

Executable MBSE Function Block Methodology Overview

www.executablembse.com

www.mbsetraining.com

Executable MBSE Function Block Methodology Overview - V2024-08-28-1111.docx

Executable MBSE Function Block Methodology Overview	1
1 Introduction	3
2 Methodological Phases	4
2.1 Concept Phase.....	5
2.1.1 Use Case Packages	5
2.1.2 Requirement Packages.....	5
2.1.3 Actor Packages	7
2.1.4 Context Packages	7
2.1.5 Signals Package	8
2.2 Requirements Definition Phase	9
2.2.1 Textual Activity Diagrams (nested under each use case)	9
2.2.2 Textual Activity Diagrams (with Requirements)	10
2.2.3 Requirement Package (for product requirements).....	10
2.3 Functional Analysis Phase (using Feature/Function Blocks)	12
2.3.1 Feature Function Package (for each use case).....	12
2.3.2 Working Copy Package (nested under Feature Function Package)	13
2.4 Logical Architectural Design Phase	14
2.4.1 System Architecture Package (for the Logical Model).....	14
2.4.2 Subsystem Package (for each Subsystem)	15
2.4.3 Subsystem Interfaces Package	16
2.4.4 Detailed Design Requirements (for each allocated function).....	16
2.5 Physical Architectural Design Phase	17
2.5.1 System Architecture Package (for the Physical Model)	17
2.6 Example Diagrams.....	18
2.6.1 uc - Use Case Diagram.....	18
2.6.2 ctx - Context Diagram.....	19
2.6.3 Table View – Simple Requirement Table	20
2.6.4 Table View – Context Diagram Flows Table	21
2.6.5 act - Textual Activity Diagram	22
2.6.6 act - Textual Activity Diagram (with Requirements)	23
2.6.7 req – Product Requirements in a simple requirements table.....	24
2.6.8 req - Action to Requirement traceability for use case	25
2.6.9 bdd – Engineer detailed design requirements feature block	26
2.6.10 req – Requirement to Function Block relation table for a Feature.....	27
2.6.11 bdd – Logical Architecture Diagram (including subsystem and function blocks)	28
2.6.12 lbd – Logical Architecture and Interfaces	29
2.6.13 req – Requirements Diagram – System example	30
2.6.14 bdd – Physical System Architecture (with allocation).....	31
2.6.15 tablel - Physical to Logical allocation relation table.....	32
2.7 Key themes.....	33
2.7.1 Using operations vs function blocks to model functions.....	33

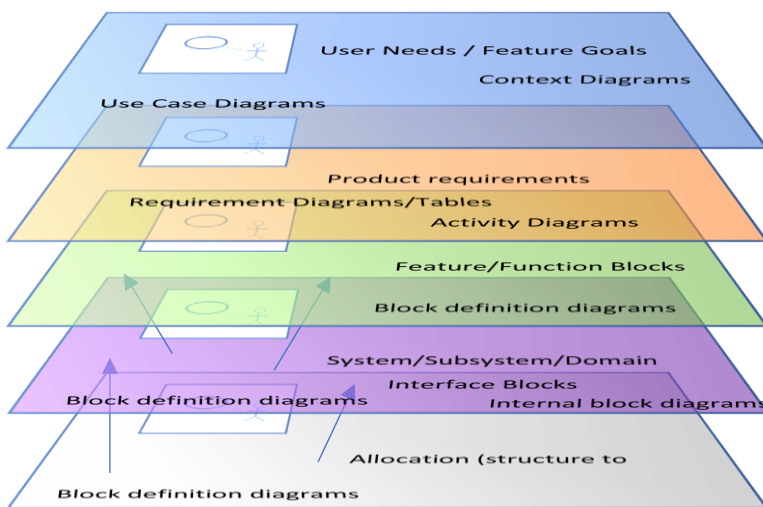
2.7.2	Requirements vs modeling elements as formal hand-off.....	33
2.7.3	Use case driven vs functional decomposition.....	33
2.7.4	White box vs black box modeling	33
2.7.5	Logical vs physical modeling	33
2.8	FAQs	35
2.8.1	Can my model have multiple System Blocks?	35
2.8.2	Can I show function blocks on a sequence diagram?	35
2.8.3	What about Requirements Management tools?	35
2.8.4	What benefit is use case modeling?	35
2.8.5	Does the same person do all the phases of the model?	36
2.8.6	Do you have to maintain the activity diagrams?	36
2.9	New Term types	37

Also important for this are relationships between the model and requirements, particularly where it is textual requirements that are the hand-off to downstream design and testing teams.

2 Methodological Phases

The Executable MBSE profile presents the ability to decompose systems in two separate ways, structurally and functionally, together with ways to relate the structural and behavioral features of a system together.

The phases are layers that have defined inputs and defined outputs, with a modeling method that can be used to perform the transformation between the input and the output. For some projects it may be appropriate to upper layers. For larger projects different teams work in distinct phases, handing off between them.



Concept Phase

Requirements Definition

Functional Analysis

Logical Architectural Design

Physical Architectural Design

2.1 Concept Phase

The concept phase is about understanding the operational needs of the system, irrespective of how it is implemented. Related to this are the binding constraints that the solution space will need to consider. This includes initial scope of user needs and concept of operation requirements.

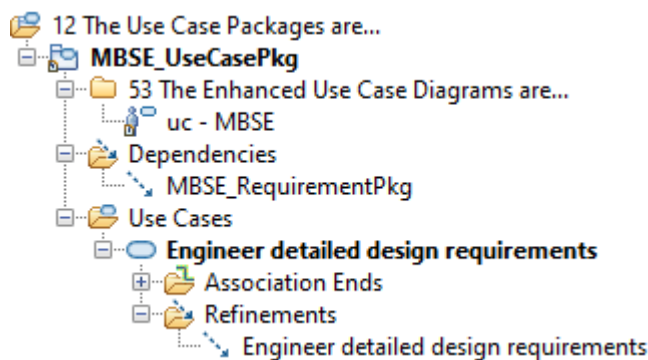
Light-weight modeling is used in the method to:

1. Ensure that diagrams are easily understood by domain experts with little or no training.
2. Ensure that diagrams can be drawn by experts who understand the customer's needs with reduced training using only a small subset of the modeling language.

The profile introduces several new term packages for this purpose. The new term package types create a jigsaw of different type pieces that can be stitched together to form a model with different people working on different pieces concurrently. Initially there may be one of each of the below package types in a model.

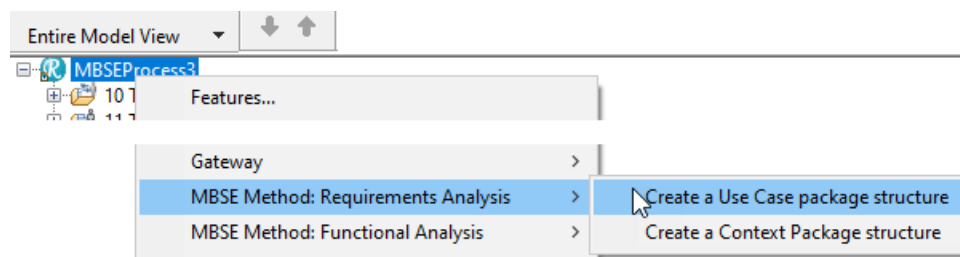
2.1.1 Use Case Packages

Use Case Package: Used to create use case diagrams and use cases with nested activity diagrams that show the steps of use cases and how they relate to requirements and actors (an actor is a UML entity that is used to capture roles played by elements external to the system).



See example uc - Use Case Diagram.

A helper is provided by the profile to create use case package structure with a use case diagram template. The helper provides a form that allows the modeler to choose to also create an associated model structure.



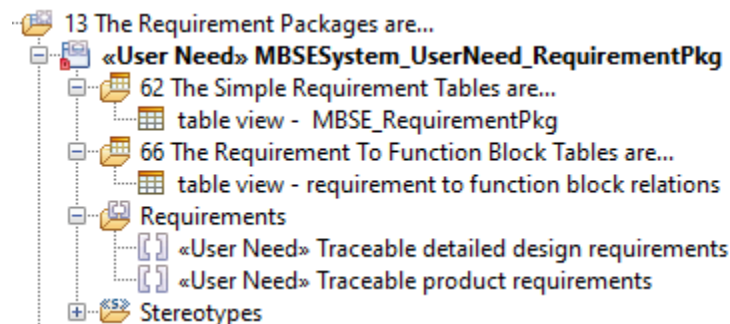
A project could have more than one use case package.

2.1.2 Requirement Packages

Requirements Package: A library package that groups together related requirements that are part of the model. We keep these in a separate package because they are considered cross-cutting

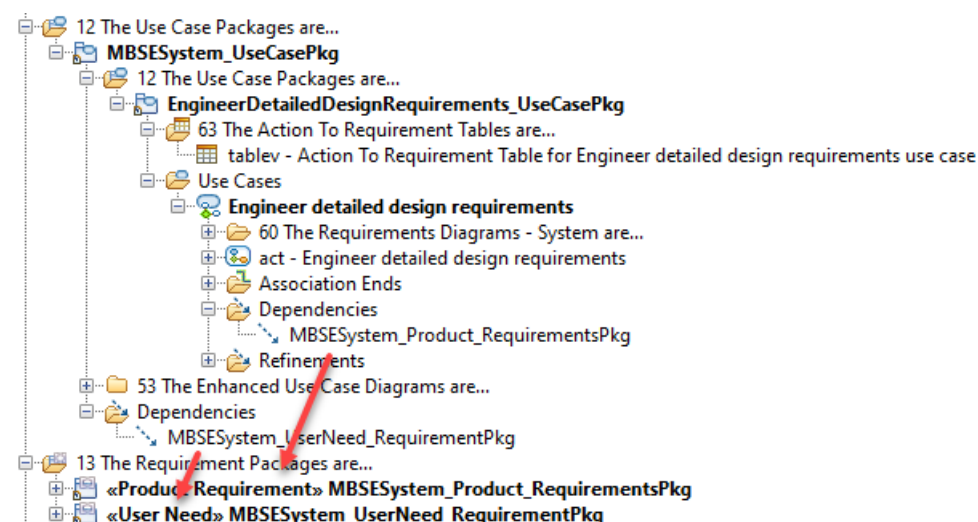
constructs that would be shown on many different diagrams. The package also allows us to more easily create tables that show the information in a tabular and textual format.

The package can have relation or element tables that show the relationships that the model has to the requirements such as the **Table View - Simple Requirement Table**



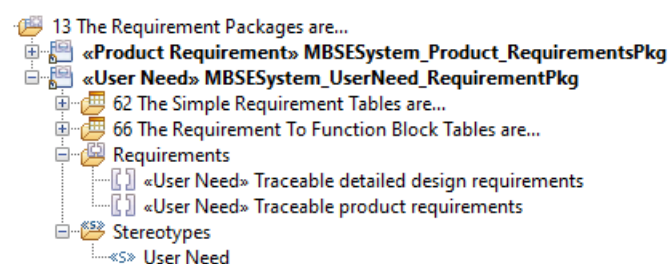
See example: Table View – Simple Requirement Table.

A helper provided by the profile will automatically move requirements created on a diagram into a requirement package based on the presence of a dependency to a requirement package being present in the model hierarchy of the diagram.



