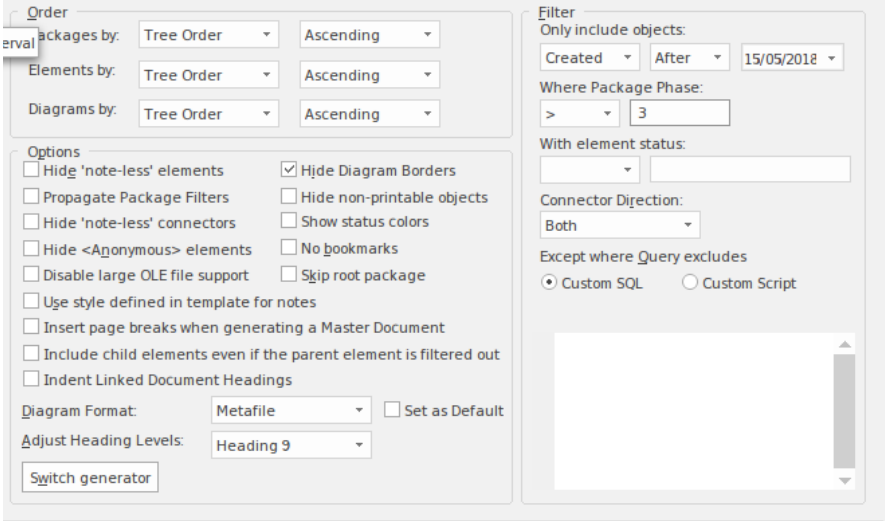
Ribbon: Publish > Model Reports > Report Builder

Use of Documentation

Modelers, Analysts, Architects, Project Managers and others can use the facility to produce a wide range of document-based publications and reports, such as a System Requirements Specification, Use Case Report, Data Dictionary, Solution Architecture Description and more. It can also be used for ad-hoc reporting to create reports such as a list of the most volatile requirements. HTML documentation can also be published to allow stakeholders who don't have access to Enterprise Architect to view the models from an Intranet site that can just be placed on a file system without the need for a Web Server.

Options for Documentation

There are several options that can be set to tailor the information that is included in a generated document, including the ordering of elements and diagrams and hiding certain elements. Filters and word substitutions and other options can also be applied.



Learn more about Documentation

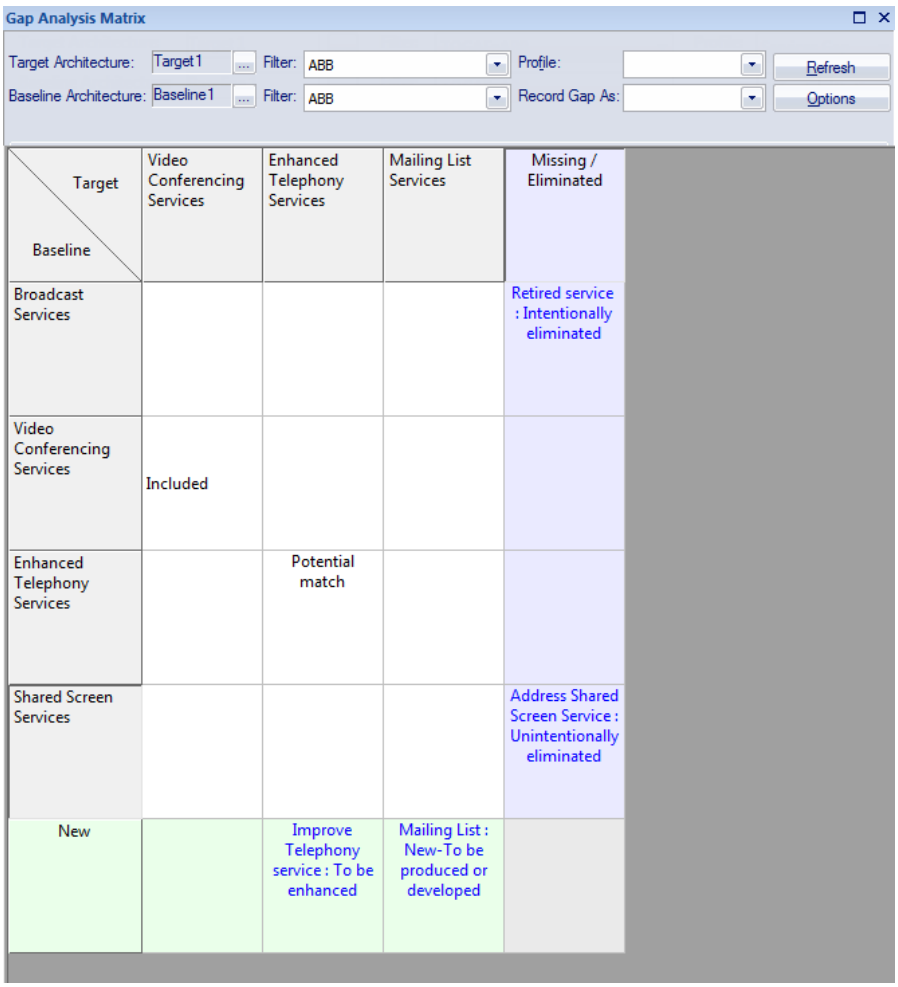
Model Publishing

Gap Analysis Matrix

Getting to know the Gap Analysis Matrix

Introducing the Gap Analysis Matrix

The Gap Analysis Matrix is a specialized Relationship Matrix that is used to record the gaps that exist between two versions of some part of an enterprise. The gaps between two different versions of an architecture could be recorded, or the gaps between two versions of Capabilities or Staff Competencies, or two versions of Information or Data. The tool is structured similarly to a spreadsheet with columns and rows. The elements that make up the baseline (starting point) are listed as rows and the elements that make up the target (end point) are listed as columns. There is a column for recording missing or eliminated elements and a row for recording new elements. At the intersection of a baseline element and target element, notes can be added that describe any details of the relationship between the two elements.



Where to find the Gap Analysis Matrix

Ribbon: Design > Package > Gap Analysis

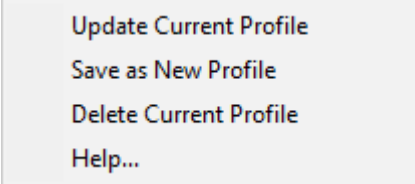
Usage of the Gap Analysis Matrix

The Gap Analysis Matrix can be used for both business and technical analysis. It is a general purpose tool for recording the details of a comparison between different versions of some part of an enterprise. In business analysis it can be used to analyze Staff Competencies, Data and Information, Business Processes, Lines of Business and more, comparing them within current and future states of an

enterprise. In Enterprise Architecture the matrix can be used to record gaps between baseline, transition and target architectures, comparing Capabilities, Architecture and Solution Components and more.

Options for the Gap Analysis Matrix

The Gap Analysis Matrix can be configured to display different parts of the repository. Once the appropriate Packages have been chosen for the Target and Baseline, and the types of element have been selected for the filter, the Gap element type can be selected. The element chosen for the gap will restrict the available elements to represent the gap for 'Missing' or 'New' elements in cells in the matrix. There are a number of choices available from the 'Options' menu, including being able to update, delete and save the Gap Analysis Matrix as a profile, giving it a name so that it can be recalled at a later time.



