SYSTEMVARUHUSET™

SysML – Vad och varför

- Bakgrund
- Varför
- Vad
  - Relation till UML
  - Innehåll
    - Struktur
    - Beteende
    - Krav
  - Cross cutting constructs
    - Allocations
    - Profiles
- Diskussion

Bakgrund

SYSTEMVARUHUSET™

SysML

- Formell standard 2007-09-01
- http://www.omg.org/spec/SysML/1.0/PDF
- Ursprung: ad/2003-03-41 (UML for Systems Engineering RFP)

Relaterade standards:

- OMG XMI 2.1 model interchange standard
- ISO 10303 STEP AP233 data interchange standard for systems engineering tools.

SYSTEMVARUHUSET™

Varför SysML?

- SysML is intended to unify the diverse modeling languages currently used by systems engineers.
- Since SysML uses UML 2 as its foundation, systems engineers modeling with SysML and software engineers modeling with UML 2 will be able to collaborate on models of software-intensive systems. This will improve communication among the various stakeholders who participate in the systems development process and promote interoperability among modeling tools.

Relationen till UML

SYSTEMVARUHUSET™

UML och SysML

Ingår ej i SysML:

- *Klass*
- *Composite structure*
- *Objekt*
- Komponent
- Kommunikation
- Interaktionsöversikt
- Timing
- Deployment

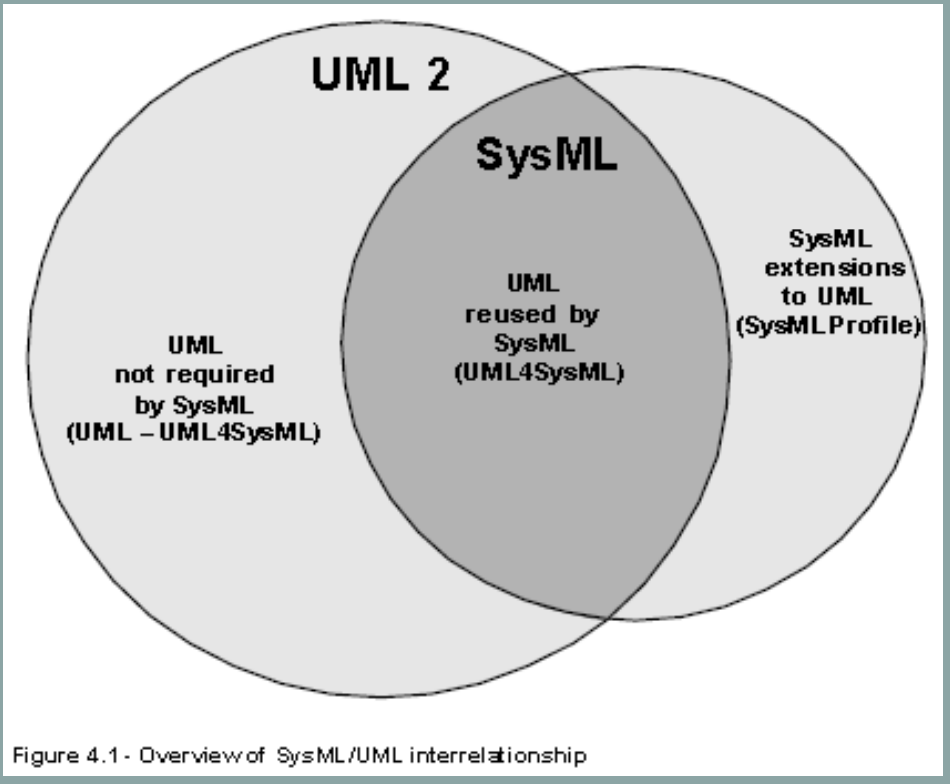


Figure 4.1 - Overview of SysML/UML interrelationship

Relationen till UML

SYSTEMVARUHUSET™

UML's metaklasser som används i SysML

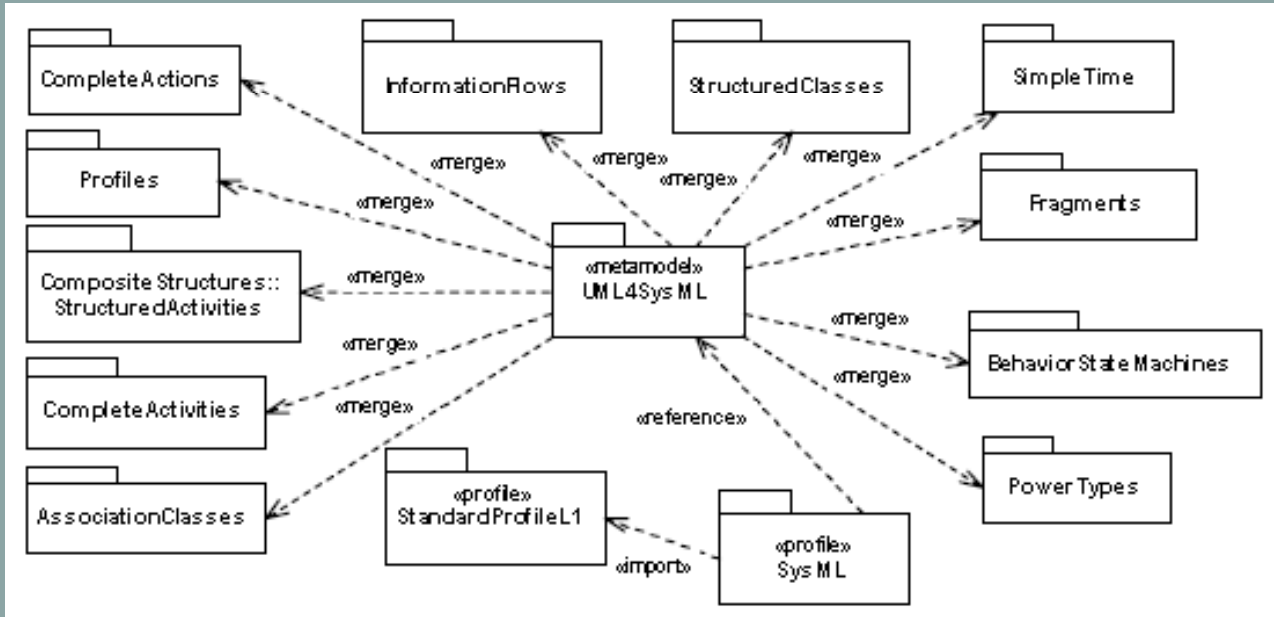


Figure 4.2 - SysML Extension of UML

Relationen till UML

SYSTEMVARUHUSET™

Utökningar i SysML

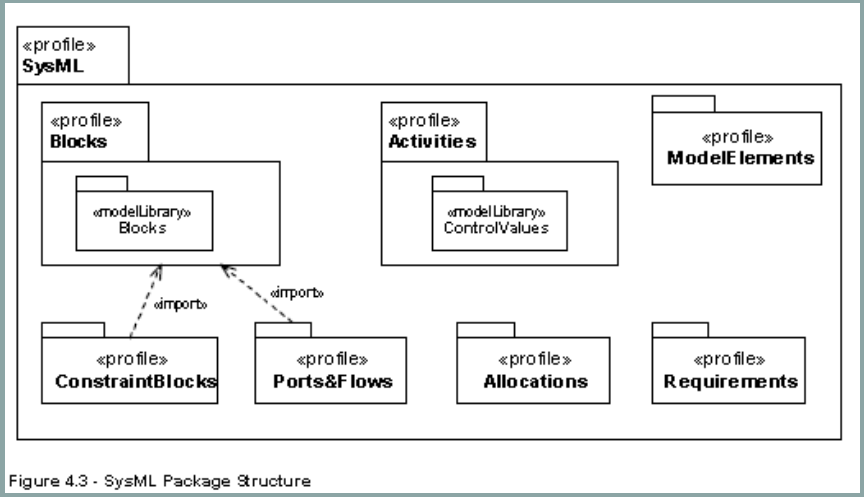


Figure 4.3 - SysML Package Structure

The SysML packages extend UML as follows:

- SysML::Model Elements refactors and extends the UML kernel portion of UML classes
- SysML::Blocks reuses structured classes from composite structures
- SysML::ConstraintBlocks extends Blocks to support parametric modeling
- SysML::Ports and Flows extends UML::Ports, UML::InformationFlows and SysML::Blocks
- SysML::Activities extends UML activities
- SysML::Allocations extends UML dependencies
- SysML::Requirements extends UML classes and dependencies

SysMLs innehåll

SYSTEMVARUHUSET™

Diagram i SysML

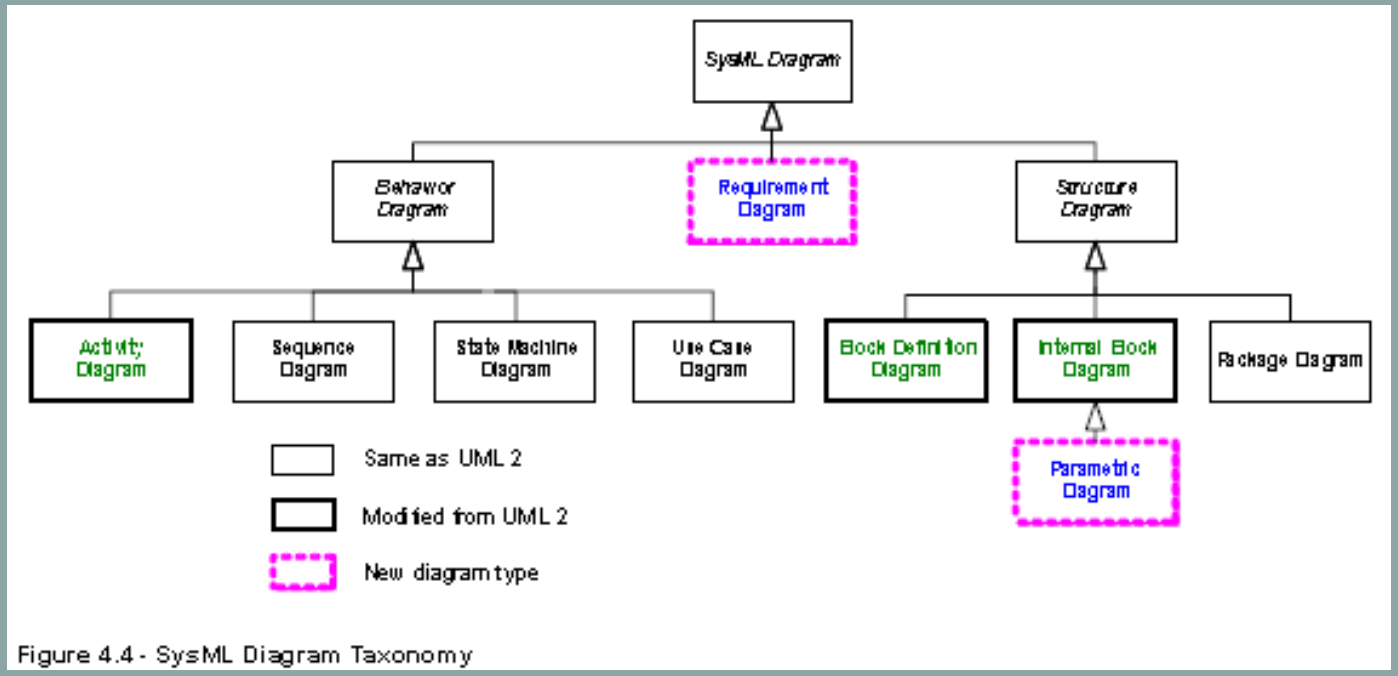
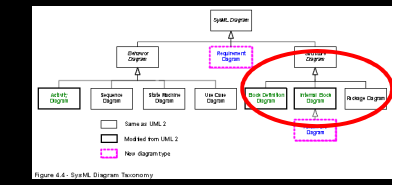


Figure 4.4 - SysML Diagram Taxonomy

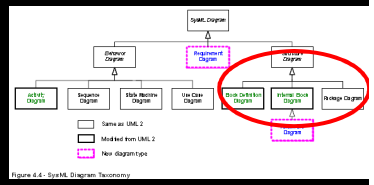
SYSTEMVARUHUSET™



Blocks

- A Block is a modular unit that describes the structure of a system or element.
- It may include both structural and behavioral features, such as properties and operations, that represent the state of the system and behavior that the system may exhibit.
- SysML blocks are based on UML classes, as extended by UML composite structures. SysML value types are based on UML data types

SYSTEMVARUHUSET™



Block diagram - Exempel

- A block definition diagram is based on the UML class diagram
- The variety of notations for associations has been reduced to simplify the burden of teaching, learning, and interpreting SysML diagrams for the systems engineering user.
  - n-ary associations
  - qualified associations