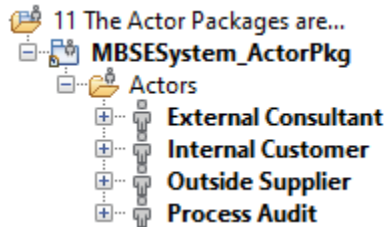
This can speed up modeling tasks by reducing overhead tasks with moving things in the browser. By hierarchically placing dependency relations the requirements in an activity diagram could be flowed to the product requirements package, whereas those on the use case diagram would flow to the user needs requirement package.

The helper will also automatically apply a stereotype to the requirement, if the requirement package has a stereotype applied to it, thus enabling speedy characterization of the requirements in a particular level.



2.1.3 Actor Packages

Actor Package: A library package that contains actors used in other packages such as context and use cases packages. Actors relate to use cases using associations to show that they participate in the use case. The actors in this package can also be used to type actor usages on a context diagram.

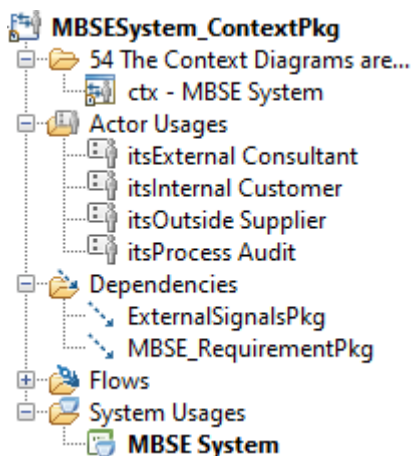


Tip: Before drawing new associations on diagrams, ensure you “complete relations” to populate ones that exist, to avoid duplication of relations.

Actor: Represents an entity outside of the system that gains value from it. The Actors are not part of the system, they are usually things that the system interacts with, e.g., Operator or a peer system. Relates to the environment the system directly interfaces with.

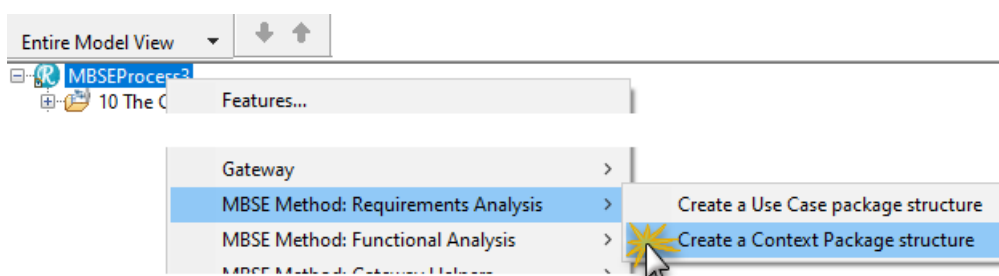
2.1.4 Context Packages

Context Package: Used to create Context Diagrams that show the flow of information to/from the system using **Context Diagrams**. Context diagrams are a specialization of internal block diagrams introduced by the profile. They use only a small subset of the language and make it easy to create events to represent elements on flows.



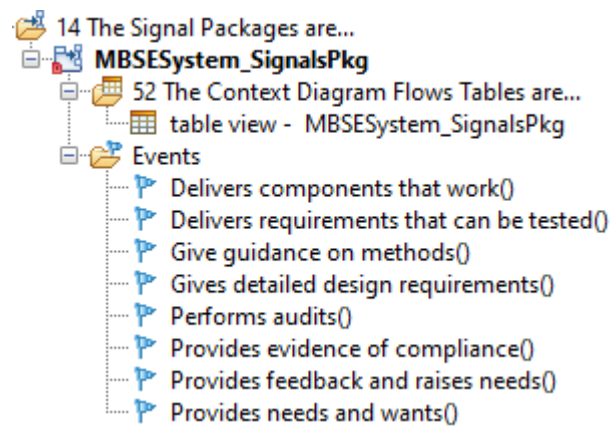
See example: ctx - Context Diagram.

A helper is provided by the profile to create a context package structure using a form that allows the modeler to choose to also create an associated model structure:



2.1.5 Signals Package

Signals Package: A library package that groups together the events used by flows in **Context Diagrams** and associated relationship tables such as a **Table View – Context Diagram Flows Table**.



See example: Table View – Context Diagram Flows Table.

Tip: Packages like signal, requirements, and actors act as libraries for elements. If one of these library packages is only used by one other package, it should be owned by that package. Conversely, if a library package has elements used across the model, it should be owned at the root of the project.

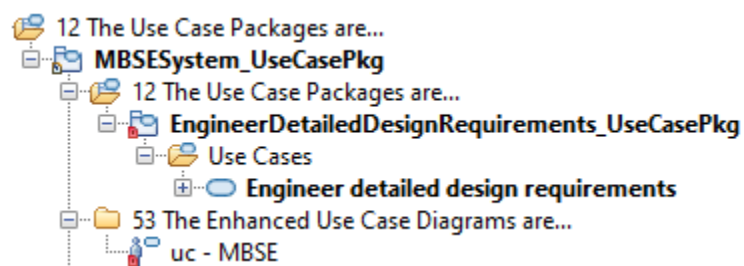
2.2 Requirements Definition Phase

This Requirements Definition phase is based on a use case driven approach to ensure a complete set of system-level product requirements driven from the user needs. The Requirements Definition phase is also an integration step. There may be multiple places in different use cases or steps where the same behavior is needed, and we would expect a single requirement to be used for both.

Use case: A use case is a set of steps including alternate flows that provide value to one of more actors. A use case driven approach can form a basis for reviewing the concept of operations with stakeholders, and a thorough source for writing system-level functional requirements.

Writing use case steps is a systematic way of creating functional product requirements. This approach acts as either an end point (if only black box analysis is needed) or as a precursor to the functional analysis phase where the goal to have functional requirements able to act as the formal hand-off; in a seam of requirements at system product level that will trace to the functional model allocated to subsystems in the white box architecture.

Use cases groupings are a way of allocating work within a team. If use cases are well segregated they can be developed concurrently by different modelers. Where their inter-dependencies mean they need developing concurrently, use cases may be grouped together into a cluster of use cases and given to a modeler who has singular oversight.



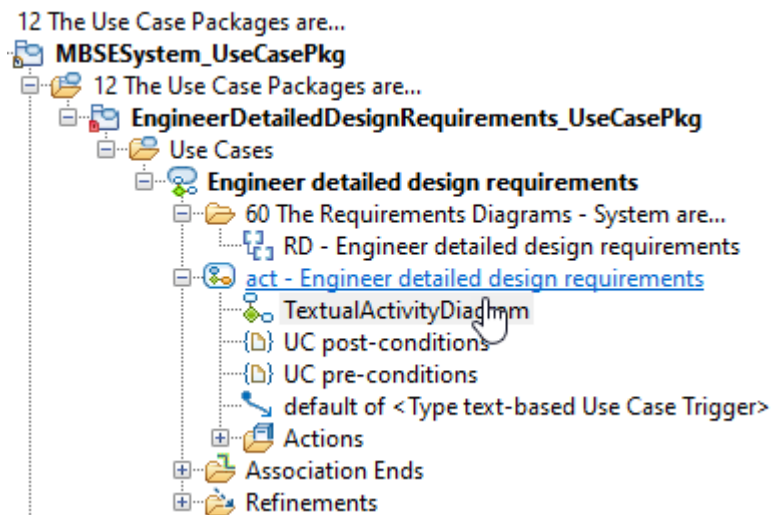
To avoid conflicts, we move the use case into its own use case package to enable the modeler to take ownership and lock particular packages that they own. For example, the “Engineer detailed design requirements” use case has been moved into a nested use case package in the browser, shown above, so that it can be worked on independently and its package locked (the latest profile provides some automation to assist this).

2.2.1 Textual Activity Diagrams (nested under each use case)

Writing a use case is akin to authoring an end-to-end story about how the system is used to achieve an outcome of value. They are best used to represent a complete end-to-end use of the system, rather than discrete needs of the system, as they play the upstream role of understanding and expressing the big picture in a way which can engage both end-users and system implementors, and as a steppingstone for the creation of the functions and test cases needed.

UML/SysML does not specify how to write the steps of a use case. Activity diagrams are chosen by this method because they have a flow-chart-like syntax that can be easily understood by engineers, with little or no training in SysML (Systems Modeling Language). Importantly, sunny and rainy day scenarios can be analyzed on the same diagram meaning they are good at capturing a big picture on the same diagram.

To model the steps of a use case we create a Textual Activity Diagram underneath each of the use cases, resulting in one activity diagram underneath each use case.



See example: act - Textual Activity Diagram.

2.2.2 Textual Activity Diagrams (with Requirements)

Use case steps are not the same as requirements, but form a good basis to write functional product requirements. Review the use case steps with stakeholders before creating requirements, to make sure you're on the right path.

Requirement: A textual statement of need written at an appropriate level. In Rhapsody the underlying type for requirements is a note, hence they can be dragged on to any diagram, and traceability to them.

Creating requirements on an activity diagram, and relating requirements to the actions, provides a systematic way to generate functional requirements that can be carried forward supported by right-click menus provided by the profile.

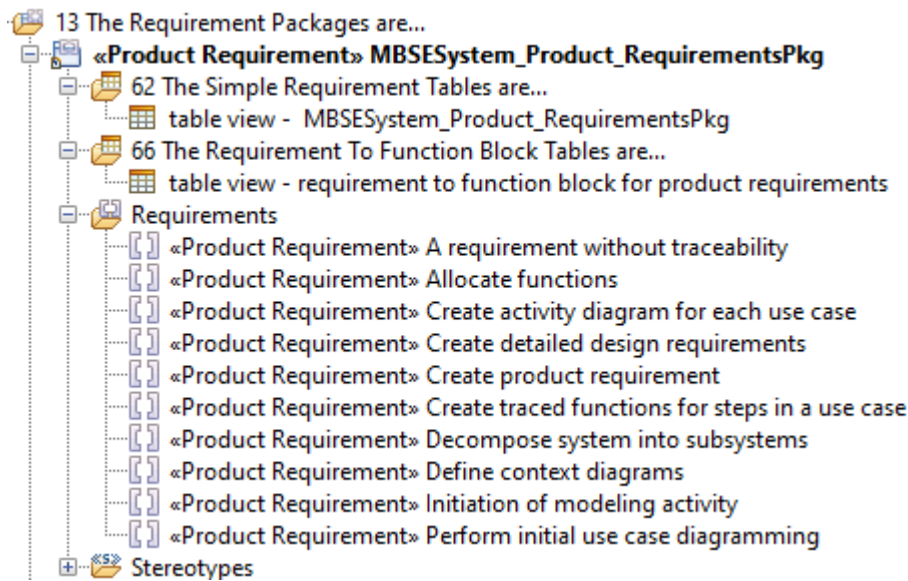
See example: act - Textual Activity Diagram (with Requirements).

We will be using the activity diagram as a steppingstone to create a functional model, and the requirements on it are the hand-off, to ensure completeness of that transformation.

Product requirements need to align their text with the actions and vice versa for the systematic process of using an activity diagram as a starting point to create traced functions to work. If requirements do not exist then it is better to create new requirements that match, than go with requirements that do not match.

2.2.3 Requirement Package (for product requirements)

We need to get the requirements at system level complete and reviewed and ensure traceability to the model to ensure completeness.