Learn more about the Gap Analysis Matrix

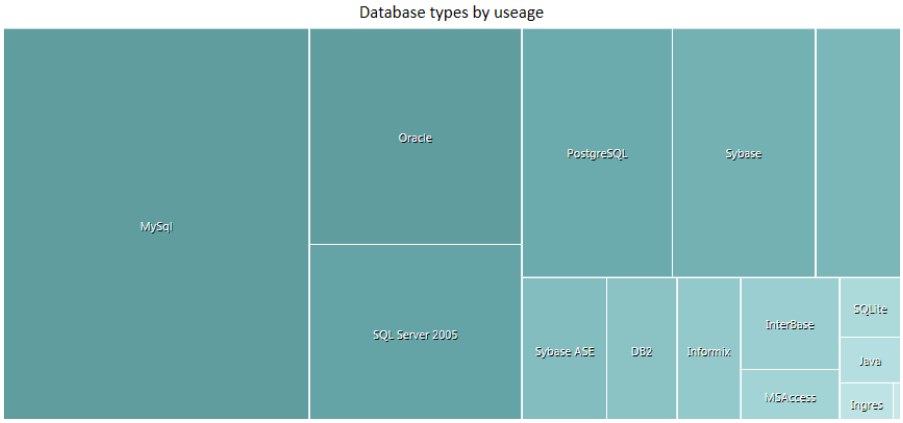
Gap Analysis Matrix

Heat Map

Getting to know the Heatmap

Introducing the Heatmap

A Heat Map is a type of chart that can be used to visualize data in two dimensions. It uses the color of rectangles to indicate a dimension of the data and the relative size of the rectangles to indicate another dimension. They are typically used to create compelling representations of data for strategic or tactical decision making. They can be used at any level of a repository from strategic architecture down to Technology architectures.



Where to find the Heatmap

Double-click on Chart element | Chart Details | Source > Package

Usage of the Heatmap

Heatmaps are typically used to create compelling representations of data for strategic or tactical decision making. They can be used with Requirements to indicate the statuses of a group of requirements and, if the metrics are available, the estimated implementation cost of each requirement. They could be used with an application or technology inventory to show the prevalence of technologies. For example, which applications were developed in a particular language or run on a particular operating system.

Options for the Heatmap

As an alternative to specifying the parameters of the Heat Map in the fields on the 'Package' tab, you can select the 'Custom SQL' tab and create a customized Heat Map using SQL. You still specify the chart type in the 'Type' field, but the other dialog fields are grayed out.

Learn more about the Heatmap

Heat Maps

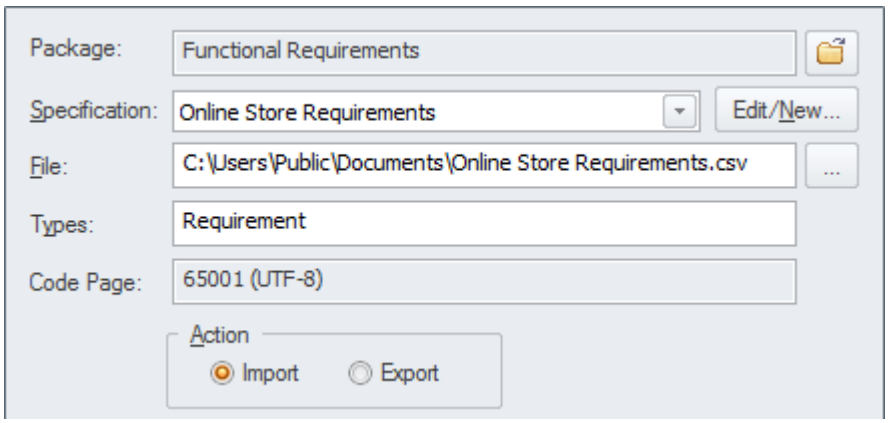
Import and Export Spreadsheets

Import and Export Spreadsheets

Introducing Import and Export Spreadsheets

This facility is a useful mechanism to import Requirements that have been defined in a Spreadsheet or a Word Processor table into Enterprise Architect. Once in Enterprise Architect the Requirements can be managed and traced to elements such as business drivers and Scenarios and Components. Alternatively Requirements in Enterprise Architect can be exported to a Spreadsheet for the purposes of providing them to a third party or for some type of numerical or statistical analysis. The mapping between fields in the Spreadsheet and the analogous properties in Enterprise Architect is completely configurable using a specification.

For more detailed information exchange, the MDG Link for Microsoft Office (available from Sparx Systems) provides additional functionality and integration points useful when dealing with complex Requirements.



Where to find Import and Export Spreadsheets

Ribbon: Publish > Model Exchange > CSV

Use of Import and Export Spreadsheets

This feature can be used to import or export Requirements from a CSV file. Before a tool such as Enterprise Architect was installed, Analysts might have used a Spreadsheet or a table in their favorite word processor to record Requirements; these can conveniently be imported using the CSV import facility. Alternatively, Requirements sometimes have to be provided to a third party who will typically specify that they want them in a Spreadsheet file; this can be achieved using the export facility.

Options to Import and Export Spreadsheets

The import and export facility is completely configurable and has a user-defined specification to facilitate the mapping of Spreadsheet fields to Requirements properties in Enterprise Architect. This facility also includes the ability to import and export fields in Tagged Values of the Requirement.

Learn more about Import and Export Spreadsheets

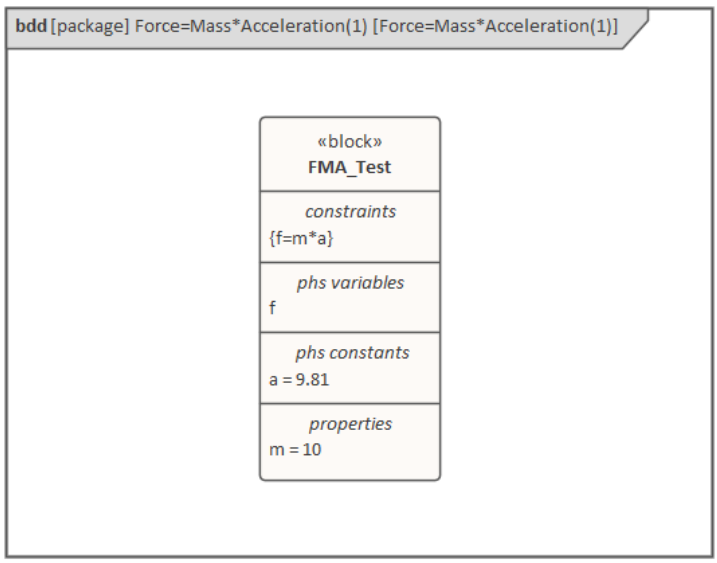
CSV Import and Export

Parametric Diagram

Getting to know the Parametric Diagram

Introducing the Parametric Diagram

The Parametric diagram is one of the Systems Modeling Language (SysML) Structural diagrams that can be used to model systems of mathematical equations. There is an assistant that will help to convert the equations into modeled elements including properties. Once the equations are modeled they can be used to create simulations using the OpenModelica integration.



Where to find the Parametric Diagram

Ribbon: Design > Diagram > Add Diagram : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = Parametric

Browser window Toolbar : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = Parametric

Browser window Context Menu | New Diagram : Perspective = Systems Engineering > SysML, Select From = SysML 1.5, Diagram Types = Parametric

Usage of the Parametric Diagram

The Parametric diagram can be used whenever you need to visualize or simulate a system of equations.

Options for the Parametric Diagram

The Parametric diagram has a number of options including the modeling of a simple equation or a system of equations.