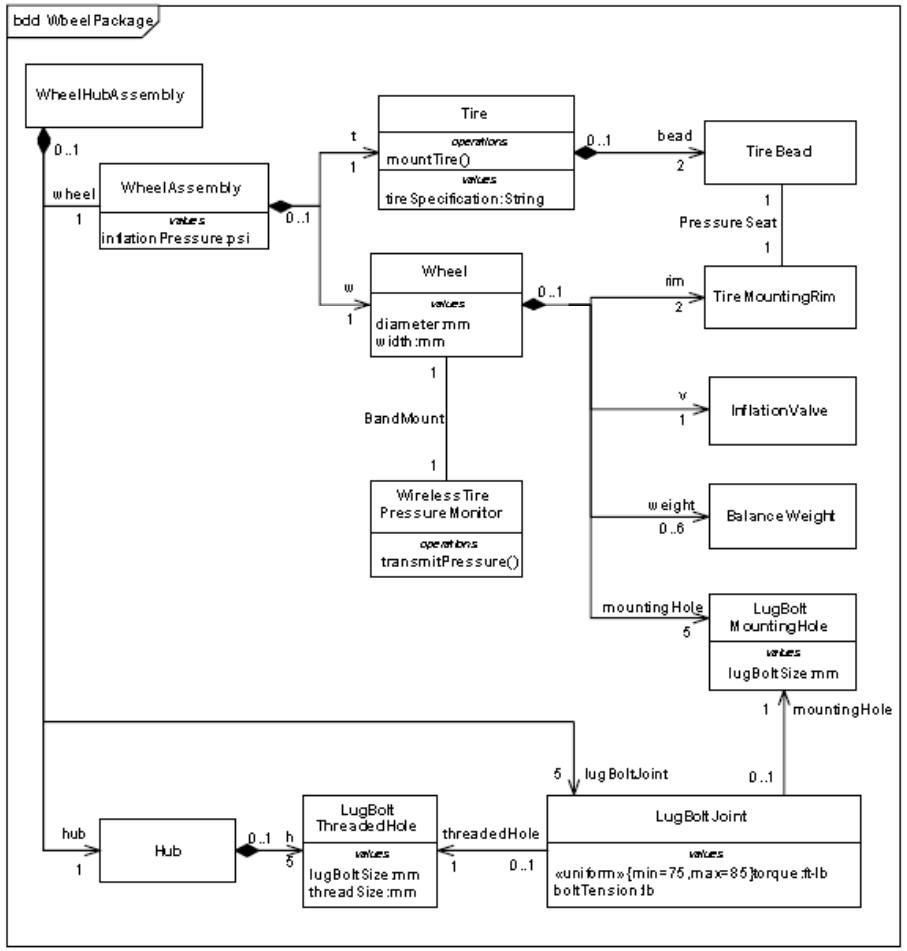
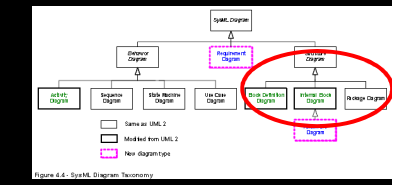


Figure 8.8 - Block diagram for the Wheel Package

SYSTEMVARUHUSET™



Internal block diagram (ibd)

An internal block diagram is based on the UML composite structure diagram

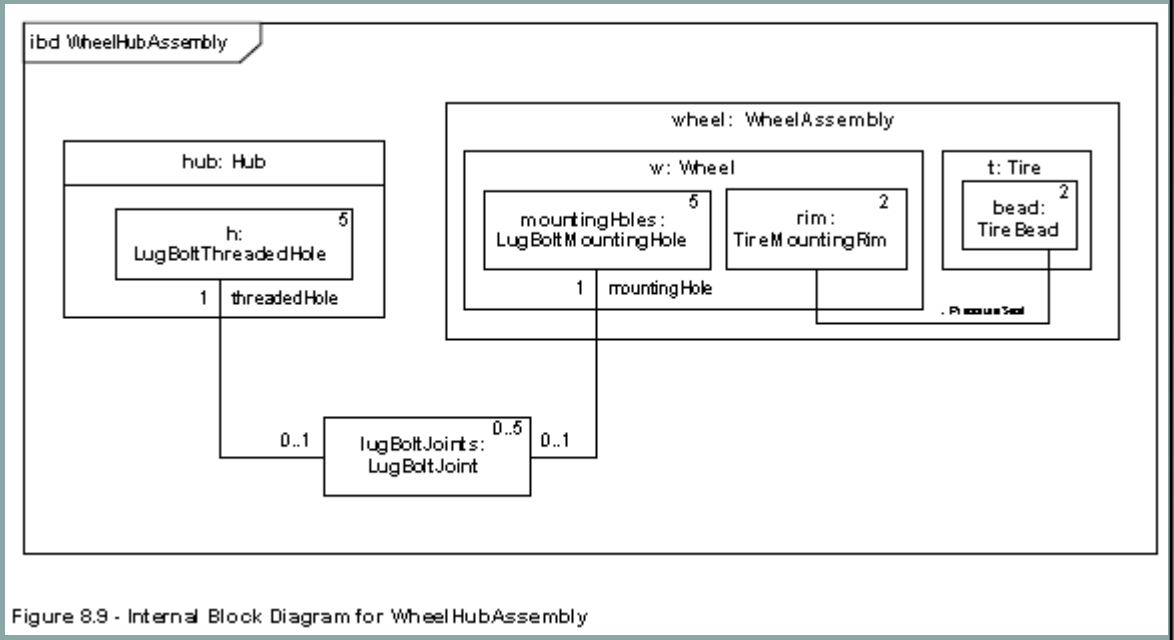
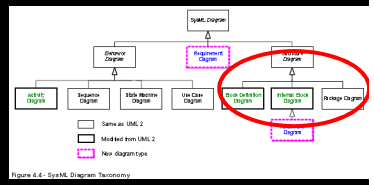


Figure 8.9 - Internal Block Diagram for WheelHubAssembly

SYSTEMVARUHUSET™



Ibd istället för objektdiagram => context blocks

SysML internal block diagrams may be used to specify blocks with unique identification and property values.

Figure 8.11 shows an example used to specify a unique vehicle with a vehicle identification number (VIN) and unique properties such as its weight, color, and horsepower.

This concept is distinct from the UML concept of instance specifications in that it does not imply or assume any run-time semantic, and can also be applied to specify design configurations.

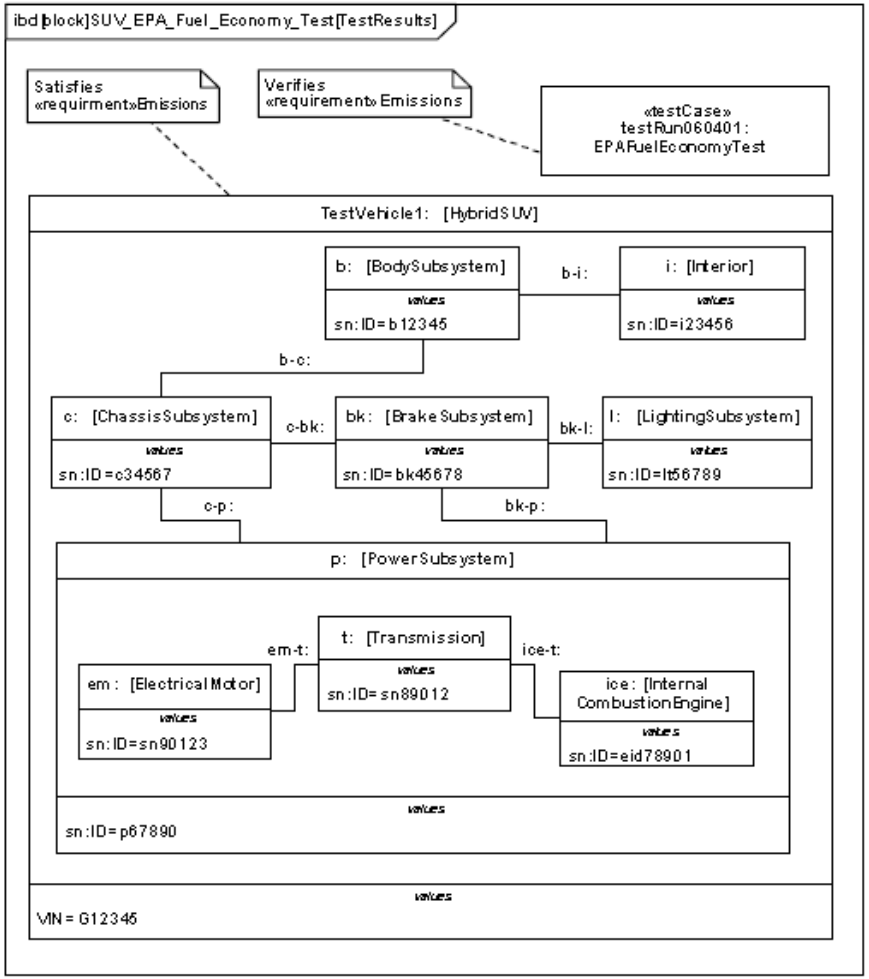
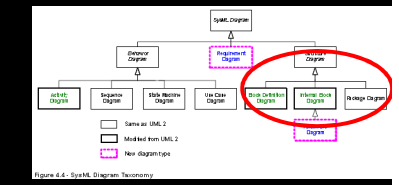


Figure 8.11 - SUV EPA Fuel Economy Test

SYSTEMVARUHUSET™

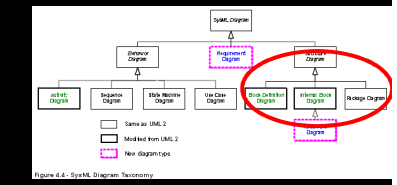


Ports and flows

- Ports and Flows provides the semantics for defining how blocks and parts interact through ports and how items flow across connectors.

- Flow ports enable flow of items between blocks and parts
- Standard ports enable invocation of services on blocks and parts.

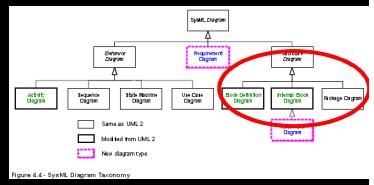
SYSTEMVARUHUSET™



Item flows

- Item flows represent the things that flow between blocks and/or parts and across associations or connectors. Whereas flow ports specify what “can” flow in or out of a block, item flows specify what “does” flow between blocks and/or parts in a particular usage context.
- This important distinction enables blocks to be interconnected in different ways depending on its usage context. For example, a tank may include a flow port that can accept fluid as an input. In a particular use of the tank, “gasoline” flows across a connector into its flow port, and in another use of the tank, “water” flows across a connector into its flow port. The item flow would specify what “does” flow on the connector in the particular usage (e.g., gas, water) and the flow port specifies what can flow (e.g., fluid). This enables type matching between the item flows and between flow ports to assist in interface compatibility analysis.