We should review these requirements with stakeholders at this point to improve their quality. This will assist with buy-in and reduce risk of downstream re-work by correcting issues before work is commenced. Tables can assist with this.

See example: req – Product Requirements in a simple requirements table.

2.3 Functional Analysis Phase (using Feature/Function Blocks)

At this point, it is necessary to make the choice about whether to represent the results of the functional modeling as operations or function blocks. We can think of this as taking the model from a 2D world to a 3D world. In a 2D world things exist on one diagram, in a 3D world the same thing exists in multiple behavior and structural views.

The profile supports two different package types for this, depending on whether functions will be modeling using operations on sequence and state machine diagrams, or function blocks on block definition diagrams and internal block diagrams. This section focused on the latter approach which introduced two new types of blocks.

Function Block: Defines a behavioral capability needed of a subsystem. As a minimum it would represent at least one functional requirement.

Feature Block: Defines a behavioral capability needed by a system. The block may be traced to a corresponding use case using a dependency. A feature block can be seen as a filtered view of a subsystem functions in a system. Initially the functions may be shown without form as parts of the feature (a black box view). However, in later stages the features may use generalization relations to inherit subsystems and their functions from a system block (a white box view).

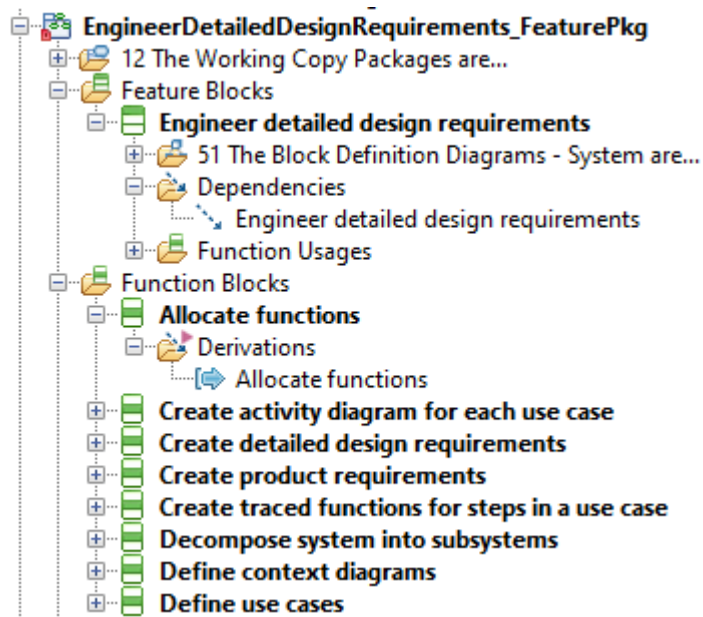
Feature blocks are a distinct type in the metamodel from function blocks to support the separation of use case modeling (which may be done by a non-modeling expert) from functional modeling (which may be done by a modeling expert).

The purpose of functional analysis is to provide modeling elements from the black box model of the system (which captures the product requirements) which can be carried forward into the white box model of the system (which decomposes the system into subsystems and captures the detailed design requirements for each). There is a systematic approach provided by helpers in the profile to transform the steps on an activity diagram into function blocks that trace to the same requirements.

We want the white box model to trace to the product requirements. Therefore, an optional input to this step is the use case activity diagram that contains actions with corresponding product requirements written at the same black-box level.

2.3.1 Feature Function Package (for each use case)

Feature Function Package: This enables creation of a function breakdown structure using Feature Blocks and Function Blocks on a block definition diagram – system diagram. Feature Blocks are externally perceived behaviors of the system (and might notionally be created for each use case). Function blocks are behaviors needed by the system to achieve the features.



See example: bdd – Engineer detailed design requirements feature block.

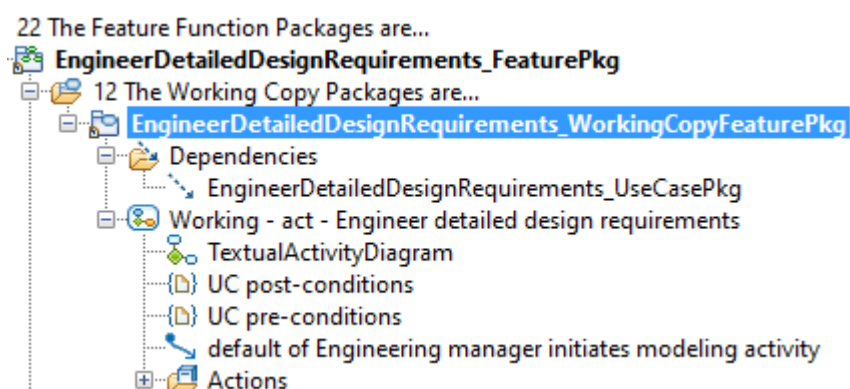
The function blocks have derivation relations to the product requirements at this point. There is often a one-to-one correspondence. We can use the relation table called Requirement to Function Block table view provided by the profile to check and review this coverage.

See example: req – Requirement to Function Block relation table for a Feature.

2.3.2 Working Copy Package (nested under Feature Function Package)

Working Copy Package: This optional package is nested underneath each of the above packages to provide a container for creating a copy of the activity diagrams being converted into functions. This is used to follow a systematic approach to creating traced functions using helpers in the profile.

The Working Copy package has a dependency relation to the use case package it is drawing its diagrams from.



The working copy package supports concurrent development of the original activity diagram by a different modeler than the one creating the function blocks, with periodic hand-off. Optionally, the upstream model may even be done in a separate model and added by reference. The benefit of this is that the modeler who is maintaining the activity diagrams might be deemed the domain expert, whereas the person creating the functions needs to understand more about modeling but does not need to be a domain expert, as the requirements have already been written and reviewed, i.e. the

functional analysis step is about a model-to-model transformation. Taking the model that was provided and converting it into a different form which can be carried forward into the architectural design phases.

The working copy's activity diagrams are considered temporary, hence may be removed from the model once the transformation has been completed. Right-click menus are provided on the package to perform the copy.

2.4 Logical Architectural Design Phase

When taking the function block approach several new term block types are introduced to support architectural design.

System Block: System Blocks represent the element that is the focus for the project, and the boundary for the purposes of use case modeling. Usually there should be one System Block per model, since this is what will define what is inside the system (blocks) and what is outside (actors).

Subsystem Block: Subsystem Blocks represent structural elements that define the properties of parts of the system. They normally represent a system sub-entity with a separate requirements specification. Implicit in this, therefore, is that the subsystem has a clear boundary and ownership. A subsystem may be logical or physical in nature. Subsystems can be decomposed into other subsystems to any level, although it is recommended to limit use of this capability and consider instead that the goal of the system architecture is to decompose one level such that each component knows what role it is playing in the overall system and its associated interfaces.

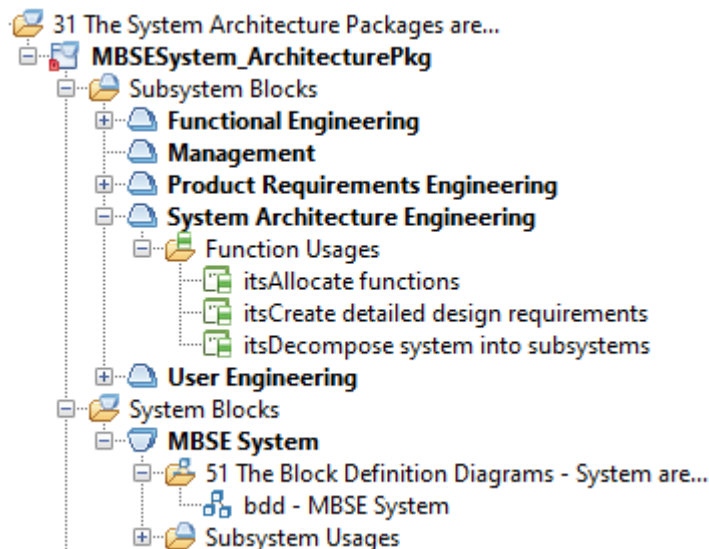
Context Block: The Context Block is at the level above the System Block. It comprises of Actor Usages and System Usages, framing the context of the system. It provide the option to create an ibd of the system context where the actors are parts of the system – a more complex, more SysML specific but potentially suboptimal way of drawing a context diagram.

The above block types play distinct roles with respect to the system boundary. Related to this are two views of the system: White box vs Black-box.

2.4.1 System Architecture Package (for the Logical Model)

System Architecture Package: Used to create a **Block Definition Diagram – System** that details the decomposition of a System Block into Subsystem Blocks.

Related to this are the logical subsystems that will perform the functions identified in the feature function package. The system architect performs this allocation to subsystems using the composition relationship.



See Example: bdd – Logical Architecture Diagram (including subsystem and function blocks).

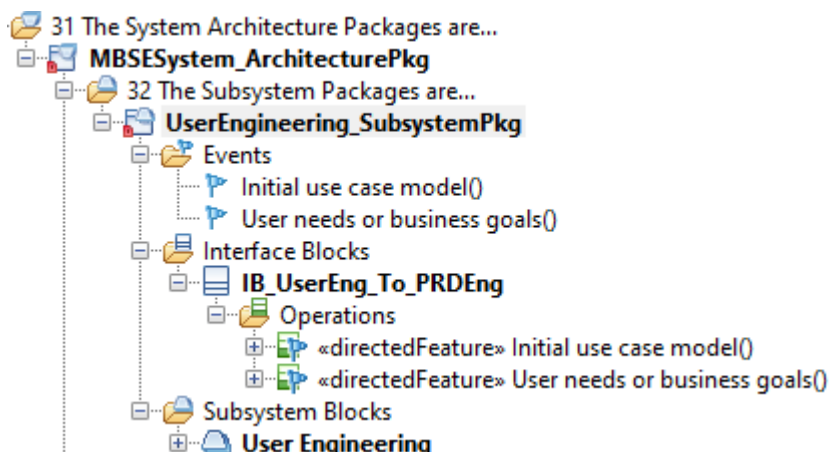
Underneath each System Block one would expect an **Internal Block Diagram – System** that shows the internal connections between the parts of the system. The difference is that the ibd shows the interfaces between the subsystems that comprise the system.

Remember internal block diagrams are always created underneath the block to which they pertain rather than at package level, and they show the usages of the blocks. Multiple internal block diagrams may show different views of the same functions, for example, what is the subsystem interaction and the functions required to implement a particular feature.

See Example: Ibd – Logical Architecture and Interfaces.

2.4.2 Subsystem Package (for each Subsystem)

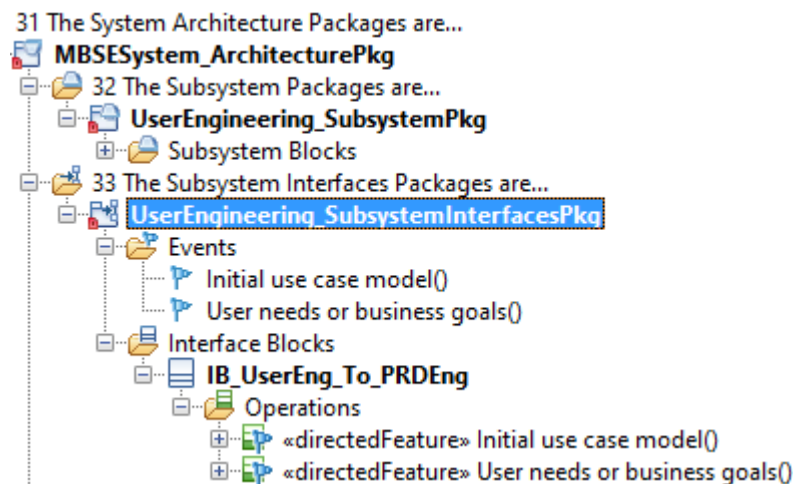
Subsystem Package: The profile allows Subsystems to be decomposed into other subsystems although some caution is suggested as this may not be necessary and it adds to complexity. If there is a hierarchical decomposition through multiple levels, then the Subsystem Blocks can be moved into their own Subsystem Package.