Learn more about the Parametric Diagram

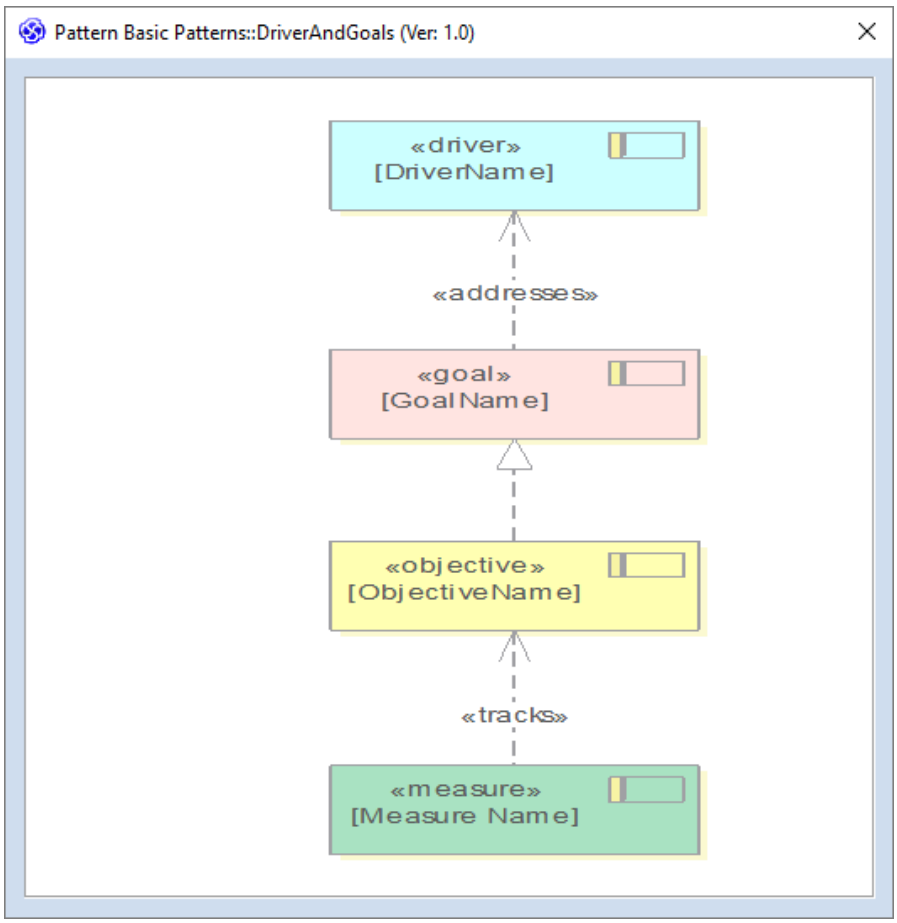
- Parametric Diagrams
- Class Diagram

Patterns

Getting to know Patterns

Introducing Patterns

A Pattern is a general reusable design solution to a commonly occurring problem within a given architectural context. Patterns are not resolved designs, but rather templates for how a problem can be solved. The concept originated in the building architecture world and was first published in a book by Christopher Alexander entitled *Design Patterns*. They were then applied to the software industry and were used extensively by the software engineering domain to solve commonly recurring software engineering problems, even though on the surface the nature of the problems seemed quite different.



Where to find Patterns

Create a Pattern:

Choose the ribbon option "Specialize > Technologies > Publish Technology > Publish Diagram as Pattern"

Use a Pattern:

In the Browser window select "Resources > Patterns > <pattern group> > Right-click on Pattern name > Add Pattern to Diagram"

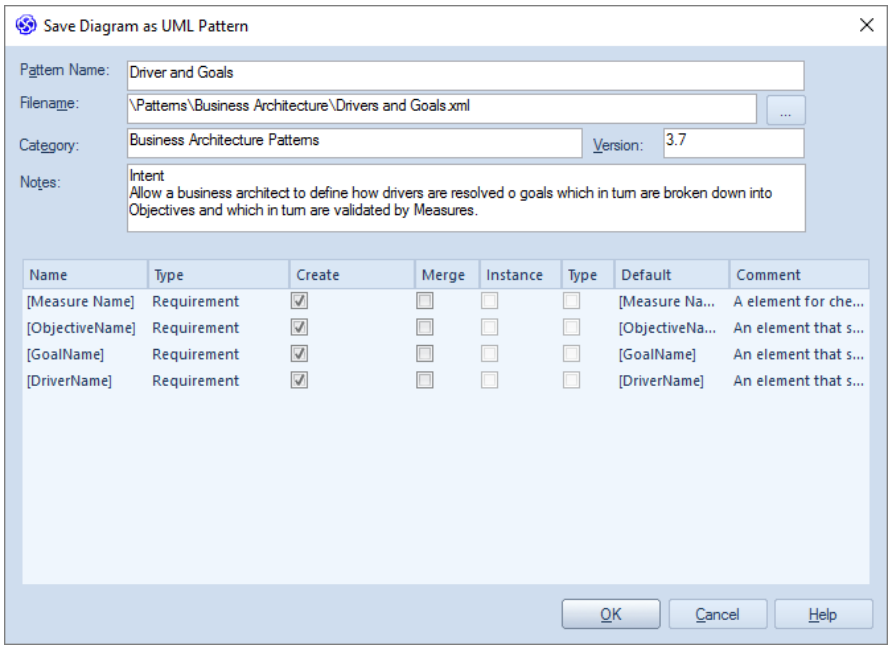
Usage of Patterns

Patterns can be applied in a wide range of situations from business to technology architecture, but are always used to apply a common solution to any number of problems or contexts that on the surface might appear quite different. Enterprise

Architect has provided a useful mechanism for mining Patterns, which means that any diagram can be published as a Pattern and then reused in the same or a different context. An example of a Pattern and its usage might be a Pattern articulating the relationship between Drivers, Goals, Objectives and Measures. An existing diagram could be published as a Pattern and then any business architecture could reuse the Pattern by simply dragging it onto an empty diagram.

Options for Patterns

Patterns are most commonly available from the 'Resources' tab of the Browser window but are also sometimes built into technologies and made available from a Toolbox page. There are a number of options available when publishing a Pattern, including the ability to describe the details of the Pattern overall and to include notes for each of the elements that make up the Pattern.