SYSTEMVARUHUSET™



Exempel – standard ports

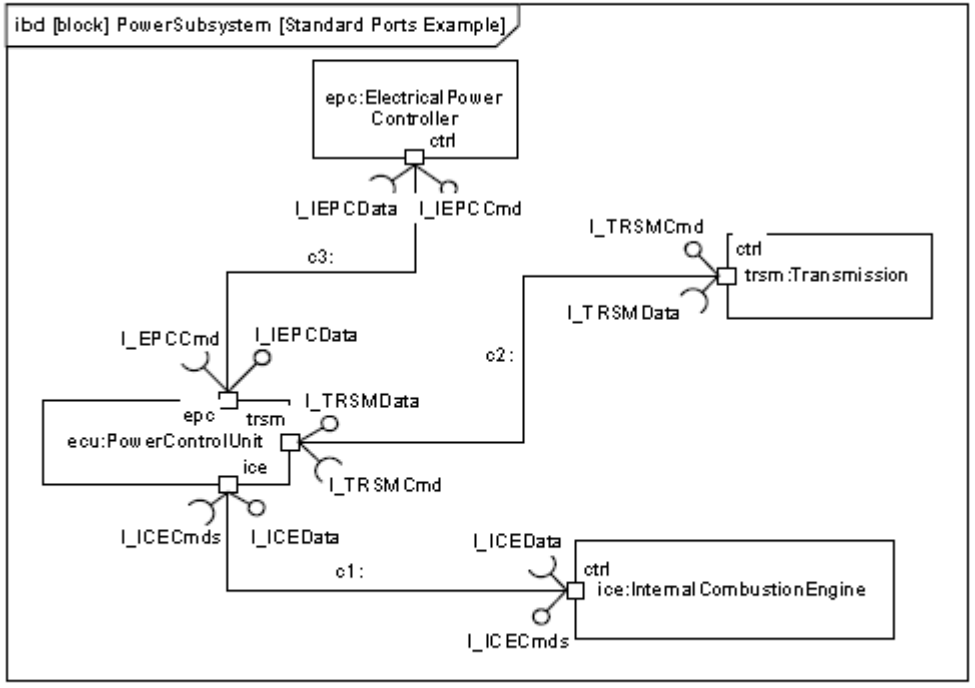
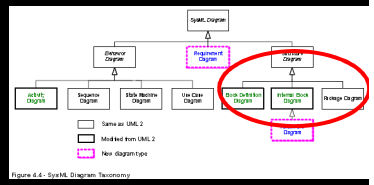
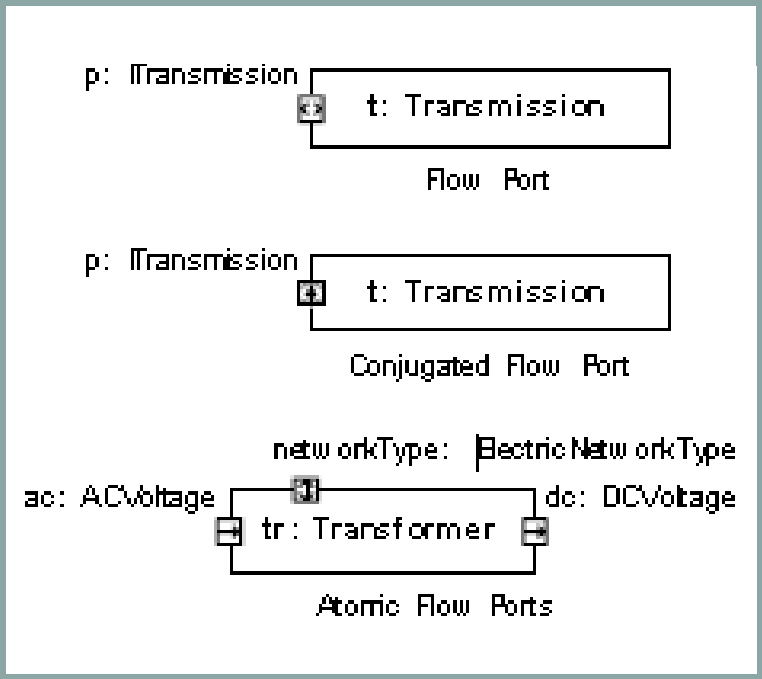


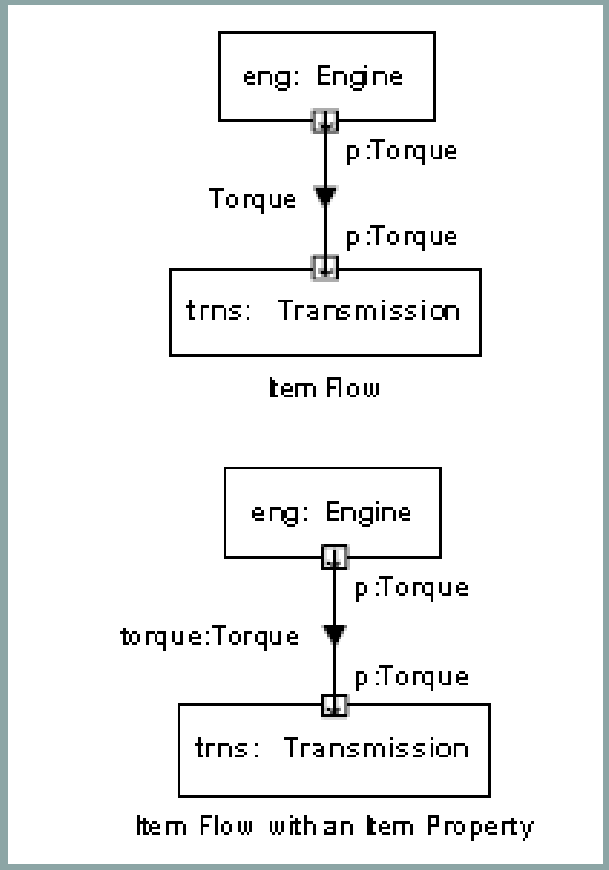
Figure 9.3 - Usage example of standard ports

SYSTEMVARUHUSET™

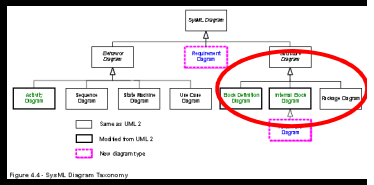


Exempel – flow ports, item flows





SYSTEMVARUHUSET™



Parametric diagram

Parametric diagrams model a network of constraints on system properties to support engineering analysis, such as performance, reliability, and mass properties analysis.

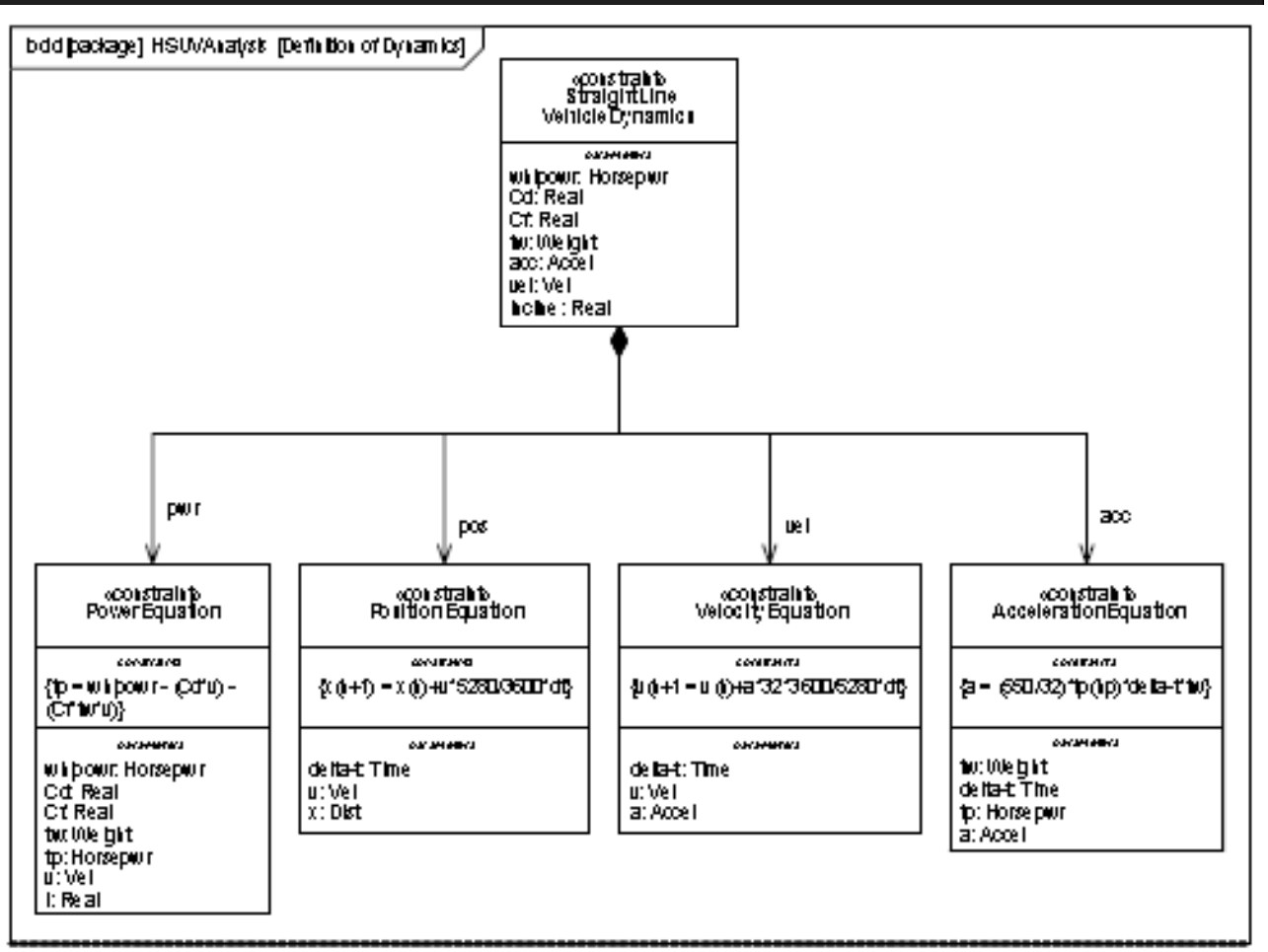
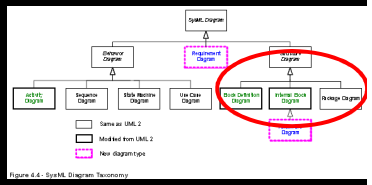


Figure 10.2 Definition of constraint blocks on a block definition diagram



SYSTEMVARUHUSET™

Exempel – parametric diagram

A parametric diagram is defined as a restricted form of internal block diagram.

the only connectors that may be shown are binding connectors connected to constraint parameters on at least one end.

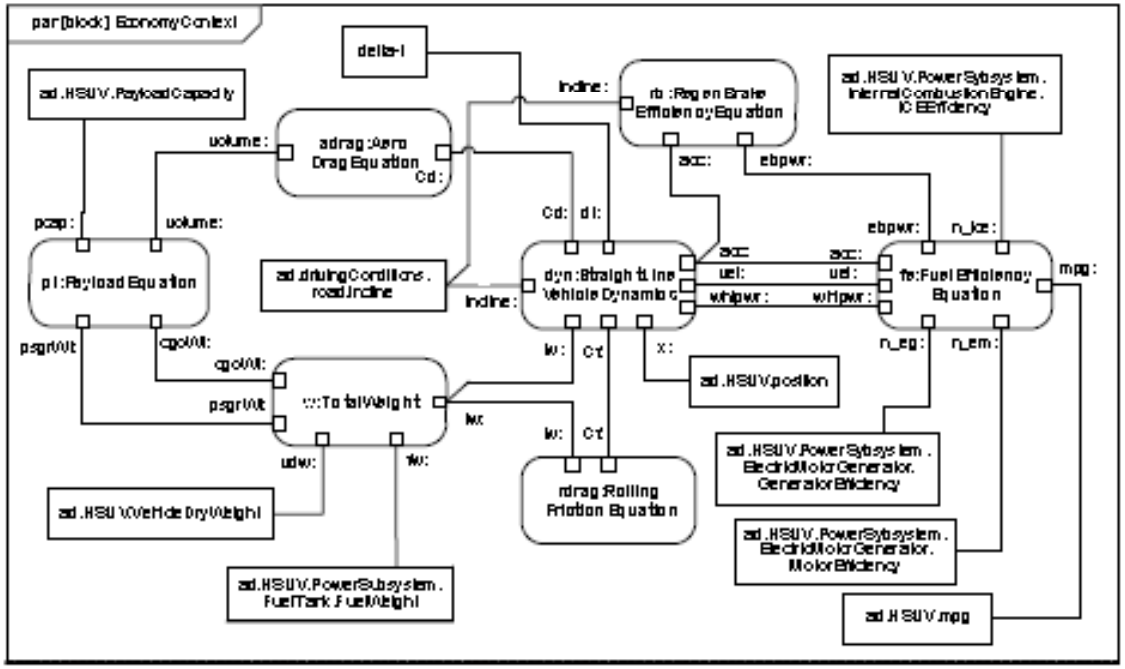
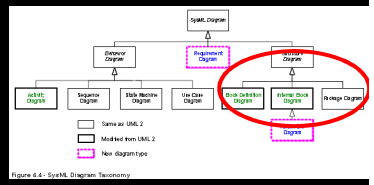