This helps segregate parts of the model, which can be valuable when there are multiple levels of decomposition and modeling being performed concurrently by different modelers.

2.4.3 Subsystem Interfaces Package

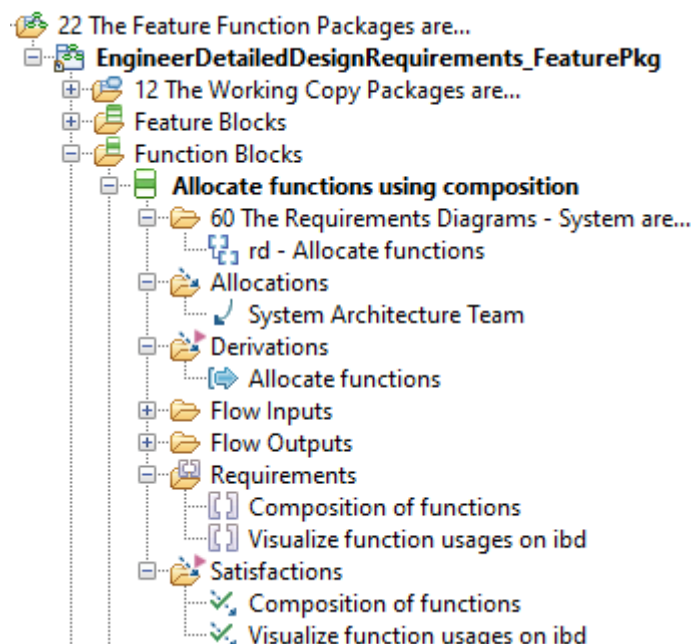
Subsystem Interfaces Package: Library package for elements that define the interfaces of elements irrespective of their implementation and typically houses **events** and **interface blocks**. This enables the interfaces to be shared between models, or related element or relations tables to be more easily created.



The profile assumes the use of Rhapsody events for discrete signals and flows. This is because they can be used in flow ports and interface blocks, and used on timing diagrams, sequence diagrams and state machine diagrams, i.e., they have maximum flexibility to link to things in the same model.

2.4.4 Detailed Design Requirements (for each allocated function)

We can think of the function as a heading in the subsystem requirements specification under which the functional and non-functional requirements related to the behavior are articulated as textual requirements. To support this, the profile allows for a requirements diagrams to be created underneath the function block, which acts like a heading in a document would.



See example: req – Requirements Diagram – System example.

2.5 Physical Architectural Design Phase

The approach to physical decomposition of the system uses the same new term block types and block definition diagram that were introduced to support logical architectural design.

System Block: System Blocks represent the assembly block that represents the top of the system. It should have a «Physical» non-new term stereotype applied.

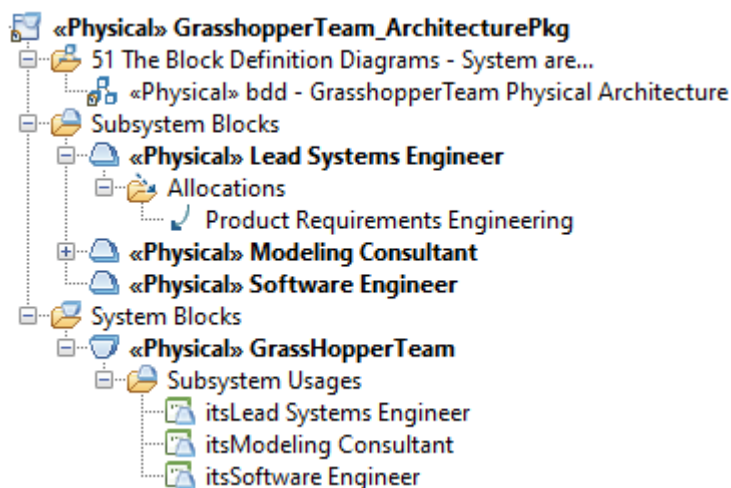
Subsystem Block: Subsystem Blocks represent structural elements that comprise parts of the system. They normally represent a system sub-entity with a separate requirements specification. It should have a «Physical» non-new term stereotype applied.

Subsystems can be comprised of other subsystems to any arbitrary level.

2.5.1 System Architecture Package (for the Physical Model)

System Architecture Package: Used to create a **Block Definition Diagram – System** that details the decomposition of a System Block into Subsystem Blocks.

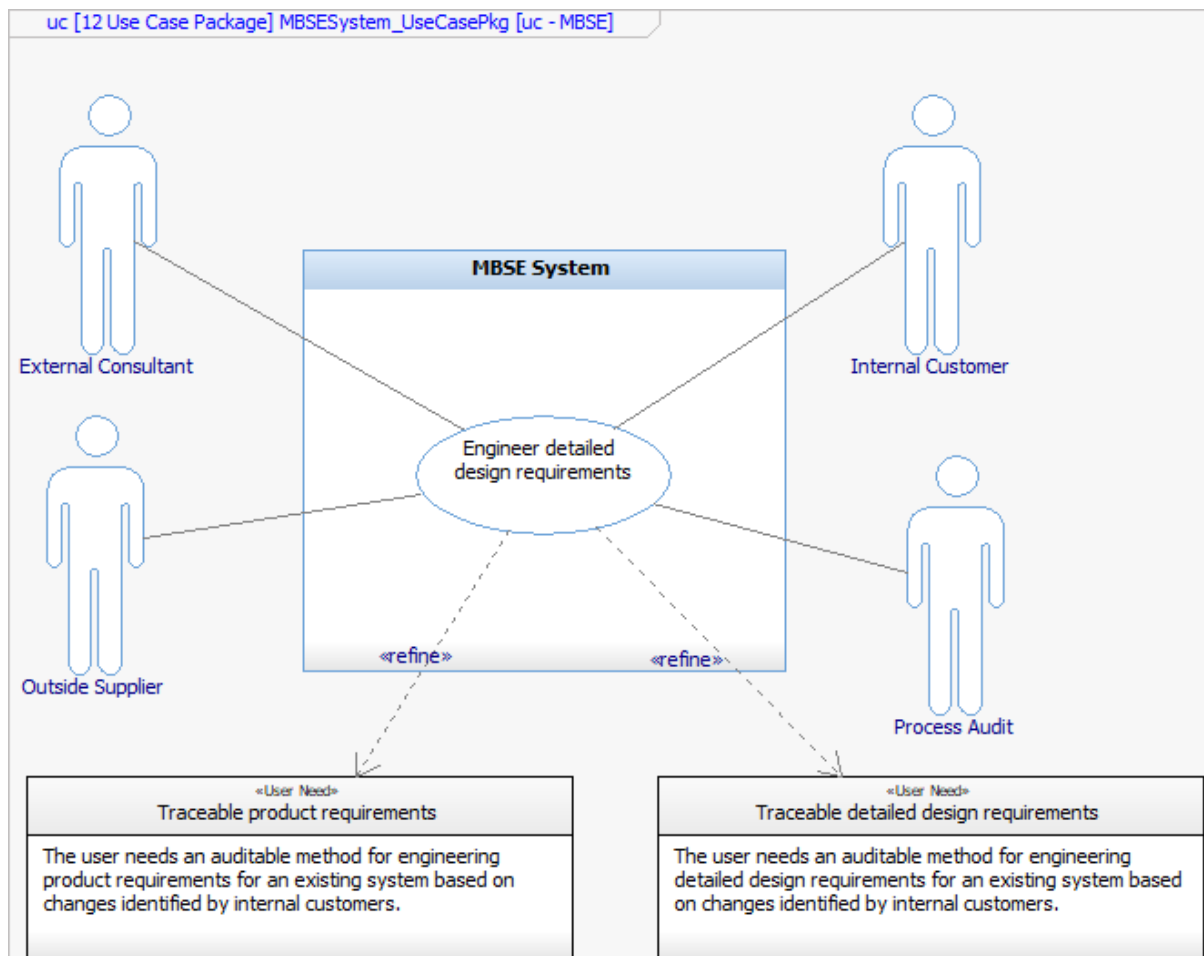
Related to this is the use of allocation of physical subsystems to logical subsystems.



See bdd – Physical System Architecture

2.6 Example Diagrams

2.6.1 uc - Use Case Diagram

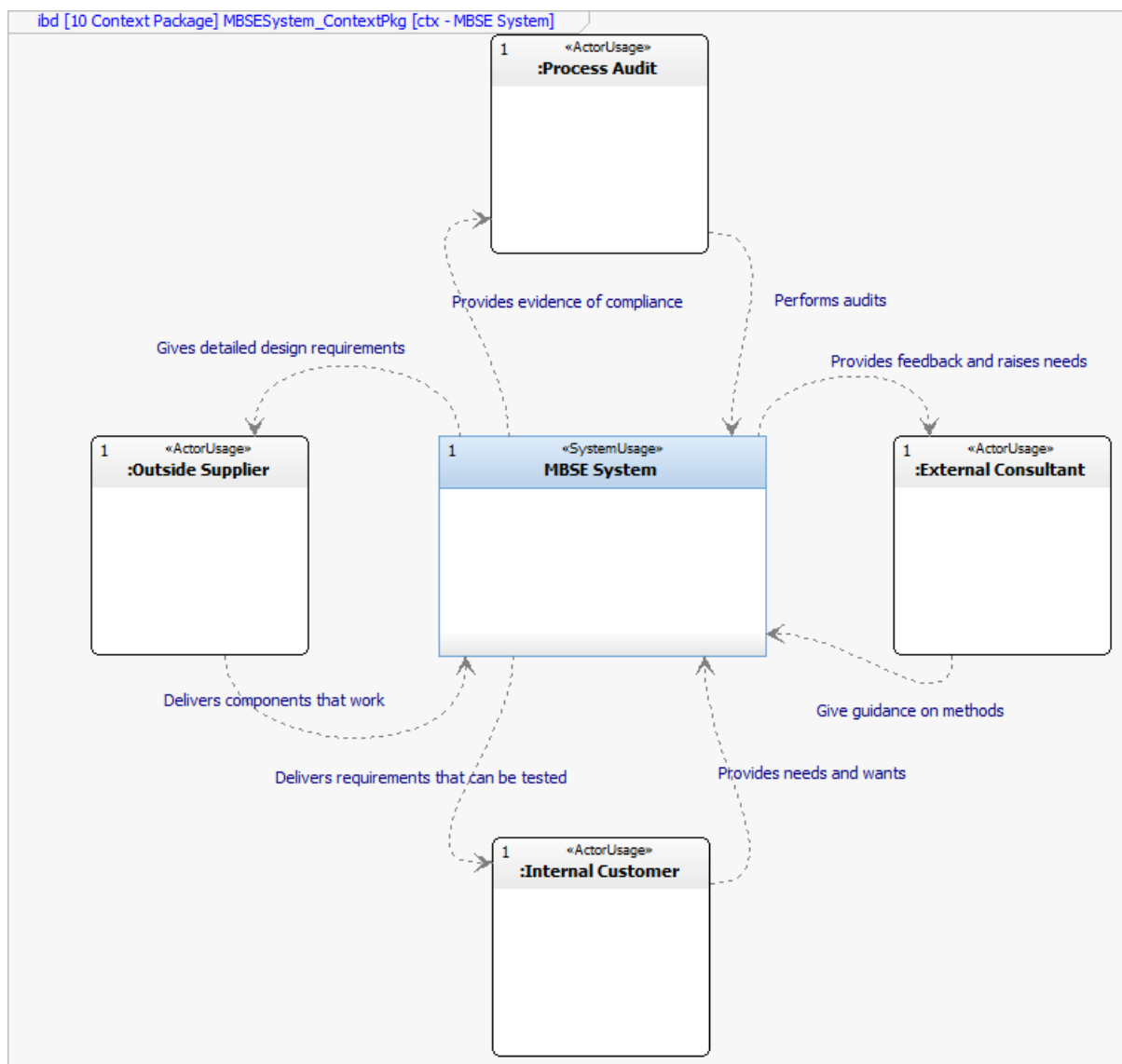


The above diagram shows a simple use case diagram (uc -). The uc is important in defining the boundary of the system of interest for the model.