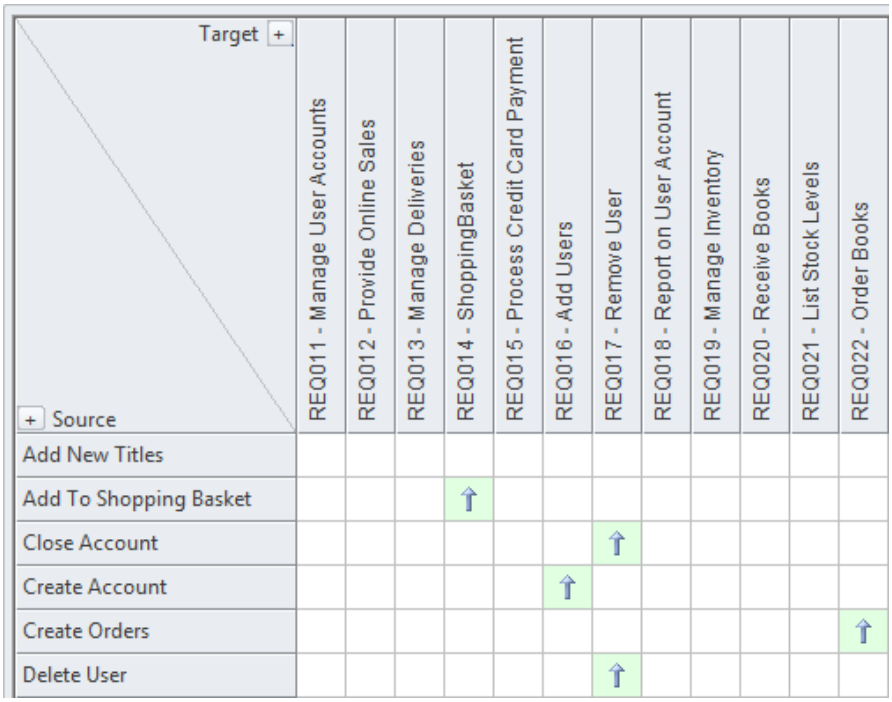


Relationship Matrix

Getting to Know the Relationship Matrix

Introducing the Relationship Matrix

The Relationship Matrix provides a visualizing compelling matrix-style view for a convenient analysis of the way that Requirements are related to each other and to other elements in the model. It can be used to view the relationships between Stakeholders and their Requirements, how Use Cases are related to Business Requirements or Functional Requirements, how Capabilities are related to Business Drivers, which Components implement a set of Requirements, and more. Any number of matrices can be defined quickly and then saved to be viewed in workshops, or included in documentation generated automatically from the model or exported to a spreadsheet file. When a matrix is created, connections can be viewed by placing the Requirements on one axis of the matrix and the connected elements on the other axis, then the cells of the matrix will indicate the direction of the relationship.



Where to find the Relationship Matrix

In the Browser window, click on a Package and select:

- The 'Resources' tab | Matrix Profiles | Right-click on a profile | Open Matrix Profile or
- The Start ribbon > All Windows > Design > Tools > Package Matrix

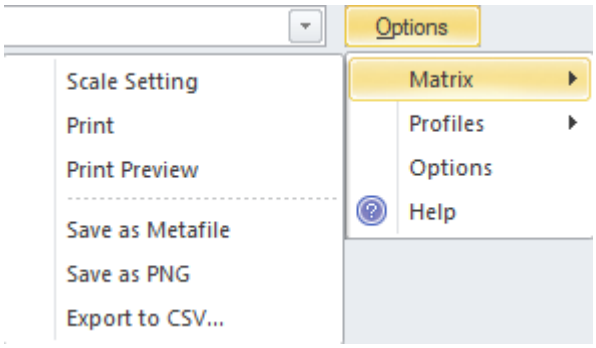
Usage of the Relationship Matrix

To display the relationships that exist between elements - such as which Requirements are realized by which Use Cases - in two Packages in a visually compelling matrix. It is useful in analyzing missing elements or relationships; for example, to determine which Requirements are not realized by any Use Case, or which Components do not have corresponding Requirements or Use Cases. It is particularly useful in workshops with Business Stakeholders who might not be familiar with seeing Requirements in Trace diagrams.

Options for the Relationship Matrix

There is a range of options that can be set for the Relationship Matrix, including saving it to the 'Resources' tab of the Browser window or to a CSV format for

opening in a spreadsheet. The appearance of the Relationship Matrix can also be altered by sorting the elements, showing an outline numbering view, and suppressing Package names. These items are available from the Options button on the Relationship Matrix.



Learn more about the Relationship Matrix

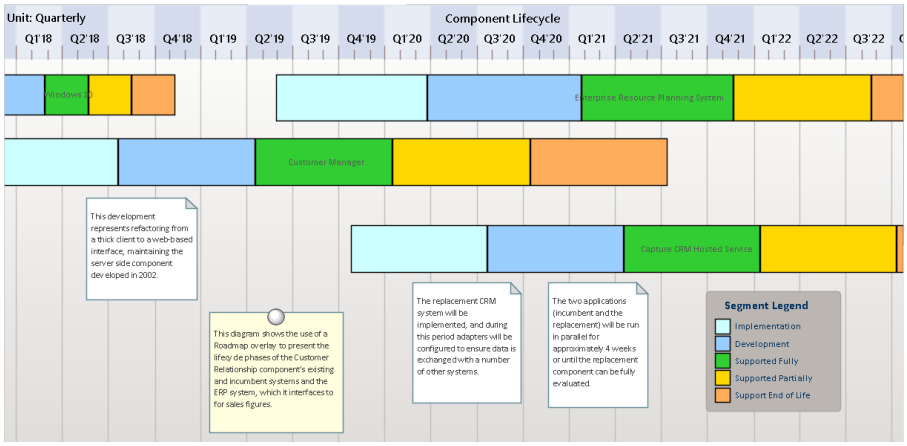
Relationship Matrix

Roadmap Diagram

Getting to know the Roadmap Diagram

Introducing the Roadmap Diagram

The Roadmap diagram is an overlay that can be applied to any diagram to describe significant phases in elements and how they change with the passage of time.



There is no restriction on the type of element that can appear on the diagram, and any diagram can have a Roadmap overlay defined. Significant user defined phases in the element's lifetime are represented by colored bars, which can be set to show duration. The colors and the phases can be configured using a Diagram Legend, which automatically applies them to the elements in the diagram. They are particularly useful in Enterprise Architecture diagrams for describing capability and application Roadmaps.

Where to find the Roadmap Diagram

Ribbon: Layout > Diagram > Roadmap

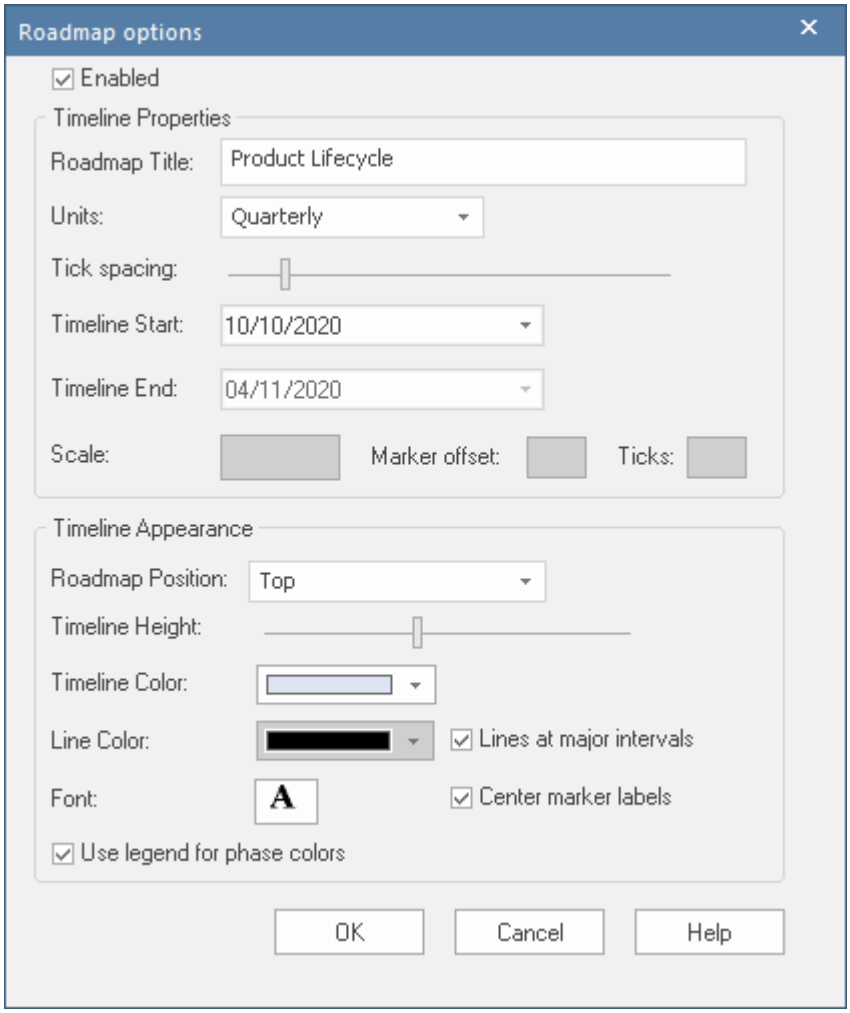
Diagram Context Menu: Roadmap

Usage of the Roadmap Diagram

The Roadmap diagram has a wide range of uses in Enterprise Architecture where they can be used to show application and capability roadmaps to Systems Engineering, where they are used to show timing in low level components.