Figure 10.3 -Usage of constraint blocks on a parametric diagram

SYSTEMVARUHUSET™



SysML's stereotypade strukturella modellelement

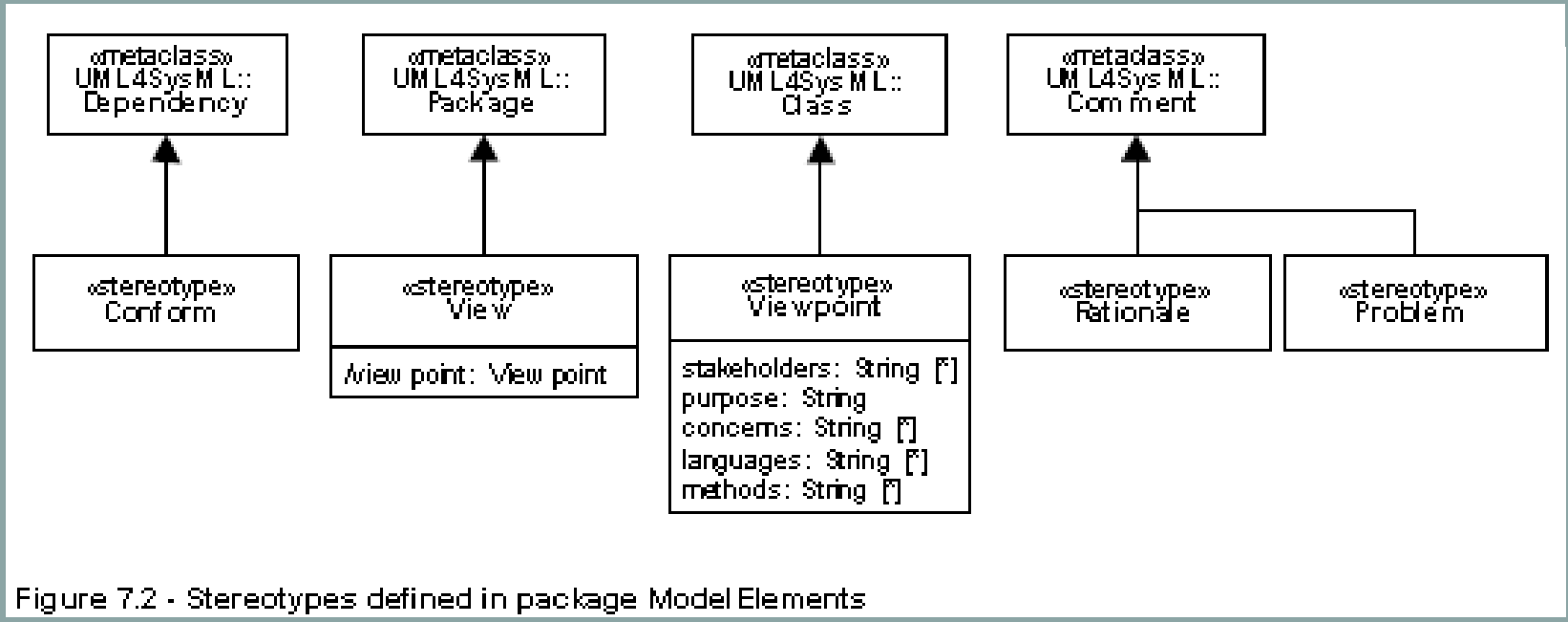
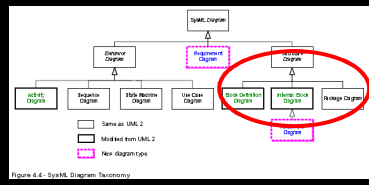


Figure 7.2 - Stereotypes defined in package Model Elements

SYSTEMVARUHUSET™



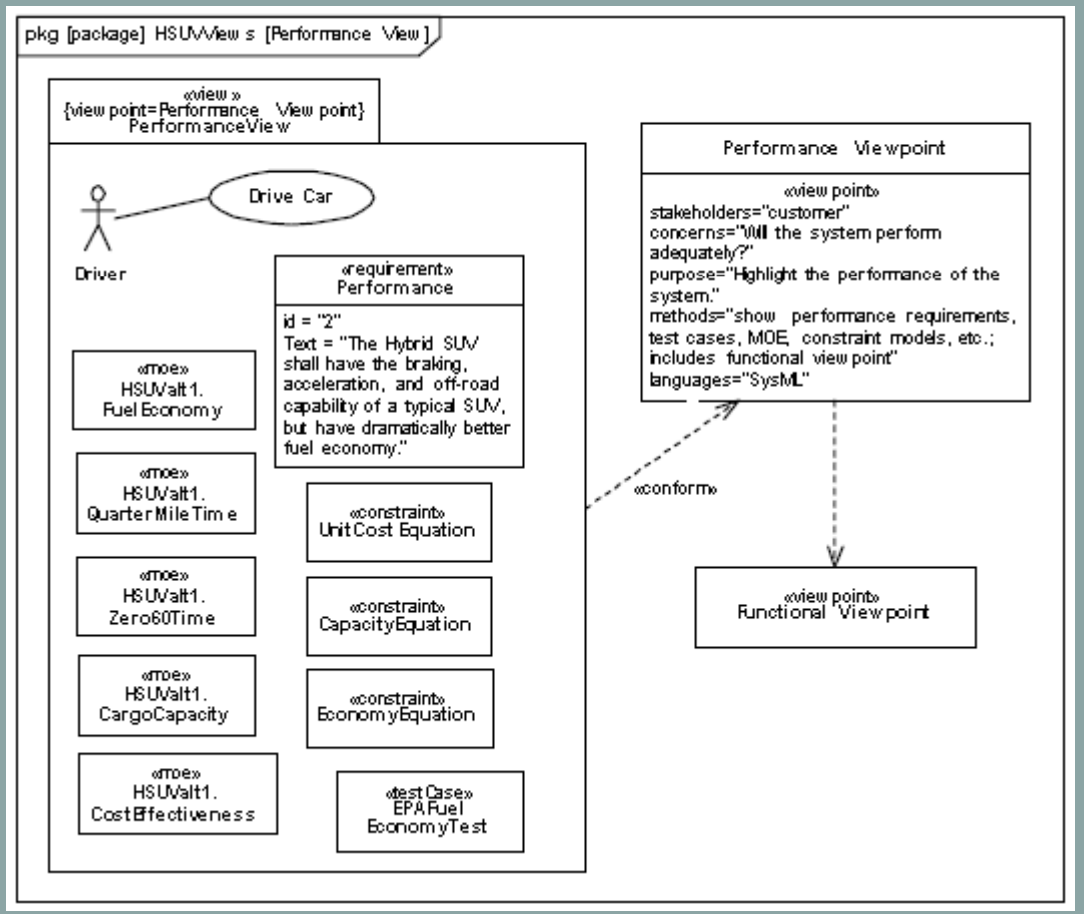
Exempel – View, Viewpoint och conform

View – Representerar systemet

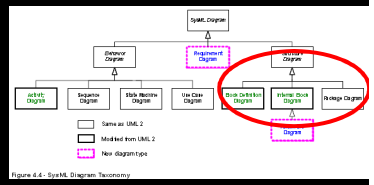
Viewpoint – Regelverk

Conforms – Systemet följer regelverket

Överensstämmer med IEEE1471 - Recommended Practice for Architecture Description of Software-Intensive Systems



SYSTEMVARUHUSET™



Exempel – Rationale, problem

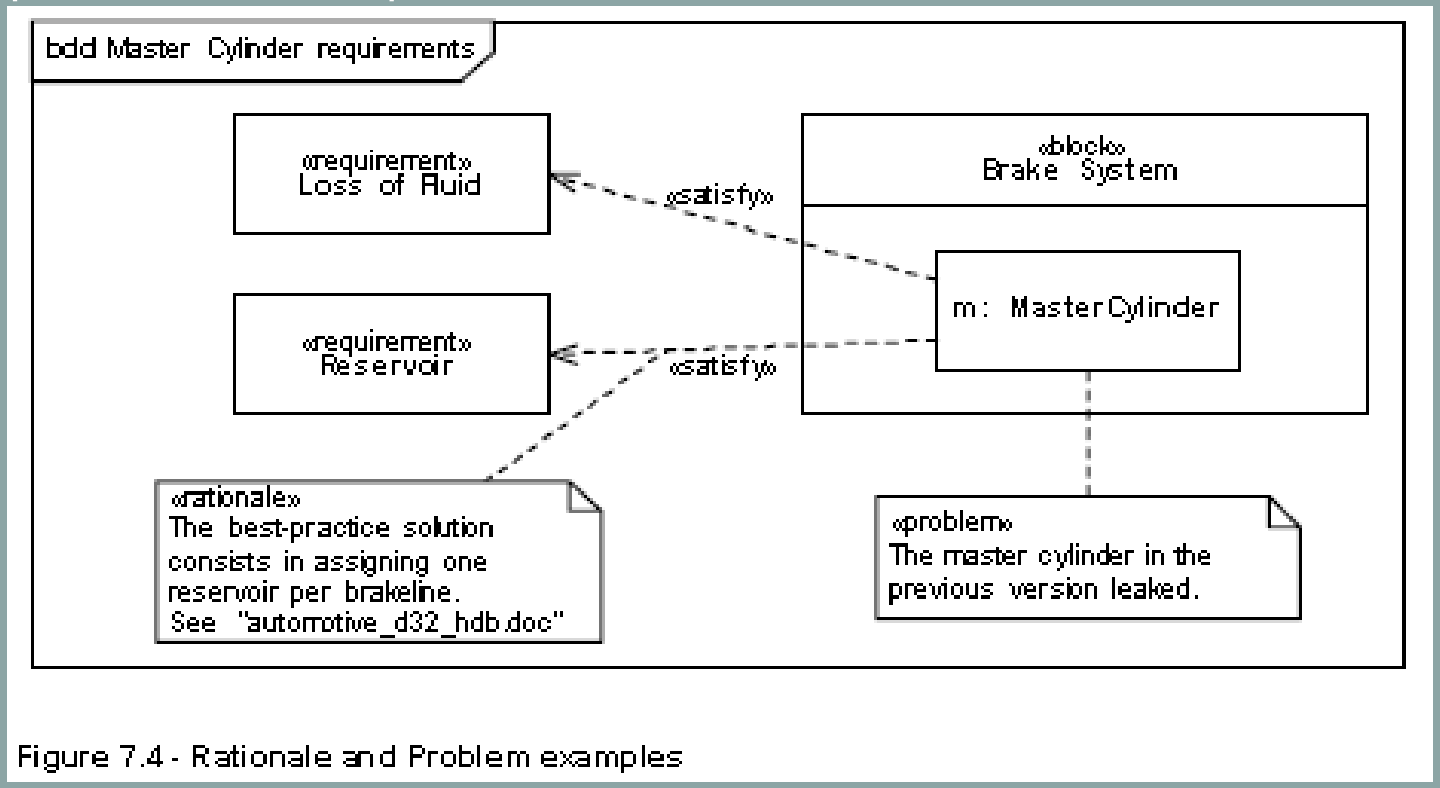
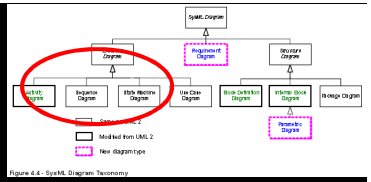


Figure 7.4 - Rationale and Problem examples

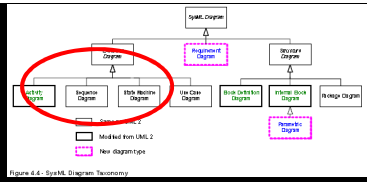
SYSTEMVARUHUSET™



Att beskriva beteende med SysML

- Activities - defines the extensions to UML 2 activities, which represent the basic unit of behavior that is used in activity, sequence, and state machine diagrams. The activity diagram is used to describe the flow of control and flow of inputs and outputs among actions.
- Interactions - defines the constructs for describing message based behavior used in sequence diagrams.
- State Machines - describes the constructs used to specify state based behavior in terms of system states and their transitions.
- Use Cases - describes behavior in terms of the high level functionality and uses of a system, that are further specified in the other behavioral diagrams referred to above.

SYSTEMVARUHUSET™

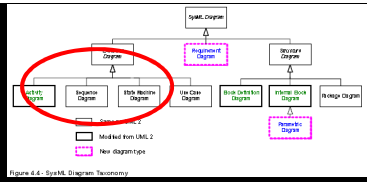


Utökat aktivitetsdiagram

- Activity modeling emphasizes the inputs, outputs, sequences, and conditions for coordinating other behaviors. It provides a flexible link to blocks owning those behaviors.

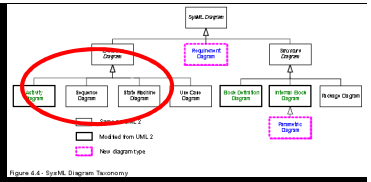
- Activities as classes
- Timelines
- Control as data
- Continuous Systems
- Probability

SYSTEMVARUHUSET™



Activities as classes

- In UML 2.1, all behaviors including activities are classes, and their instances are executions. Behaviors can appear on block definition and class diagrams, and participate in generalization and associations. SysML clarifies the semantics of composition association between activities, and between activities and the type of object nodes in the activities, and defines consistency rules between these diagrams and activity diagrams.