The diagram does not define the use case steps, it provides a high-level functional context diagram. Often the use case will be related to other diagrams in the model such as a nested activity diagram that describes the steps. The profile listener provides double-click functionality to open up related diagrams, or create a new activity diagram. Requirements shown on this diagram align with the user needs or system constraints. If requirements are important then having them shown can be useful. Requirements and their traceability exist, where they are shown on the diagram. In this instance a user-defined none new term «User Need» stereotype has been applied to them.

See sections: Use Case Packages, Requirement Packages.

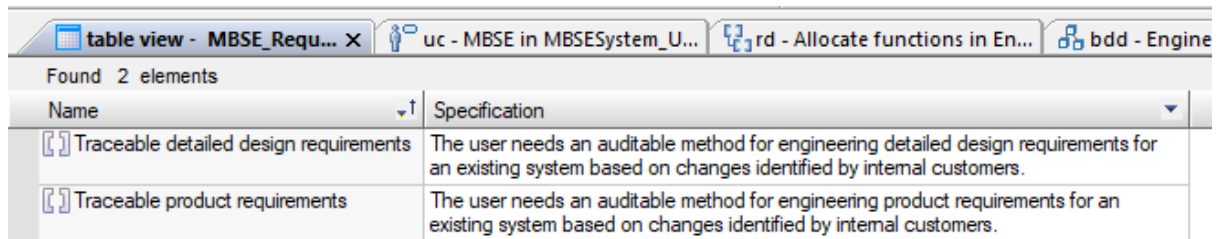
2.6.2 ctx - Context Diagram



The above shows a simple context diagram (ctx -). The graphical notation is purposely simplified to make it easy to draw flows with events.

See sections: Context Packages.

2.6.3 Table View – Simple Requirement Table

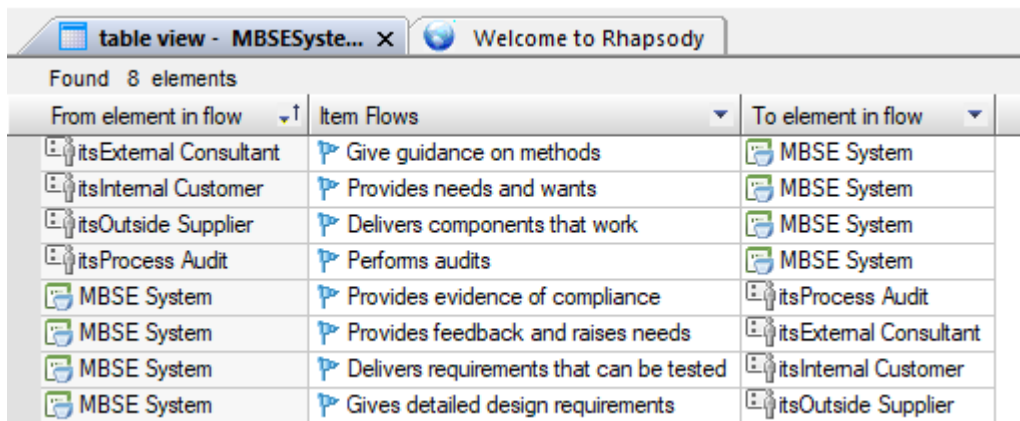


Name	Specification
Traceable detailed design requirements	The user needs an auditable method for engineering detailed design requirements for an existing system based on changes identified by internal customers.
Traceable product requirements	The user needs an auditable method for engineering product requirements for an existing system based on changes identified by internal customers.

The above table is a Table View – Simple Requirement Table, a new term table type added by the profile that just shows the requirement name and its specification text.

See section: Requirement Packages.

2.6.4 Table View – Context Diagram Flows Table

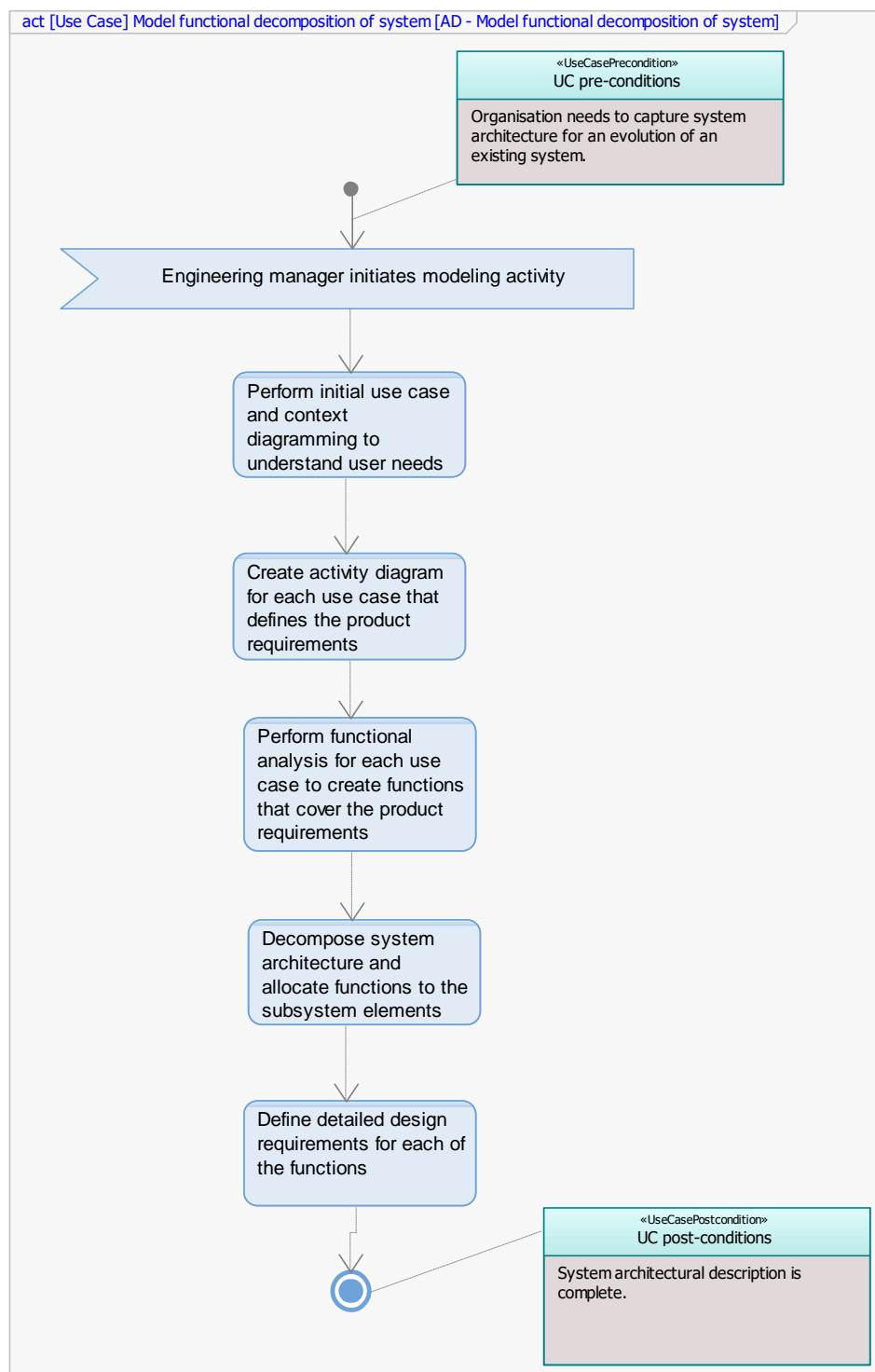


From element in flow	Item Flows	To element in flow
itsExternal Consultant	Give guidance on methods	MBSE System
itsInternal Customer	Provides needs and wants	MBSE System
itsOutside Supplier	Delivers components that work	MBSE System
itsProcess Audit	Performs audits	MBSE System
MBSE System	Provides evidence of compliance	itsProcess Audit
MBSE System	Provides feedback and raises needs	itsExternal Consultant
MBSE System	Delivers requirements that can be tested	itsInternal Customer
MBSE System	Gives detailed design requirements	itsOutside Supplier

The table above shows a new term table type which lists the events on flows in the context diagram and want the from and to element of the flow is.

See section: Context Packages, Signals Package.