Options for the Roadmap Diagram

The Roadmap overlay has a range of options that determine the properties of the timeline, such as the scale of the time rulers, units, their positions, and the appearance of the time line including fonts and colors. The height and position of the timeline can also be configured to suit the diagram and display.



The Diagram Legend can be configured to define the phases in the element's lifetime, to set the specification of the colored bands and more. Roadmap segments can be shown or hidden on individual elements in cases where a particular segment might not apply to one or more of the elements on the diagram.

Learn more about the Roadmap Diagram

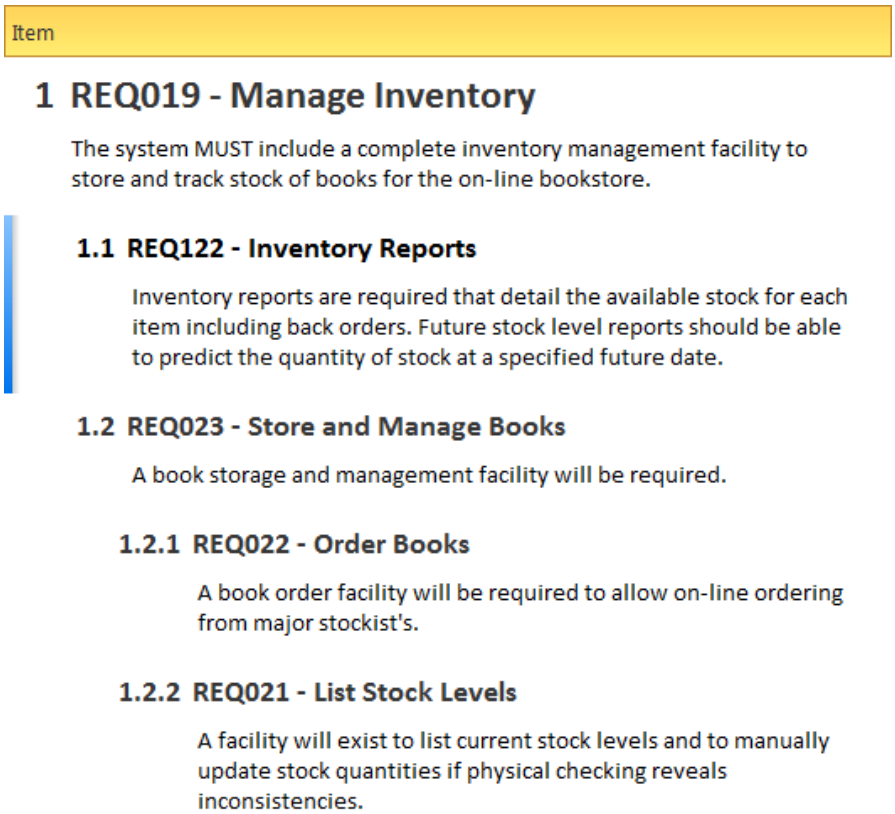
Roadmap Diagrams

Specification Manager

Getting to Know the Specification Manager

Introducing the Specification Manager

The Specification Manager is the central tool for working with Requirements; it provides an interface resembling a Word Processor or Spreadsheet tool for entering, maintaining and viewing Requirements. New Requirements can be created with names and detailed descriptions and properties such as Status and Priority can be added from drop-down lists. Existing Requirements can be viewed and managed in a convenient view, and changing them in the Specification Manager will change them in all other places in the repository such as diagrams and windows. It is the perfect tool for those analysts more comfortable working with text rather than diagrams and who are accustomed to working in a Word Processor or Spreadsheet. It has the added advantage that the requirements are part of a model and can be traced to other elements, including Business Drivers, Stakeholders and Solution Components.



Where to find the Specification Manager

Browser window | Right-click on Package | Specification Manager

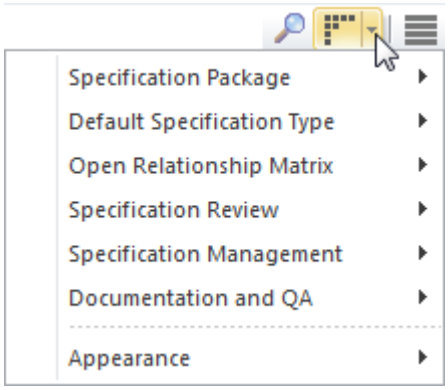
Usage of the Specification Manager

To create, view and maintain Requirements in a text based tool that resembles working in a word processor or spreadsheet. Details can be added to the Requirements and Requirement properties can be added from drop-down lists. When the Requirements are changed in the Specification Manager the changes are conveniently reflected in the Browser window and all other windows.

Options for the Specification Manager

There are a wide range of options available from the options menu, to tailor the way you use the Specification Manager. These include Level (hierarchical)

Numbering, Auto Naming, Spell Check, Documentation, Import and Export of Requirements, access to various related tools and more.