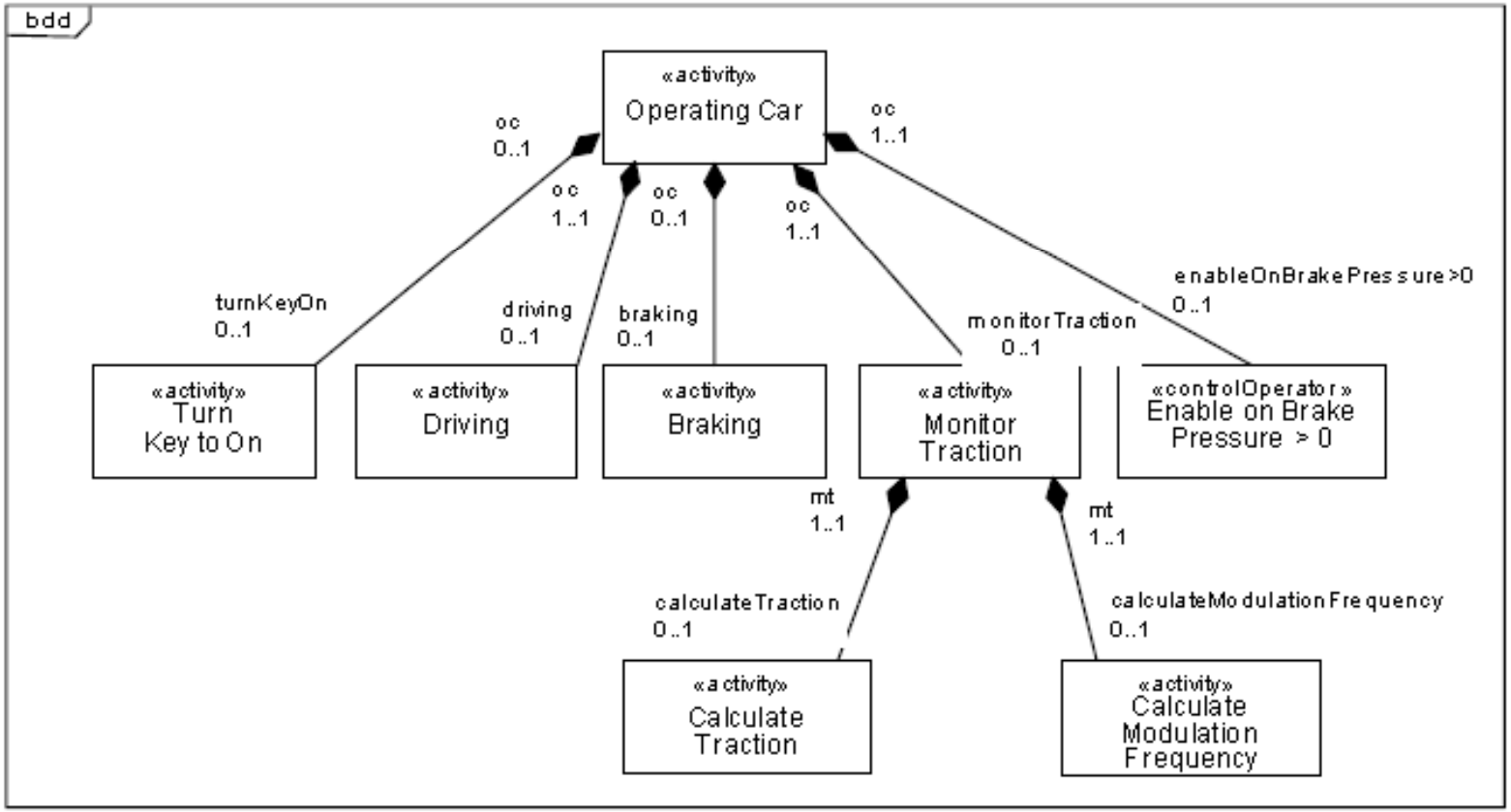
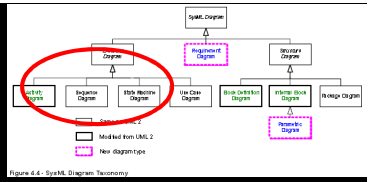


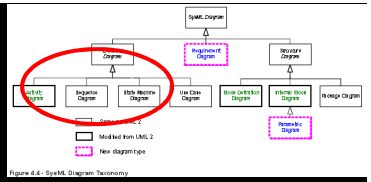
Figure 11.13 - Example block definition diagram for activity decomposition



SYSTEMVARUHUSET™

Timelines

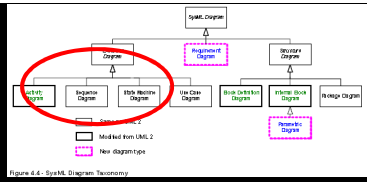
- The simple time model in UML can be used to represent timing and duration constraints on actions in an activity model. These constraints can be notated as constraint notes in an activity diagram. Although the UML 2 timing diagram was not included in this version of SysML, it can complement SysML behavior diagrams to notate this information. More sophisticated SysML modeling techniques can incorporate constraint blocks to specify resource and related constraints on the properties of the inputs, outputs, and other system properties.



SYSTEMVARUHUSET™

Control as data – disabling actions

- SysML extends control in activity diagrams as follows.
- In UML 2.1 Activities, control can only enable actions to start. SysML extends control to support disabling of actions that are already executing. This is accomplished by providing a model library with a type for control values that are treated like data
- A control value is an input or output of a control operator, which is how control acts as data. A control operator can represent a complex logical operation



SYSTEMVARUHUSET™

Stereotyper i aktivitetsdiagrammet

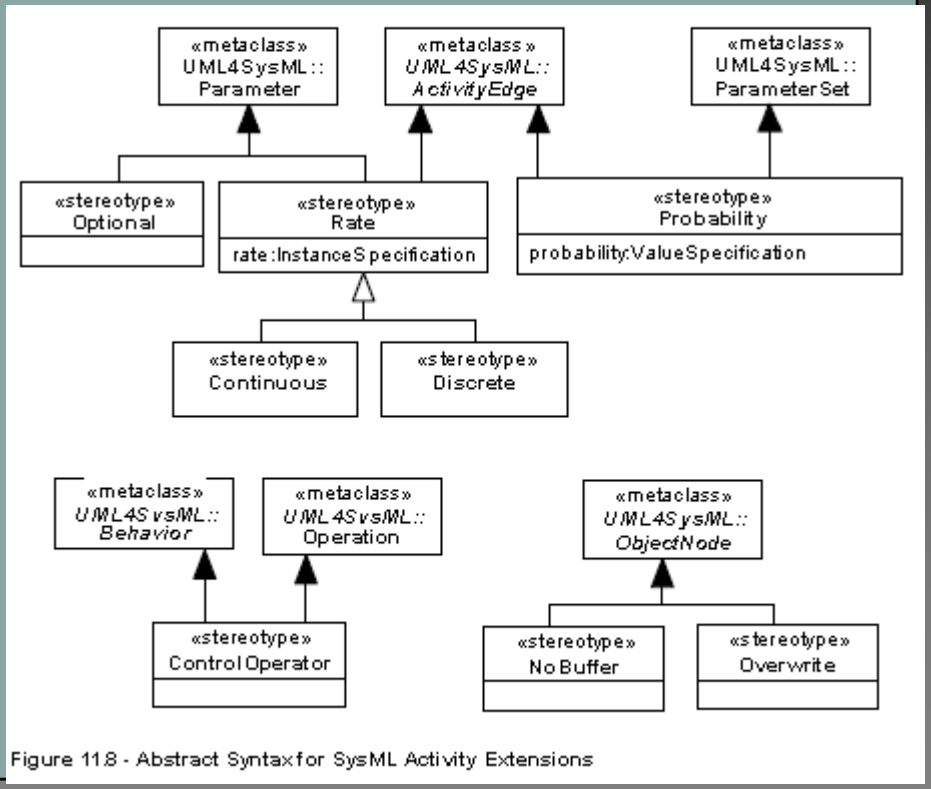
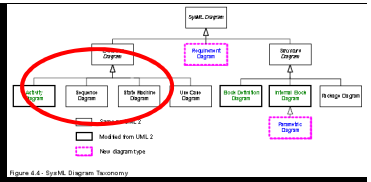


Figure 11.8 - Abstract Syntax for SysML Activity Extensions

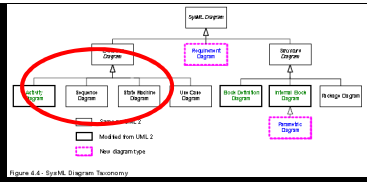


SYSTEMVARUHUSET™

Continuous systems – rate of flow, replacing values & descarding values

SysML provides extensions that might be very loosely grouped under the term "continuous," but are generally applicable to any sort of distributed flow of information and physical items through a system. These are:

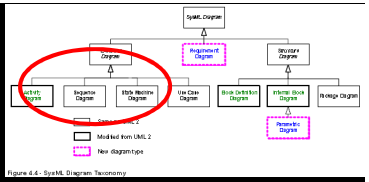
- Restrictions on the rate at which entities flow along edges in an activity, or in and out of parameters of a behavior. This includes both discrete and continuous flows, either of material, energy, or information.
- Extension of object nodes, including pins, with the option for newly arriving values to replace values that are already in the object nodes SysML also extends object nodes with the option to discard values if they do not immediately flow downstream. These two extensions are useful for ensuring that the most recent information is available to actions by indicating when old values should not be kept in object nodes, and for preventing fast or continuously flowing values from collecting in an object node, as well as modeling transient values, such as electrical signals.



Probability

SysML introduces probability into activities as follows:

- Extension of edges with probabilities for the likelihood that a value leaving the decision node or object node will traverse an edge.
- Extension of output parameter sets with probabilities for the likelihood that values will be output on a parameter set.



SYSTEMVARUHUSET™

Continuous

Continuous rate is a special case of rate of flow where the increment of time between items approaches zero. It is intended to represent continuous flows that may correspond to water flowing through a pipe.

It is independent from UML streaming. A streaming parameter may or may not apply to continuous flow, and a continuous flow may or may not apply to streaming parameters.

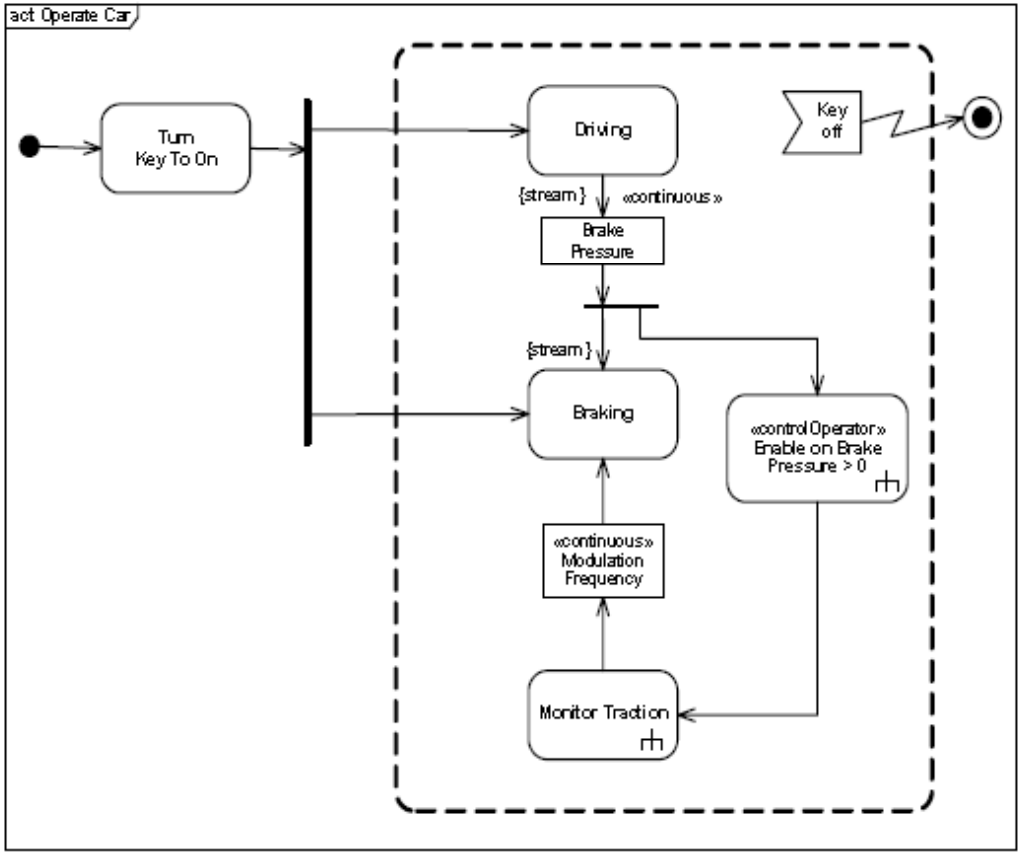
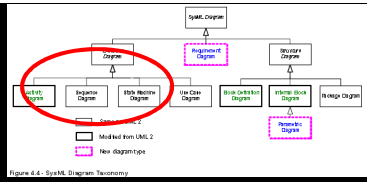