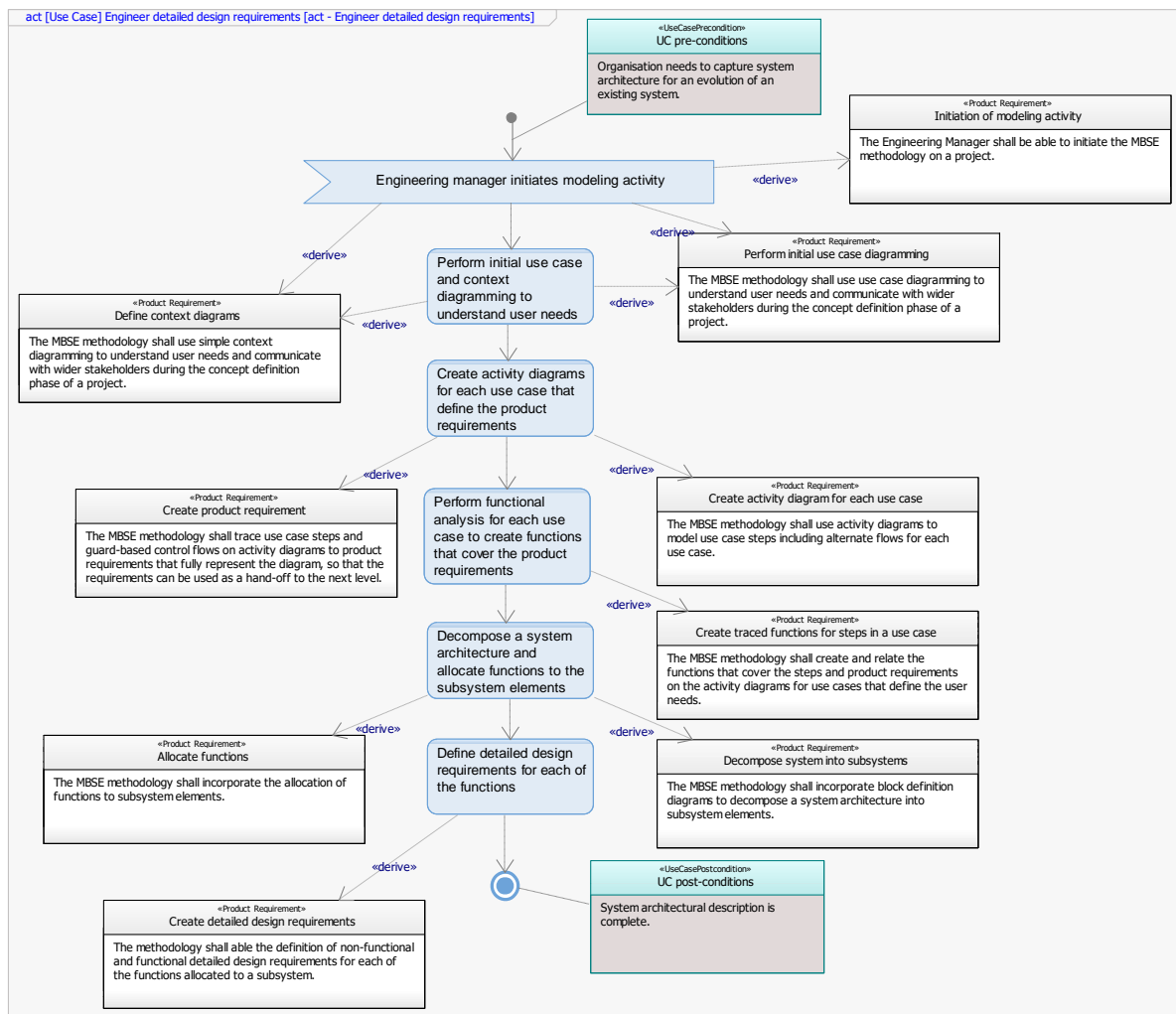
2.6.5 act - Textual Activity Diagram



The above diagram was created from a textual activity diagram with a pre and post condition boxes pre-populated by the helper to improve consistency and speed of modeling.

See section: Textual Activity Diagrams (nested under each use case).

2.6.6 act - Textual Activity Diagram (with Requirements)



The above diagram shows textual requirements created from the actions of the activity diagram. These are product requirements of the system that specify the essential behavioral components necessary in the system for the use case. The process for creating requirements is systematic and the profile provides helpers to speed it up, including automatically moving the requirement into a requirements package and applying stereotypes. A Custom View is provided by the profile to hide the requirements from the diagram if desired.

See section: Textual Activity Diagrams (with Requirements).

2.6.7 req – Product Requirements in a simple requirements table

Found 10 elements	
Name	Specification
A requirement without traceability	This requirement does not have a derivation relation to a function block.
Allocate functions	The MBSE methodology shall incorporate the allocation of functions to subsystem elements.
Create activity diagram for each use case	The MBSE methodology shall use activity diagrams to model use case steps including alternate flows for each use
Create detailed design requirements	The methodology shall able the definition of non-functional and functional detailed design requirements for each of the functions allocated to a subsystem.
Create product requirement	The MBSE methodology shall trace use case steps and guard-based control flows on activity diagrams to product requirements that fully represent the diagram, so that the requirements can be used as a hand-off to the next level.
Create traced functions for steps in a use case	The MBSE methodology shall create and relate the functions that cover the steps and product requirements on the activity diagrams for use cases that define the user
Decompose system into subsystems	The MBSE methodology shall incorporate block definition diagrams to decompose a system architecture into subsystem elements.
Define context diagrams	The MBSE methodology shall use simple context diagramming to understand user needs and communicate with wider stakeholders during the concept definition phase
Initiation of modeling activity	The Engineering Manager shall be able to initiate the MBSE methodology on a project.
Perform initial use case diagramming	The MBSE methodology shall use use case diagramming to understand user needs and communicate with wider stakeholders during the concept definition phase of a

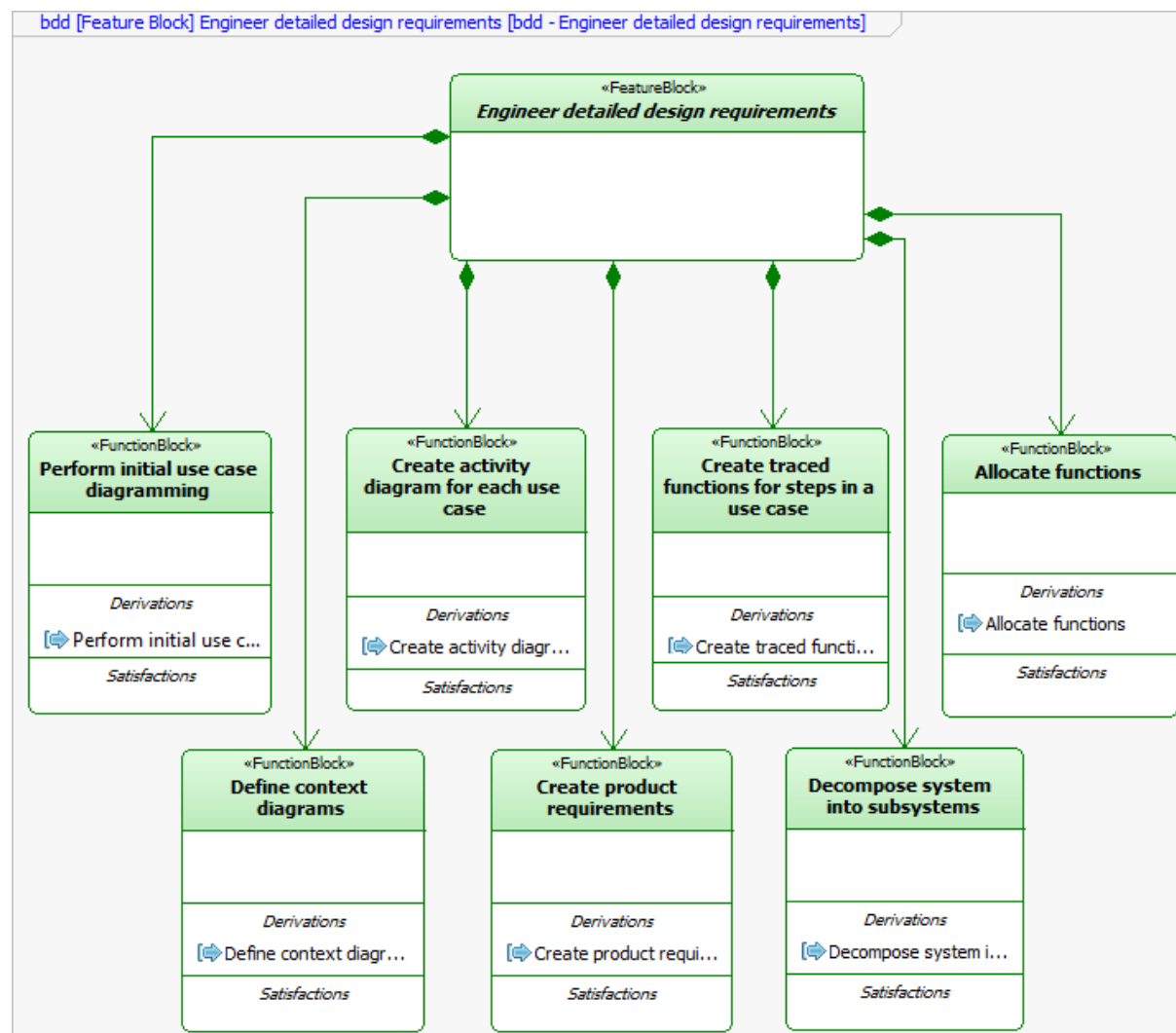
2.6.8 req - Action to Requirement traceability for use case

Found 11 elements			
Activity	Action	Requirement	Specification
<div>Working - act - Engineer detailed design requirements</div>	Create activity diagrams for each use case that define the product requirements	Create activity diagram for each use case	The system shall create activity diagrams for each use case that define the steps performed including alternate flows.
	Create activity diagrams for each use case that define the product requirements	Create product requirement	The system shall create product requirements that trace to actions and guard-based control flows on the activity diagram that fully represent the diagram.
	Decompose a system architecture and allocate functions to the subsystem elements	Allocate functions	The system shall allocate functions to the subsystem elements
	Decompose a system architecture and allocate functions to the subsystem elements	Decompose system into subsystems	The system shall decompose a system architecture to the subsystem elements
	Define detailed design requirements for each of the functions	Create detailed design requirements	The system shall define detailed design requirements for each of the functions
	Engineering manager initiates modeling activity	Define context diagrams	When engineering manager initiates modeling activity the feature shall perform context diagramming to understand user needs
	Engineering manager initiates modeling activity	Perform initial use case diagramming	When engineering manager initiates modeling activity the feature shall perform initial use case diagramming to understand user needs
	Perform functional analysis for each use case to create functions that cover the product requirements	Create traced functions for steps in a use case	The system shall perform functional analysis for each use case to create functions that cover the product requirements
	Perform initial use case and context diagramming to understand user needs	Define context diagrams	When engineering manager initiates modeling activity the feature shall perform context diagramming to understand user needs
	Perform initial use case and context diagramming to understand user needs	Perform initial use case diagramming	When engineering manager initiates modeling activity the feature shall perform initial use case diagramming to understand user needs
	activityfinal_2		

The above relation table uses a context pattern to traverse relationships in the model and build the table. It enables the identification of actions that do not trace.

To make the table clearer, the profile provides a right-click menu to rename actions to match their text. This rename capability is also useful for making traceability to activity diagram elements clearer in external requirements management tools.

2.6.9 bdd – Engineer detailed design requirements feature block




















In the above diagram compartments are used to show the derivation and satisfaction traceability to requirements. Derivation is used for upstream traceability, whereas satisfaction is used for downstream coverage.