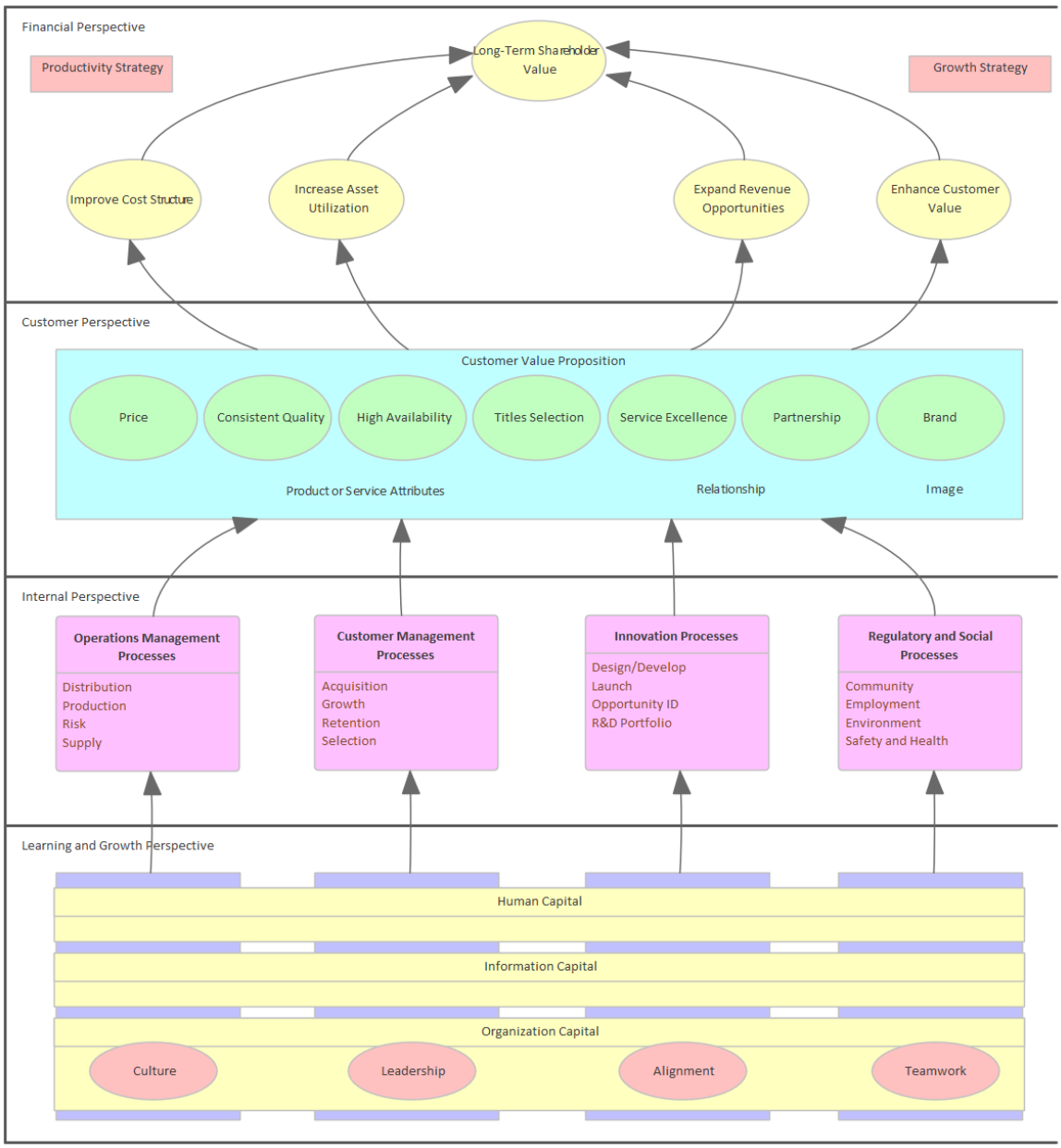


Learn more about the Specification Manager

The Specification Manager

Strategy Map

A Strategy Map is a diagram that is used to describe the primary strategic goals that are important to an organization or business team. The diagram shows four important perspectives that are the significant questions that provide the definition of a strategy. The defined perspectives are: 'Financial', 'Customer', 'Internal Business Processes' and 'Learning and Growth'. The diagram is used as a communication device to ensure there is a common understanding of the strategy, to focus organization effort and to assist with the assessment of progress.



Getting to know the Strategy Map

Where to find the Strategy Map	Ribbon: Design > Diagram > Add Diagram > Strategic Modeling > Strategy Map Browser window Toolbar : New Diagram icon > Strategic Modeling > Strategy Map Browser window context menu | New Diagram... > Strategic Modeling > Strategy

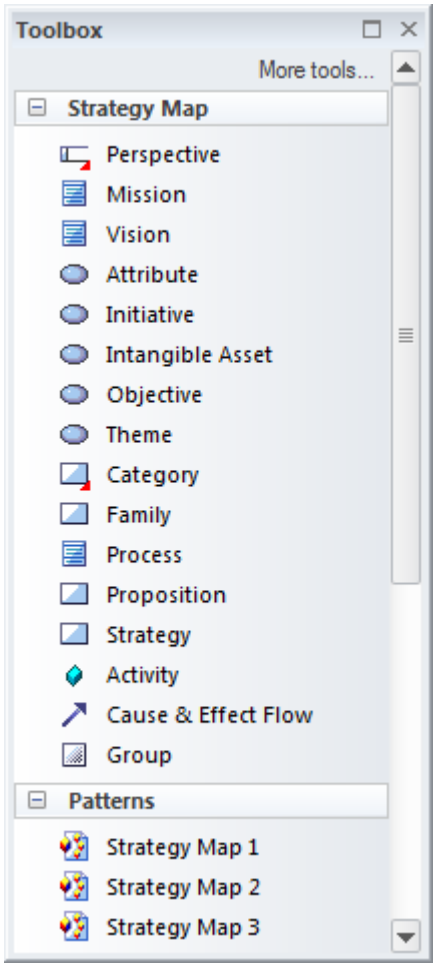
Map

Usage of the Strategy Map

The Strategy Map is used to model the key strategic goals that an organization or management team intend to achieve. Elements in each of the four perspectives can be linked to other elements in the repository to show how they could be implemented at a business, application or technology level.

Options for the Strategy Map

A Strategy Map can be created using Patterns that automatically create elements and a diagram that can be used as a starting point for the Strategy Map. There are three Patterns available, ranging from a very simple expression with a single element in each perspective to a completely worked expression with multiple elements in each perspective. A toolbox provides a range of additional elements and relationships to extend the base maps created using the Patterns.



Learn more about the Strategy Map

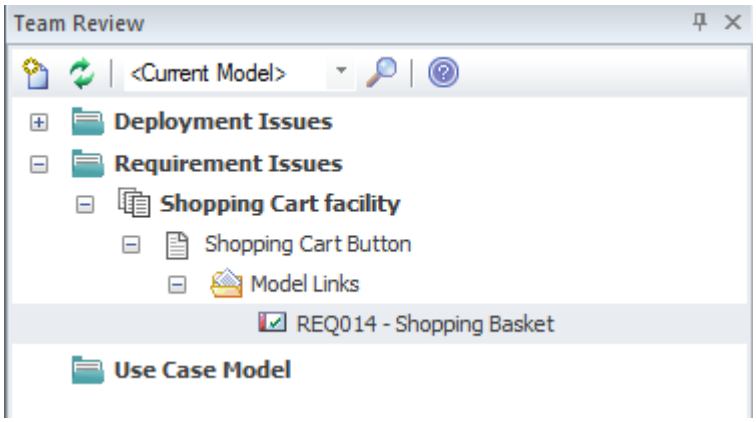
Strategy Maps

Library

Getting to know The Library

Introducing The Library

The Library window provides an opportunity for developers, modelers, customers and stakeholders to comment and provide feedback on the work in progress or at the completion of a milestone or project.



Usage of The Library

The Library feature can be used to conduct model reviews from any number of perspectives, including walk-throughs, formal model reviews, or ad-hoc reviews.

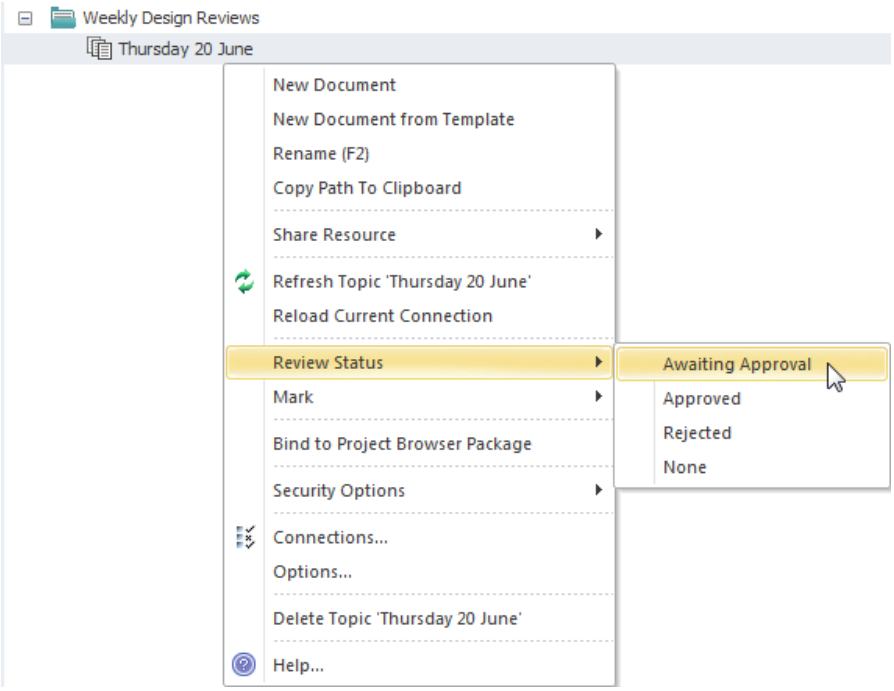
Where to find The Library

To post or view an element's discussion

Ribbon: Start > Collaborate > Model Library

Options for The Library

There is a wide range of settings available to configure the Library, available from the Category and Topic context menus, and including setting the status of the category or topic and other options. Diagrams, elements and element features can be conveniently dragged from the Browser window to create model links that can be used by team members to hyperlink directly from the Library window to these items in the Browser window.