Figure 11.10 - Continuous system example 1



SYSTEMVARUHUSET™

Continuous – Monitor traction

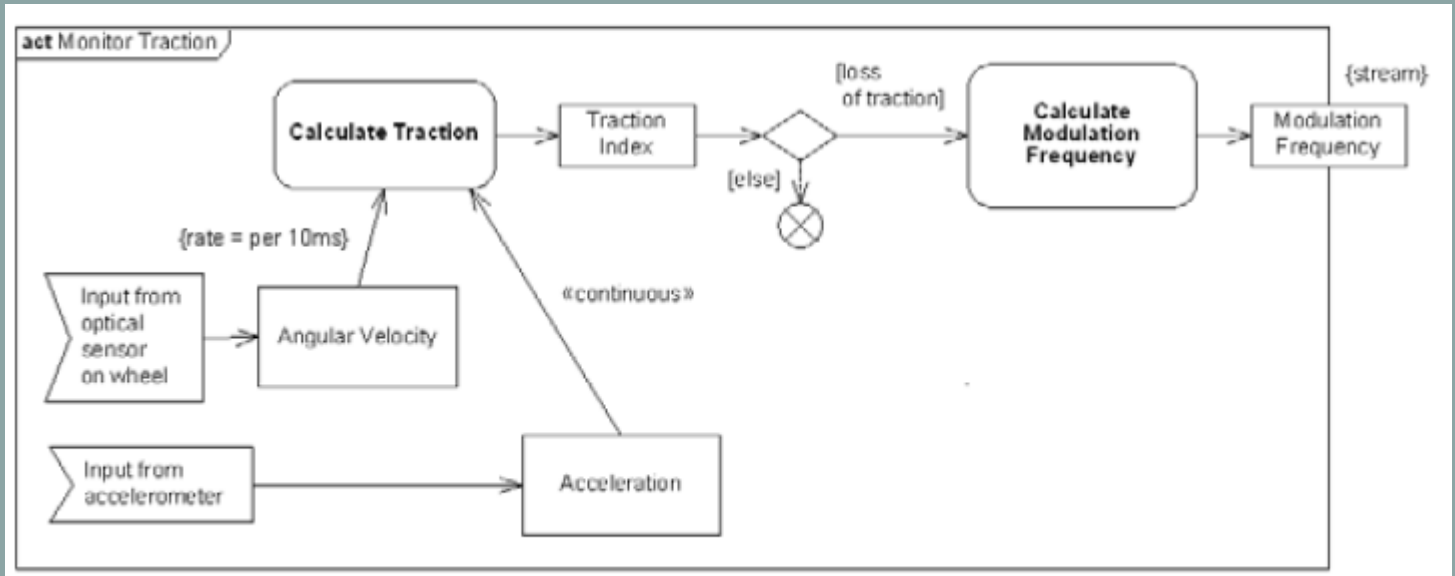
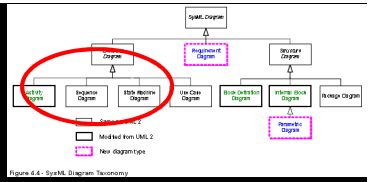


Figure 11.11 - Continuous system example 2



SYSTEMVARUHUSET™

Continuous

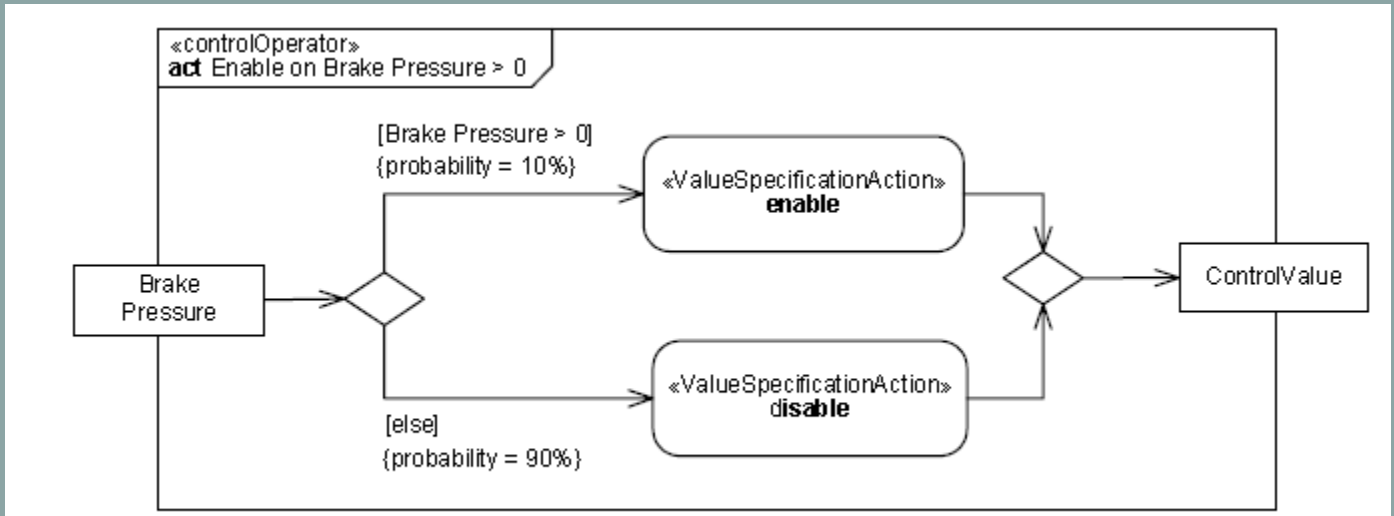
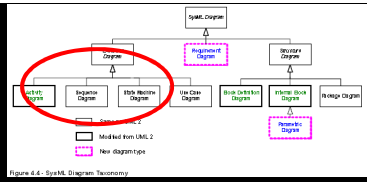


Figure 11.12 - Continuous system example 3

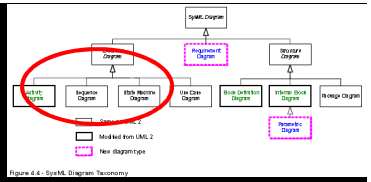
SYSTEMVARUHUSET™



Interactions

SysML includes the Sequence Diagram only and excludes the Interaction Overview Diagram and Communication Diagram, which were considered to offer significantly overlapping functionality without adding significant capability for system modeling applications. The Timing Diagram is also excluded due to concerns about its maturity and suitability for systems engineering needs.

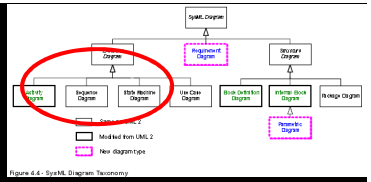
SYSTEMVARUHUSET™



State machines

The UML concept of protocol state machines is excluded from SysML to reduce the complexity of the language. The standard UML state machine concept (called behavior state machines in UML) are thought to be sufficient for expressing protocols.

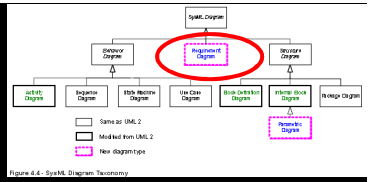
SYSTEMVARUHUSET™



Use cases

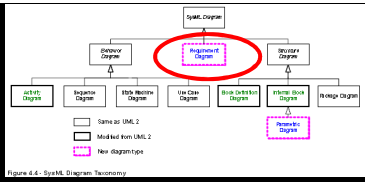
Användningsfallsdiagrammet ingår i SysML

SYSTEMVARUHUSET™



Cross cutting constructs

- Requirements
- Allocations
- Profiles



Requirements

A requirement is a stereotype of Class.

Compound requirements can be created by using the nesting capability of the class definition mechanism.

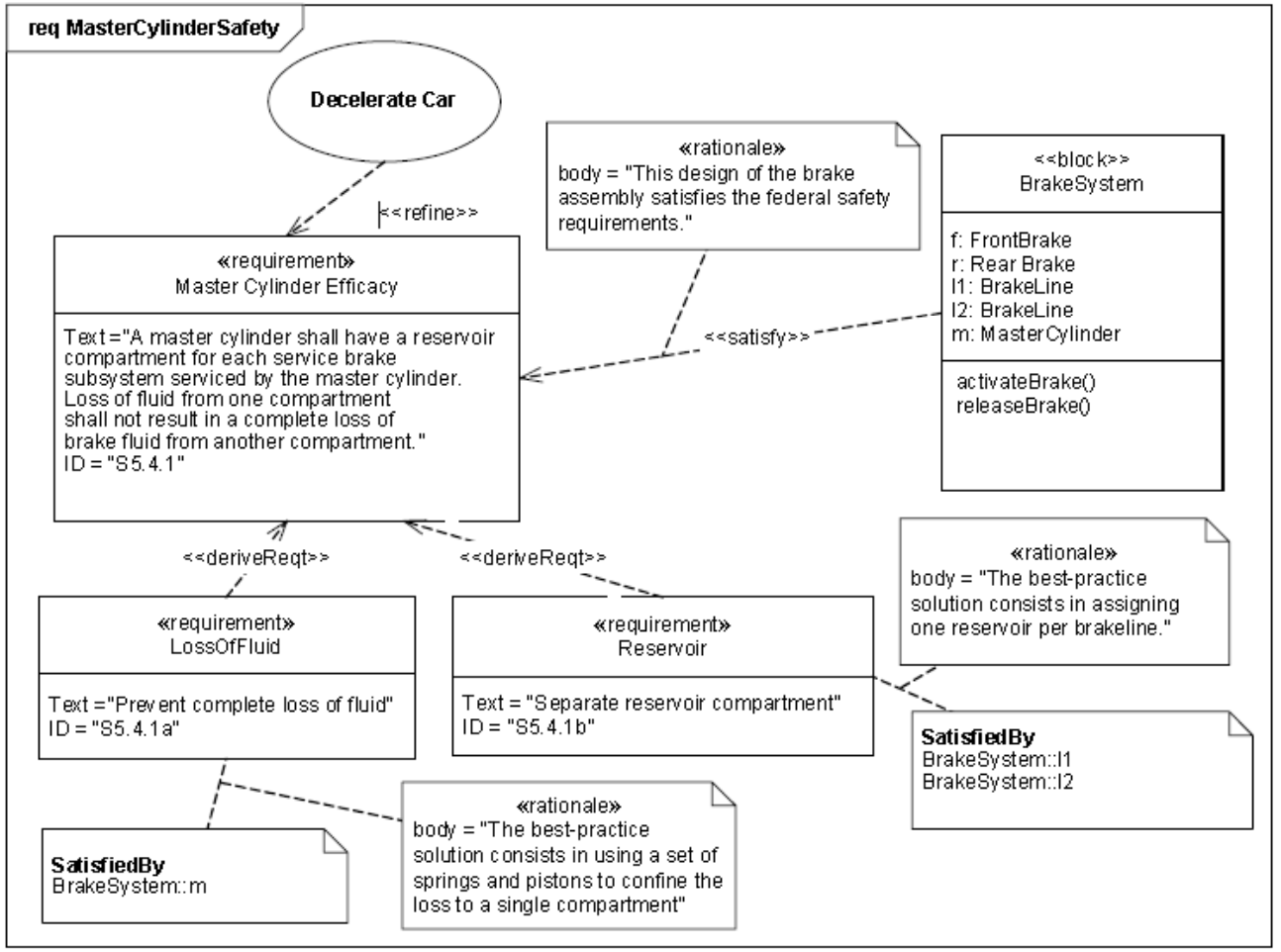
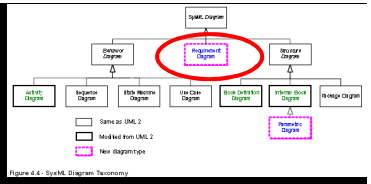