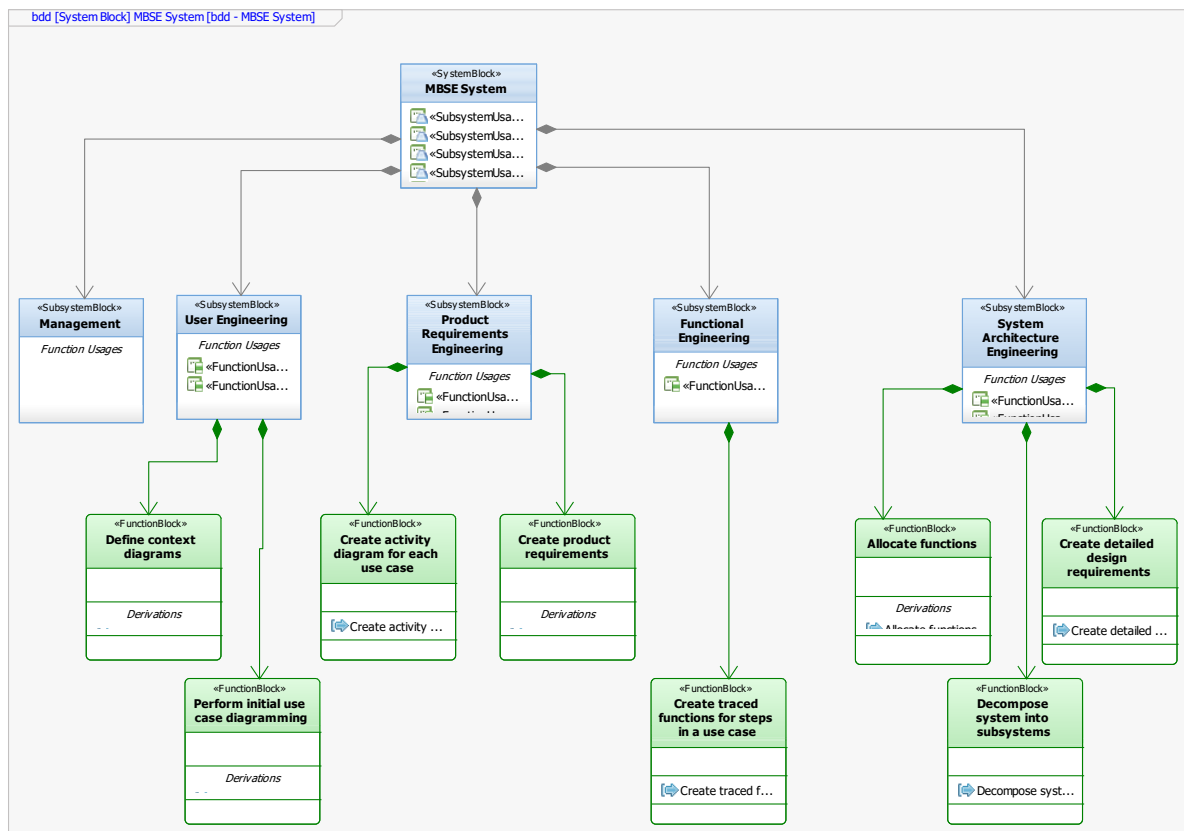
A key part of logical design is the allocation of functions to the logical architecture. The initial way to do this is using an allocate relation from the function to the subsystem that will implement it. This would be claimed by the function modeler.

The system architect would then take these allocate relations and add directed composition relations.

2.6.10 req – Requirement to Function Block relation table for a Feature

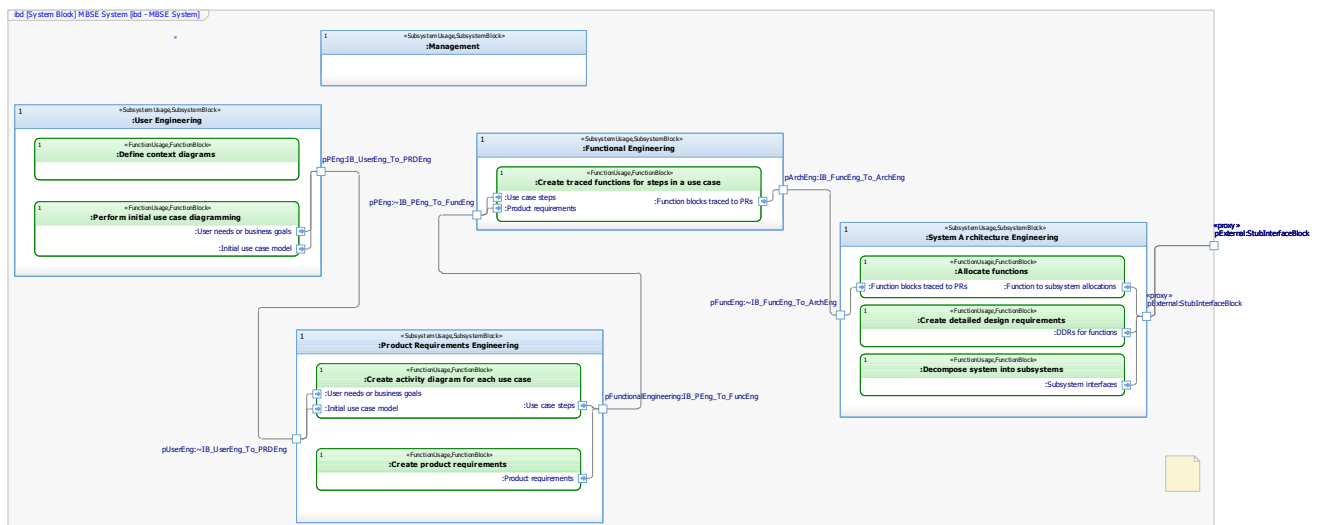
Found 9 elements				
Name (of Requirement)	Specification (of Requirement)	Function Block (with Derivation)	Description ...	Stereotypes (...)
 A requirement without traceability	This requirement does not have a derivation relation to a function block.			
 Allocate functions	The system shall allocate functions to the subsystem elements	 Allocate functions		
 Create activity diagram for each use case	The system shall create activity diagrams for each use case that define the steps performed including alternate flows.	 Create activity diagram for each use case		
 Create detailed design requirements	The system shall define detailed design requirements for each of the functions	 Create detailed design requirements		
 Create product requirement	The system shall create product requirements that trace to actions and guard-based control flows on the activity diagram that fully represent the diagram.	 Create product requirements		
 Create traced functions for steps in a use case	The system shall perform functional analysis for each use case to create functions that cover the product requirements	 Create traced functions for steps in a use case		
 Decompose system into subsystems	The system shall decompose a system architecture to the subsystem elements	 Decompose system into subsystems		
 Define context diagrams	When engineering manager initiates modeling activity the feature shall perform context diagramming to understand user needs	 Define context diagrams		
 Perform initial use case diagramming	When engineering manager initiates modeling activity the feature shall perform initial use case diagramming to understand user needs	 Perform initial use case diagramming		

2.6.11 bdd – Logical Architecture Diagram (including subsystem and function blocks)



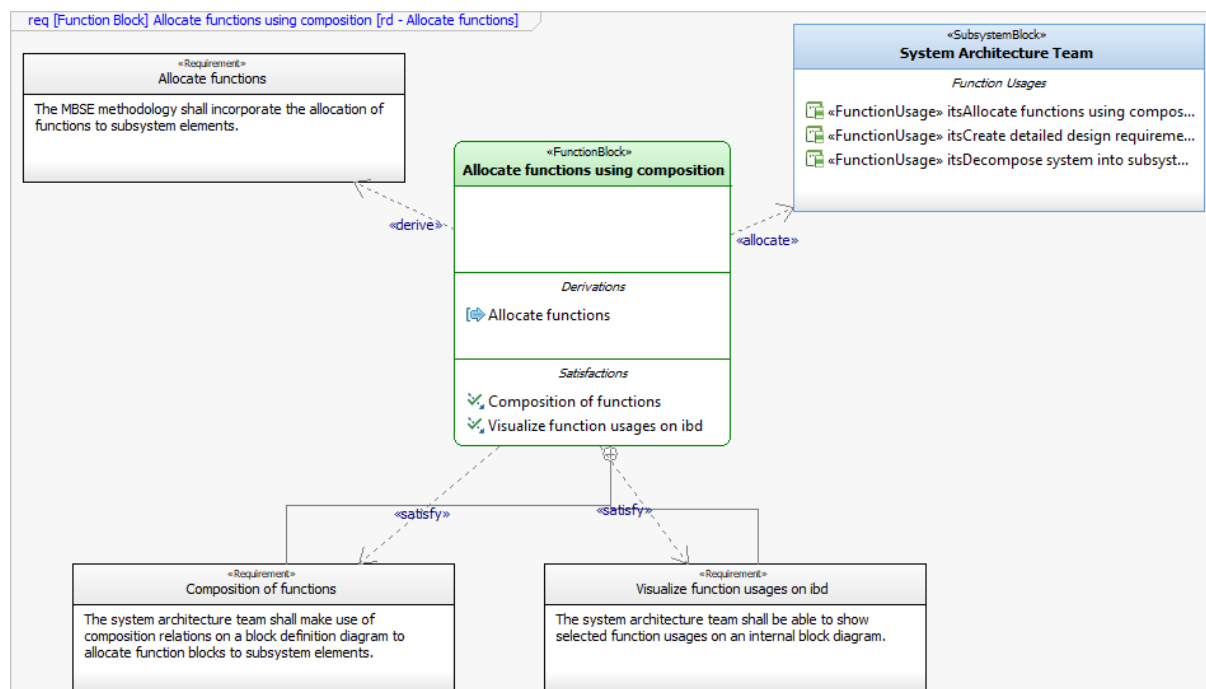
The above diagram shows the decomposition of a system block into subsystem blocks along with the allocation of function blocks to those subsystems using composition relations.

2.6.12 Ibd – Logical Architecture and Interfaces



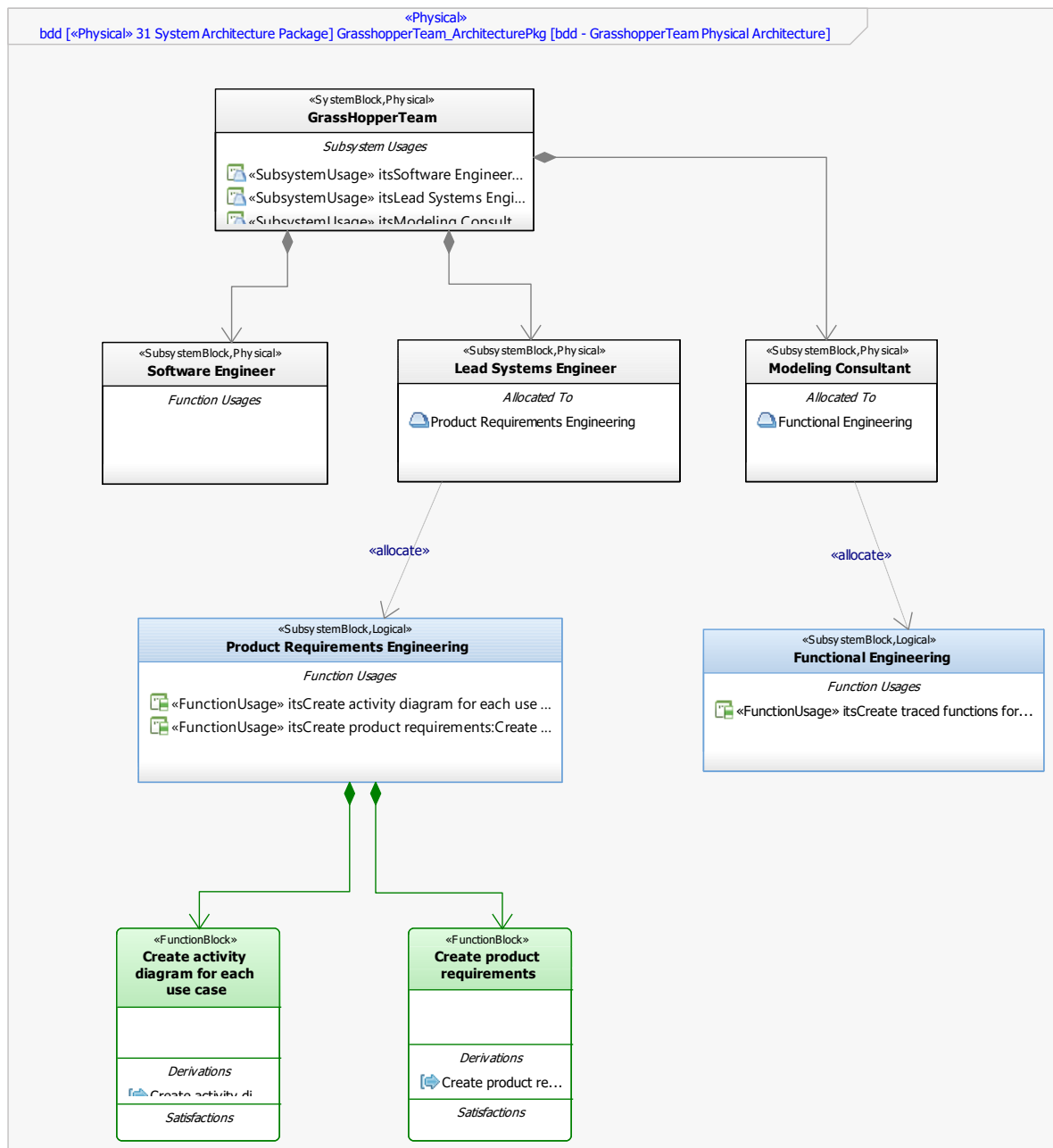
The above diagram shows the corresponding internal block diagram structure for the compositions shown on the IBD. The key thing about an IBD, rather than a BDD, is that it shows the connections and interfaces between the subsystems, i.e., what flows or is required to realize the end-to-end behavior originally defined on the use case's textual activity diagram.