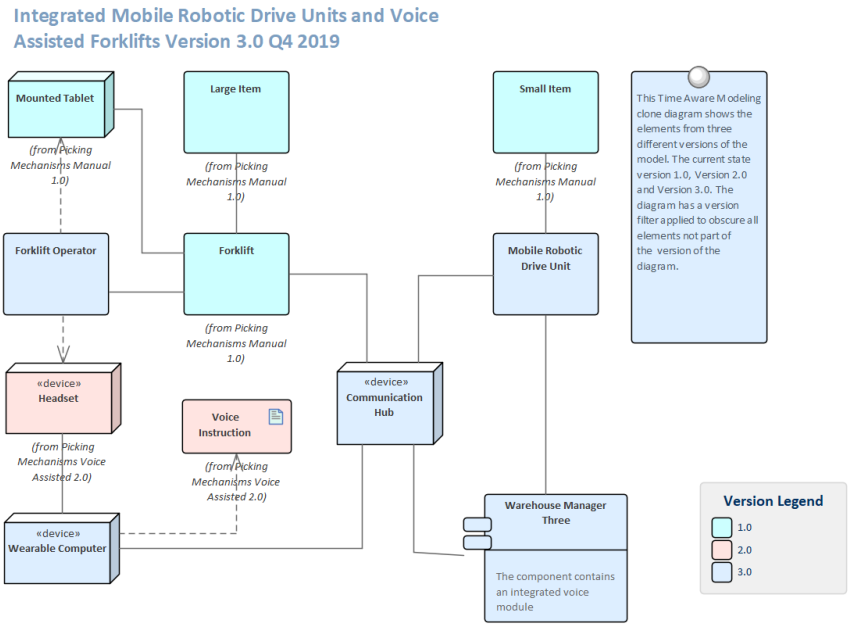
Learn more about Model Library The Model Library

Time Aware Modeling

Getting to know Time Aware Modeling

Introducing Time Aware Modeling

The Time Aware Modeling feature allows analysts, architects and others to create incremental versions of their models by providing tools that facilitate the migration of elements, diagrams and Packages through the dimension of time allowing multiple transitions or versions to be created. The baseline ('As-is', current state) models remain unaffected and any number of target ('To-Be', future state) models can be constructed for the purpose of visualization and analysis. It is particularly useful when a number of future options need to be represented and compared effectively allowing 'what-if' analysis to be conducted. Time is one of the most important dimensions in architecture as it is the substrate upon which all changes occur. Architects are aware of its importance and have traditionally created models with duplicated elements; time aware modeling allows the existing elements, diagrams and Packages to be cloned.



Where to find Time Aware Modeling

Clone Package Structure as New Version

Ribbon: Design > Package > Manage > Clone Structure as New Version

Browser window Context Menu: Clone Structure as New Version

Clone Diagram as New Version

Ribbon: Design > Diagram > Manage > Clone as New Version

Clone Element as New Version

Ribbon: Design > Element > Manage > Clone Element as New Version

Diagram Context Menu: Clone Element as New Version

Usage of Time Aware Modeling

Time Aware Modeling can be applied in a wide range of situations from business to technology models. Specifically showing how the entities represented by the

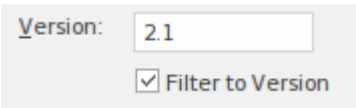
model change over time. The baseline ('As-Is', current state) models can be left unaffected while any number of target ('To-Be', future state) models can be constructed to reflect the possible evolution of the baseline models over time. There is a wide range of situations where an analyst or architect will find this tool useful; for example:

- Architects use models to document the current state and then move on to the more challenging and rewarding task of defining what the future state will look like, often in a series of transitions
- When new customers are acquired, a Business Analyst is often challenged with describing how a base product should be configured for different customer groups, resulting in different versions of the same product
- Business Strategists typically prescribe what a Capability model will look like for the organization of the future, resulting in at least two versions of the Capability model
- During Mergers and Acquisitions, Business Architects are tasked with describing the possible states of the enterprise after the takeover is complete
- Engineers are required to develop better or more efficient solutions to meet the challenges of the future, so define newer and better versions of the solution
- Testers need to be aware of different versions of a product when designing and running Test Cases
- Infrastructure engineers need to define future environments in response to performance or security concerns creating multiple versions of servers, devices and even whole facilities

All of these situations require that time is incorporated into the models so that it can be reasoned about and made explicit. Enterprise Architect's Time Aware modeling facilities can be used in all these situations to ensure that time is included as a first class citizen in the models. Time is not measured or modeled in absolute or relative terms but by representing any number of future states or differences in the form of versions.

Options for Time Aware Modeling

The Time Aware Modeling features allow a modeler to clone Packages, diagrams and elements. Most models are not trivial and Enterprise Architect provides a wide range of tools that will assist in the visualization of the models and how they change over time. The Traceability window will be particularly useful for viewing the connection between elements in the time aware models and other parts of the repository. A very useful feature is the ability to apply a filter to a diagram based on version, thus obscuring elements that are not part of a particular version.