Figure 16.4 - Links between requirements and design

SYSTEMVARUHUSET™



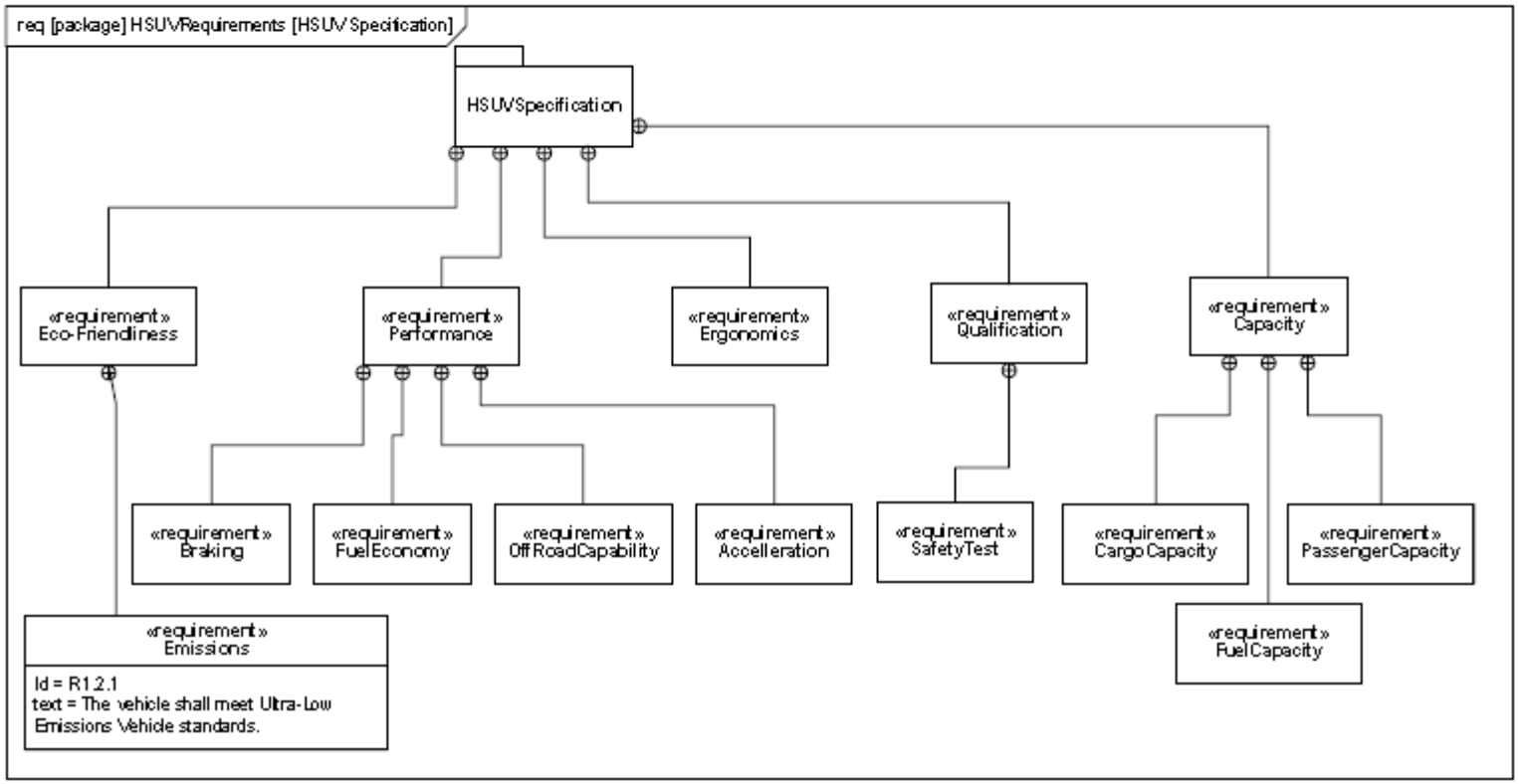
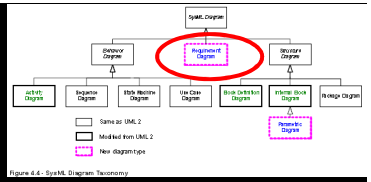


Figure B.11 - Establishing HSUV Requirements Hierarchy (containment) - (Requirements Diagram)

SYSTEMVARUHUSET™



Stereotypade relationer och egenskaper

- Derive
- Related (property)
- Master/slave - copy
- Test cases
- Satisfy
- Verify
- Refine
- Trace
- Rationale

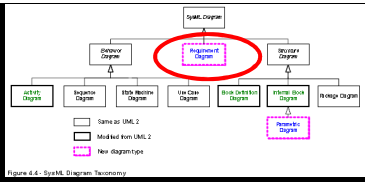
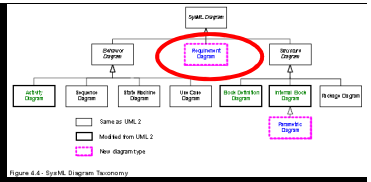


Figure 4.4 - SysML Diagram Taxonomy

SYSTEMVARUHUSET™

Derive

A DeriveReqt relationship is a dependency between two requirements in which a client requirement can be derived from the supplier requirement. For example, a system requirement may be derived from a business need, or lower-level requirements may be derived from a system requirement. As with other dependencies, the arrow direction points from the derived (client) requirement to the (supplier) requirement from which it is derived.



SYSTEMVARUHUSET™

Exempel

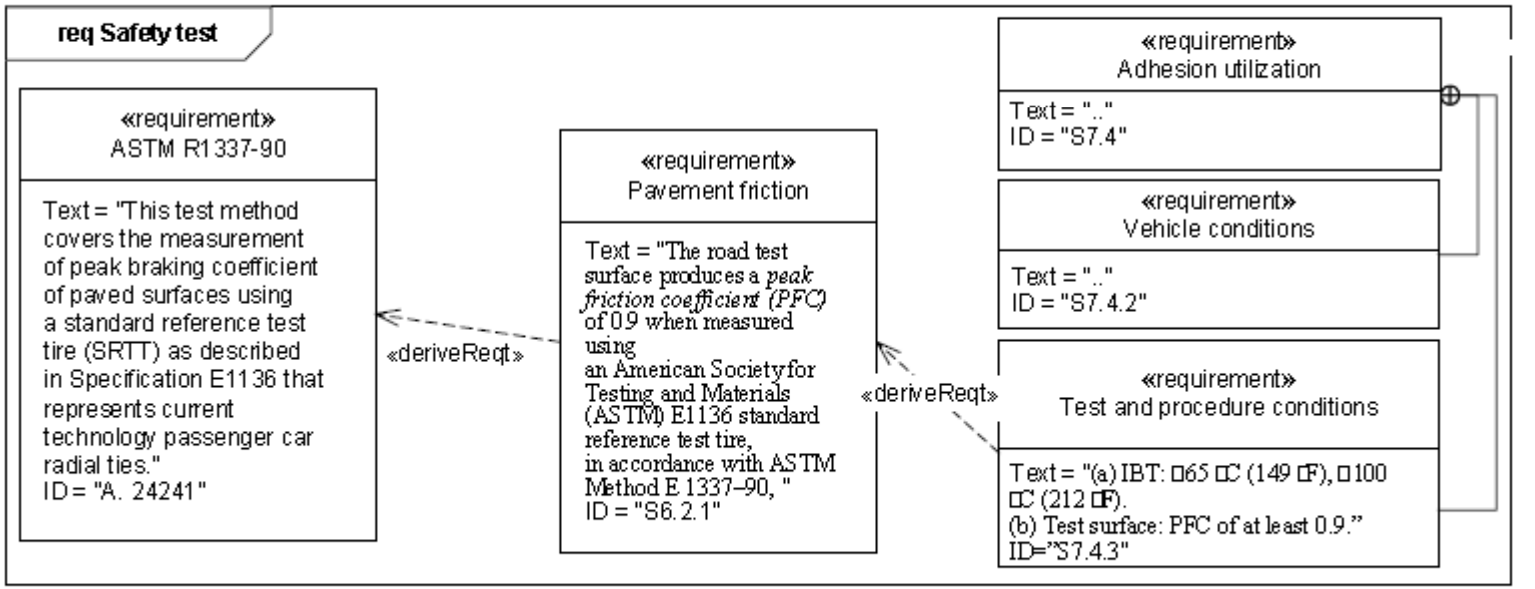
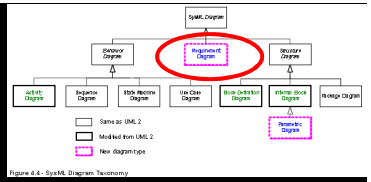


Figure 16.3 - Requirements Derivation



SYSTEMVARUHUSET™

Related (property)

This stereotype is used to add properties to those elements that are related to requirements via the various dependencies. The property values are shown using callout notation (i.e., notes) as shown in the diagram element table.

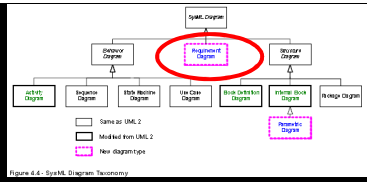
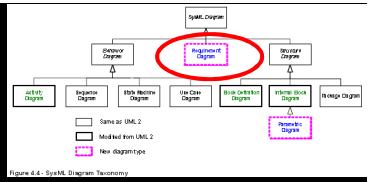


Figure 4.4 - SysML Diagram Taxonomy

SYSTEMVARUHUSET™

Copy

- A Copy relationship is a dependency between a supplier requirement and a client requirement that specifies that the text of the client requirement is a read-only copy of the text of the supplier requirement. A Copy dependency created between two requirements maintains a master/slave relationship between the two elements for the purpose of requirements re-use in different contexts. When a Copy dependency exists between two requirements, the requirement text of the client requirement is a read-only copy of the requirement text of the requirement at the supplier end of the dependency.



SYSTEMVARUHUSET™

Copy

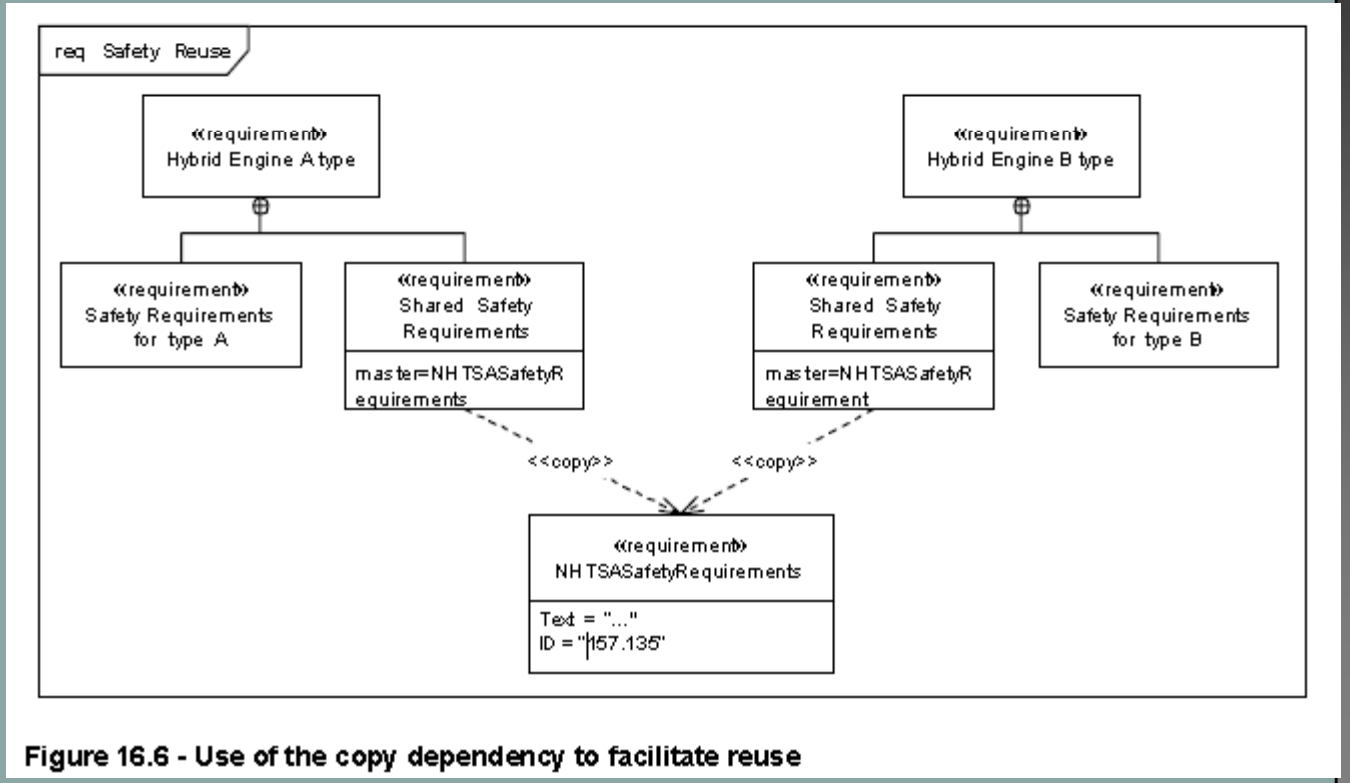
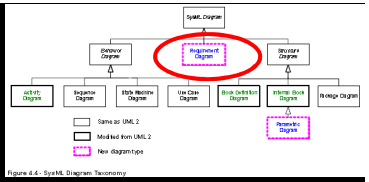


Figure 16.6 - Use of the copy dependency to facilitate reuse



SYSTEMVARUHUSET™

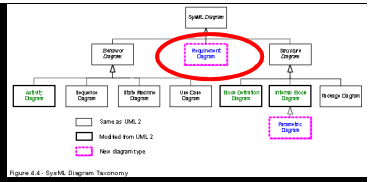
Test case

- A test case is a method for verifying a requirement is satisfied.