2.6.13 req – Requirements Diagram – System example



The above diagram is a requirements diagram – system owned by the function “Allocate functions ...”. It has a derivation relation to a product requirement in the product requirements package. The two requirements, “Composition of function” and “Visualize function usages” are underneath the function block in the browser hence the corresponding nested relations (circle with cross hairs) are shown. The profile’s listener automatically created the Satisfaction relations from the function to these downstream requirements. This relation will work if the requirements are ultimately switched to DOORS Next, whereas the nesting relation would disappear from the diagram at this point. In the above diagram Derivation is used to trace to upstream product requirements, and the Satisfaction relation has been used to trace to the downstream detailed design requirements. Consistent use of the relations in this way will enable relation tables and custom browsers, by harnessing context patterns that traverse the relations.

2.6.14 bdd – Physical System Architecture (with allocation)



The above diagram shows the decomposition of a physical system architecture into subsystem blocks and the use of allocation to allocate the physical subsystem block to the logical subsystem block (structure to structure allocation). This says that the physical subsystem will realize the logical behavior of its more abstract logical counterpart. The allocation is performed to the upstream logical counterpart because different physical architectures may realize the same logical architecture, and the logical architecture can be locked or added by reference since it is not dependent on any of the elements in the physical architecture.

See section: System Architecture Package (for the Physical Model).

2.6.15 tabel - Physical to Logical allocation relation table

Found 4 elements			
Physical System	Physical Subsystem	Allocated Logical Subsystem Functions	Allocated Logical Subsystem
GrassHopperTeam	Lead Systems Engineer	Create activity diagram for each use case	Product Requirements Engineering
	Lead Systems Engineer	Create product requirements	Product Requirements Engineering
	Modeling Consultant	Create traced functions for steps in a use case	Functional Engineering
	Software Engineer		

The above relation table is built using a context pattern that follows relationships the allocation relation from Physical to Logical structure in the model. If the physical subsystem is allocated to perform the logical behavior of a subsystem then the table will list the functions that are implicit from this. The assumption is that logical structure can be allocated to physical without split, i.e. that the logical is granular enough to be directly allocated.

2.7 Key themes

2.7.1 Using operations vs function blocks to model functions

There are two different methods supported by the Executable MBSE profile for modeling 'functions' either through operations owned by Blocks with the intent to integrate these into an interaction model or use of Function/Feature Block modeling to support function breakdown structure creation using Block Definition Diagrams (BDDs) and Internal Block Diagrams (IBDs) only. These two methods are mutually exclusive ways of performing functional analysis. Both can start with the same concept and requirements definition modeling, which can be common to both.

This document focuses on the latter, i.e., use of function blocks rather than operations as this is seen as the easier to adopt method. Using the function block method, the behavioral and structural models are both created using block definition diagram and internal block diagrams, with options to exploit compartments display options for showing relationship information on the diagram and easily interweave the functional and structural elements.

2.7.2 Requirements vs modeling elements as formal hand-off

The method uses textual requirements as the formal hand-off because:

1. They are a common currency for exchange across differently managed groups.
2. They can be moved into an external requirements management tools for exchange with suppliers or test engineering teams
3. They can be ubiquitous and linked to many different elements in a model and unlike elements such as Operations, they can be shown on any diagram. For example, we might express a test case as a sequence diagram, we might express the requirement as a function block, or want to show the requirement as a transition on a state machine, all of which could be traced to the same requirement, which may start off in the model, but be switched into the requirements management tool.

2.7.3 Use case driven vs functional decomposition

The approach taken for this is based on use case analysis, rather than functional decomposition, and will leads to a definition of functions lifted from use case steps, rather than the deep hierarchy of functions created using a classic function breakdown structure approach.

2.7.4 White box vs black box modeling

If we are modeling the black box view of a vehicle, then we would be modeling the user perceived actions and the visible/known responses of the system, e.g., when the key holder presses the lock on a key the vehicle locks all the doors and flashes the lights. If we model the white box then we are looking inside at the components, their behaviors, and interactions, e.g., the radio receiver receives the signal and decodes it, and the door controller coordinates to send a signal to cause locks to actuate. The white box is therefore more detailed and complex in nature, the black box view is more abstract and might be implemented by multiple architectures to get the same user perceived system behavior.

2.7.5 Logical vs physical modeling

A logical model is like an analysis model and can be devised as a more abstract view of the solution and problem space. In a software electromechanical system, we may define subsystems that represent logical needs in terms of controllers, sensors, and actuation and how they interact.

Whether the controllers are allocated to the same hardware board, could be deferred as could what technology is used to implement the sensing. For example, there are multiple ways of sensing vehicles surroundings; radar or camera. One could defer this choice to the physical model and focus the logical model on the needs.

2.8 FAQs

2.8.1 Can my model have multiple System Blocks?

Potentially. The key is to understand the value of this vs the separation of concern that comes from having separate models, one for each system.

The System Block represents the system for which product requirements are being generated. Where there are multiple levels of hierarchy, Subsystem Blocks are used to model these, keeping the system block as the one that the use cases in the model are being written for. If there is a desire to have use cases at distinct levels, then have separate models.

If the system is logical in nature, i.e., the physical elements are independent, but share the same actors, then one might have multiple system blocks, although it may be simpler to have a different model for each system in this case, since the use case models will be for different systems.

2.8.2 Can I show function blocks on a sequence diagram?

No, not really. An alternative to the sequence diagram which flows top to bottom is to use a timing diagram. With a timing diagram we can use them to express the function blocks, which provides a way to add events that cause the functions to be chained.

2.8.3 What about Requirements Management tools?

Requirements may start initially in the model but then switched so that their master is moved into a requirements management tool, where that tool then takes on the role as master, and the model is showing the links to it.

The binding between the requirements phase and functional phase is textual requirements, supported by activity diagrams. This requirements focus is driven from the desire for the methodology to align with the deployment of modeling in conjunction with a formal requirements management tool, either DOORS classic or DOORS Next.

As such, the Executable MBSE places a focus on textual requirements such as the stitching between different models at different layers. This aligns with the analogy of a sandwich approach to modeling. The idea that a model provides the filling between layers of bread in a stacked sandwich. E.g., system requirements are the input to design synthesis and subsystem and interface requirements are the output. There is a modeling methodology to perform the transformation from one level to another.

2.8.4 What benefit is use case modeling?

Use case modeling is the starting point that allows us to go from one black box to white box, a technique that originated in the 1990's with object-orientated analysis and design, a way of creating software designs by analyzing the problem space, and then designing a solution from the analysis.

A key difference between functional decomposition and use case modeling is that the latter is likely to lead to a flatter functional structure, i.e., you will end up with a layer of functions that in theory is complete and ready to be allocated without further decomposition. If you use functional decomposition, then it is likely that you will have multiple levels of functions before you reach a level that makes sense to allocate to subsystems.

2.8.5 Does the same person do all the phases of the model?

Not usually as one person would not normally own the user needs, product requirements and detailed design requirements in a project. If they did, then it's probable the project is quite small in scope or lacks the needs for a formal methodology.

The purpose of the methodological is to orchestrate multiple people working in parallel on the same project. That said, one might envisage for example:

1. The model is owned by one person, they borrow helpers and have the flexibility to use them as they wish to construct the views they want. Model falls away once they have stopped work on the project (1-2 people working on the same system).
2. The model is just used for training and business evaluation purposes.
3. The model is owned by the business and is the primary mechanism for accumulating the project system engineering understanding. It is maintained by multiple people who work as "cogs" in a machine, handing off artefacts from one layer to another, accumulating knowledge, but rarely working in all the levels (e.g., 50 people working on the same system). This instance different people with separate roles work in part of the model.

2.8.6 Do you have to maintain the activity diagrams?

Understanding the temporal aspects is important. For example, over time knowledge is acquired and incorporated into the model. This means that adding new features requires more work on identifying the differences and less work on redocumenting that the current situation is. You might think of requirements definition as a one-off. Maintaining activity diagrams as you progress through the other diagrams is not necessary as you will conduct a hand-off of requirements.

2.9 New Term types

New term stereotypes are a way to give elements in Rhapsody a more specific meaning than that specified in the base UML or SysML language. In SysML for example, the Block new term is a general-purpose element for specifying a structural element. By adding a concept of a system block the profile specializes SysML to imply a block which has a specific purpose of defining the properties of a system, rather than a subsystem.

Different New Term diagrams are provided to make it easier to draw specific diagrams related to the method.

System Context Diagram: Provides a simplified drawing toolbar and helpers for modeling a classic system context diagram, aka a “petal diagram”, where flows are used to show either roles or data/material flows captured as events.

Block Definition Diagram – System: A specialization of a classic SysML BDD but with the New Term types in the menus and the focus limited to relationships needed. This can be used to perform functional decomposition (via function blocks) or structural decomposition (via system/subsystem blocks) with directed composition relations.

Internal Block Diagram – System: A specialization of a classic SysML IBD but with the New Term types in the menus and the focus limited to relationships needed. This can be used to represent a combined view of the structural and behavioral decomposition created using the BDD – System. This specialization of a classic SysML IBD can also be used to show the flow of Function Usages with decision, parallel, and data flow logic. In classic systems engineering sense this is a functional flow diagram and is remarkably like an activity diagram. We have explicitly chosen to use the class model in Rhapsody for this, however, not the activity model, to provide a greater cooperation between how the structural and behavioral decompositions are performed. IBDs should always be created as a child of the element that owns the usages, not at package level.

Enhanced Use Case Diagram: Same as a normal use case diagram but with the refinement relation tool in the toolbar.

Textual Activity Diagram: Same as a normal activity diagram but set up with a template with pre and post conditions, display options set to allow free flowing text in things like Event Receptions, and more complex features such as swim-lanes and object flow removed, as we have other (better) ways of doing functions that actions on activity diagrams.