Learn more about Time Aware Modeling

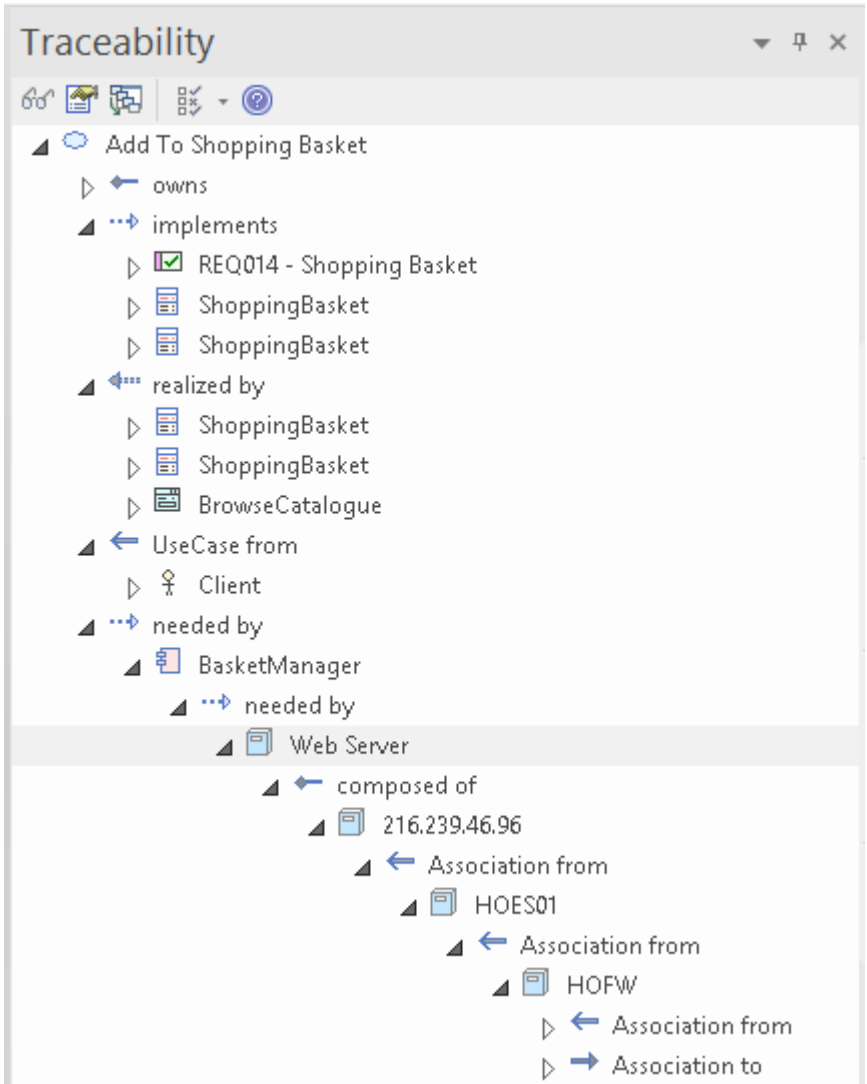
Time Aware Models

Traceability Window

Getting to Know the Traceability Window

Introducing the Traceability Window

The Traceability window provides a hierarchical view of element connections, allowing traceability to be visualized and queried as elements are traversed in the model. This tool is particularly useful because a modeler will often choose to hide diagram relationships, but by selecting an element in the diagram and viewing its connections in the Traceability window all its relationships will be revealed.



Where to find the Traceability Window

Start > Application > Design > Traceability

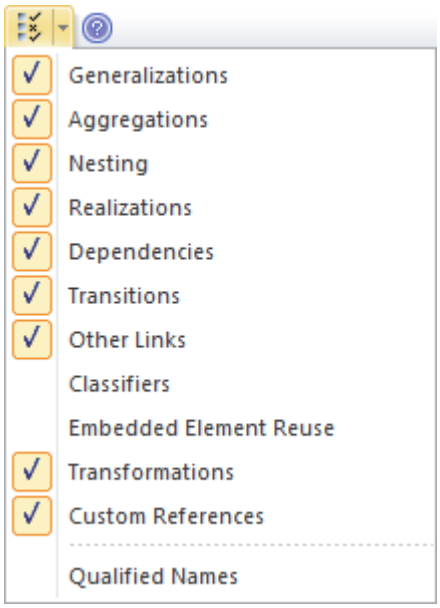
Usage of the Traceability Window

The Traceability window provides a hierarchical view of the way an element is connected to other elements in the repository, along with the type of each relationship. This window gives a complete list of all relationships that cannot be seen by viewing elements in the Browser window and that also might not appear in any diagrams. It is very useful for managing Requirements and tracing how a Requirement is related to upstream process elements such as Business Drivers and downstream process elements such as Components. It is a useful tool, enabling

newcomers to a model to gain a quick understanding of which are the important and well connected elements. Before you delete an element in the model, you should use the Traceability window to ensure that you understand that element's existing relationships.

Options for the Traceability Window

There is a series of options that restrict traceability to specified connector types; these options can be set to alter what is displayed in the window. The options are available from the toolbar at the top of the window.



Learn more about the Traceability Window

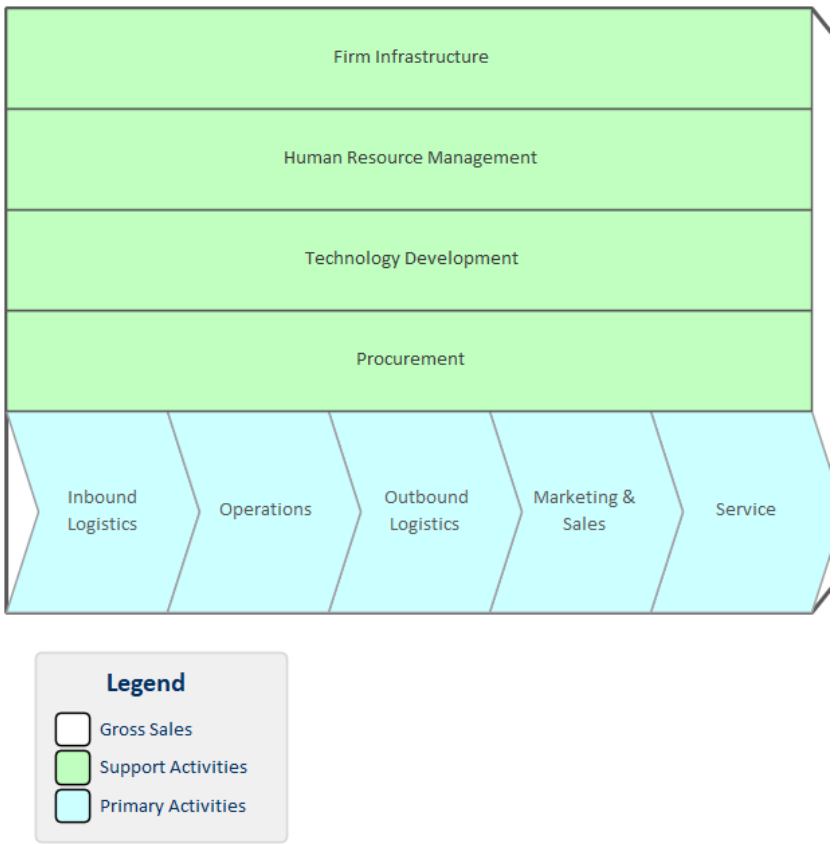
The Traceability Window

Value Chain

Getting to know the Value Chain

Introducing the Value Chain

The Value Chain is a strategic diagram that allows the primary and secondary activities in an organization to be modeled. The diagram can be created from a Pattern that adds and connects the five primary activities in a chain and the four supporting activities underpinning them.



The Gross Sales element acts as a frame for the Primary Activity and Support Activity elements.

An Analyst working at the strategic business unit level will often be asked to model the activities the business unit performs to provide value to its customers. The Value Chain is the preferred tool for creating this strategic representation of the sequence of activities that an organization performs.

Where to find the Value Chain

Ribbon: Design > Diagram > Add Diagram > Strategic Modeling > Value Chain

Browser window Toolbar : New Diagram icon > Strategic Modeling > Value Chain

Browser window context menu | New Diagram... > Strategic Modeling > Value Chain

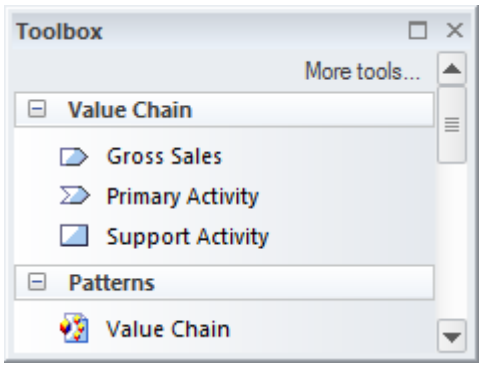
Usage of the Value Chain

The Value Chain is an important tool to assist with strategic planning allowing the whole sequence (or chain) to be understood. It also allows the chain to be broken down into its constituent activities allowing the evaluation of costs, resource and

value to be determined and potentially improved.

Options for the Value Chain

Each one of the Primary and Supporting Activities can be linked to other elements in the model including a Linked Document and elements that define benchmarks.



The Value Chain diagram (like any diagram) can be viewed as an element list which makes working with the element's properties easier.

Diagram Filters can also be used when presenting the diagrams to draw attention to parts of the diagrams.

Learn more about the Value Chain

Value Chains