Attributes

/verifies: Requirement [*]

Derived from all requirements that are the supplier of a «verify» relationship for which this element is a client.



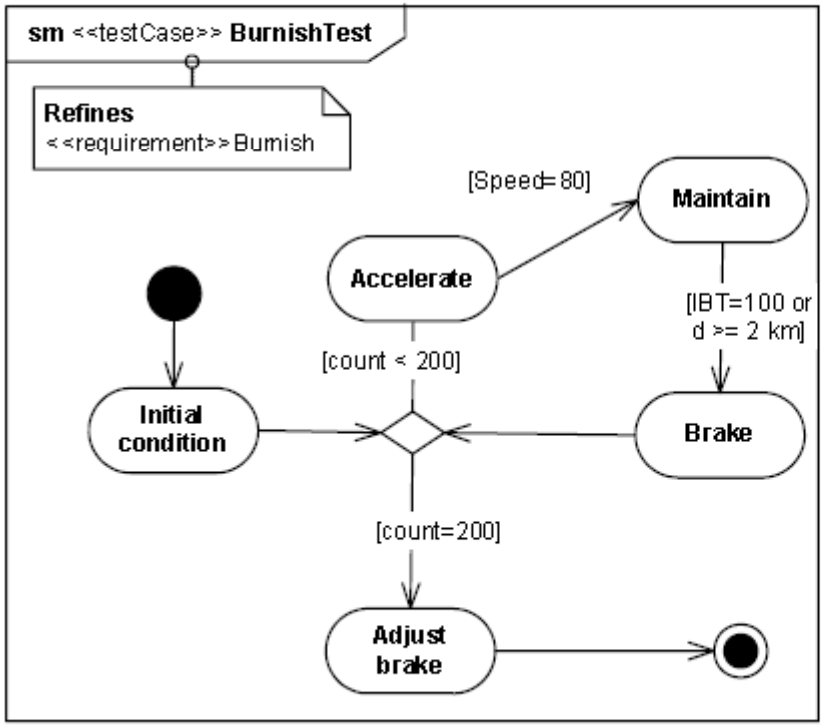
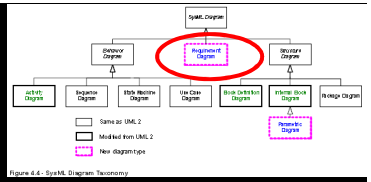


Figure 16.8 - Linkage of a Test Case to a requirement: This figure shows the Test Case as a State Diagram



SYSTEMVARUHUSET™

Satisfy

A Satisfy relationship is a dependency between a requirement and a model element that fulfills the requirement.

The arrow direction points from the satisfying (client) model element to the (supplier) requirement that is satisfied.

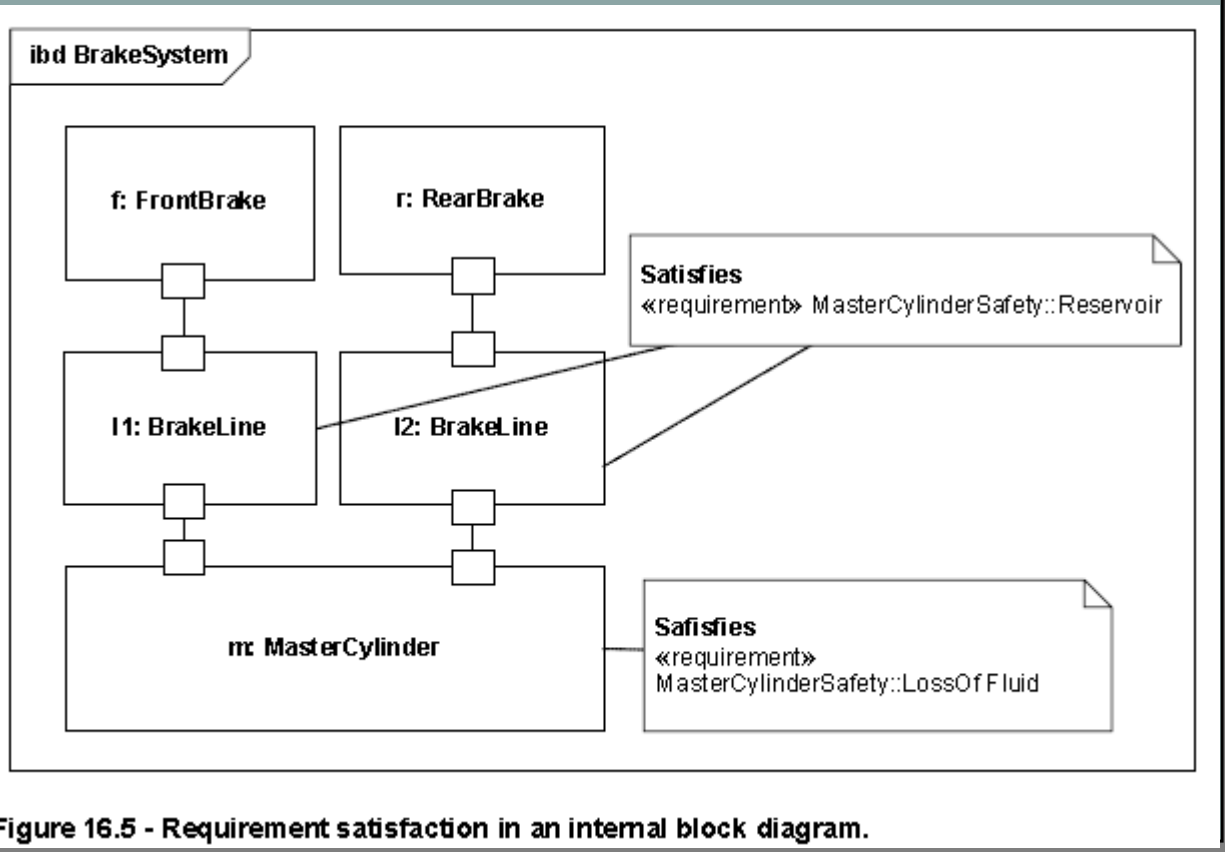
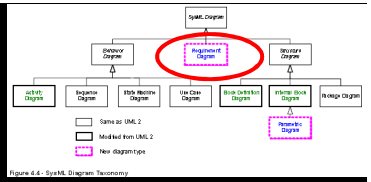


Figure 16.5 - Requirement satisfaction in an internal block diagram.



SYSTEMVARUHUSET™

Verify

A Verify relationship is a dependency between a requirement and a test case that can determine whether a system fulfills the requirement.

The arrow direction points from the (client) test case to the (supplier) requirement.

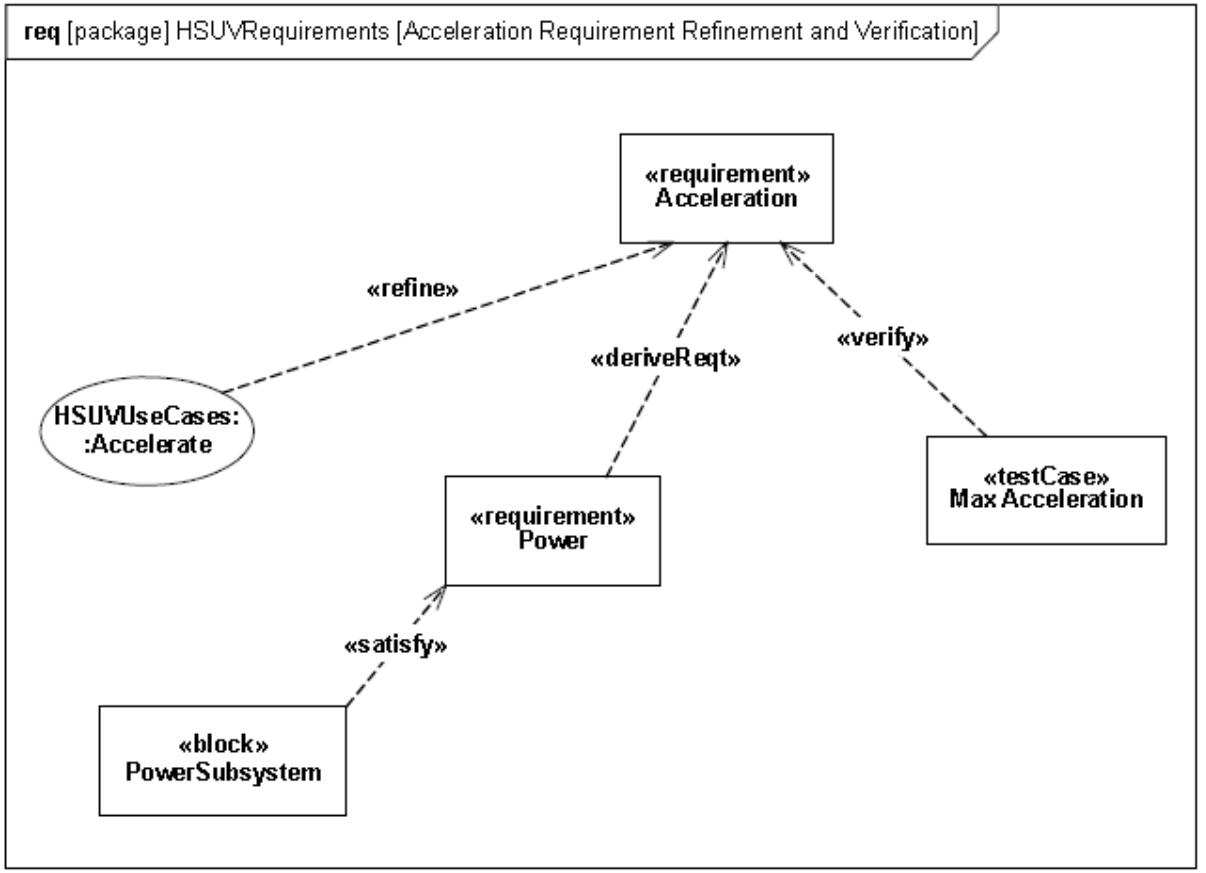
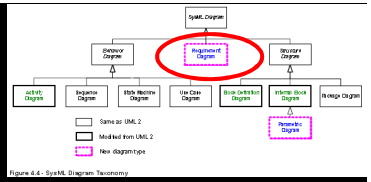


Figure B.13 - Acceleration Requirement Relationships (Requirements Diagram)



SYSTEMVARUHUSET™

Requirements tables

table [requirement] Performance [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.

SYSTEMVARUHUSET™

Allocations

Allocation is the term used by systems engineers to denote the organized cross-association (mapping) of elements within the various structures or hierarchies of a user model.

The concept of "allocation" requires flexibility suitable for abstract system specification, rather than a particular constrained method of system or software design. System modelers often associate various elements in a user model in abstract, preliminary, and sometimes tentative ways.

Allocations can be used early in the design as a precursor to more detailed rigorous specifications and implementations. The allocation relationship can provide an effective means for navigating the model by establishing cross relationships, and ensuring the various parts of the model are properly integrated.

SysML does not try to limit the use of the term "allocation," but provides a basic capability to support allocation in the broadest sense. It does include some specific subclasses of allocation for allocating behavior, structure, and flows. A typical example is the allocation of activities to blocks (e.g., functions to components).

SYSTEMVARUHUSET™

Allocate

Allocate is a dependency based on UML::abstraction. It is a mechanism for associating elements of different types, or in different hierarchies, at an abstract level. Allocate is used for assessing user model consistency and directing future design activity. It is expected that an «allocate» relationship between model elements is a precursor to a more concrete relationship between the elements, their properties, operations, attributes, or sub-classes.

Constraints

A single «allocate» dependency shall have only one supplier (from), but may have one or many clients (to). If subtypes of the «allocate» dependency are introduced to represent more specialized forms of allocation, then they should have constraints applied to supplier and client as appropriate.

SYSTEMVARUHUSET™

Behavior allocations

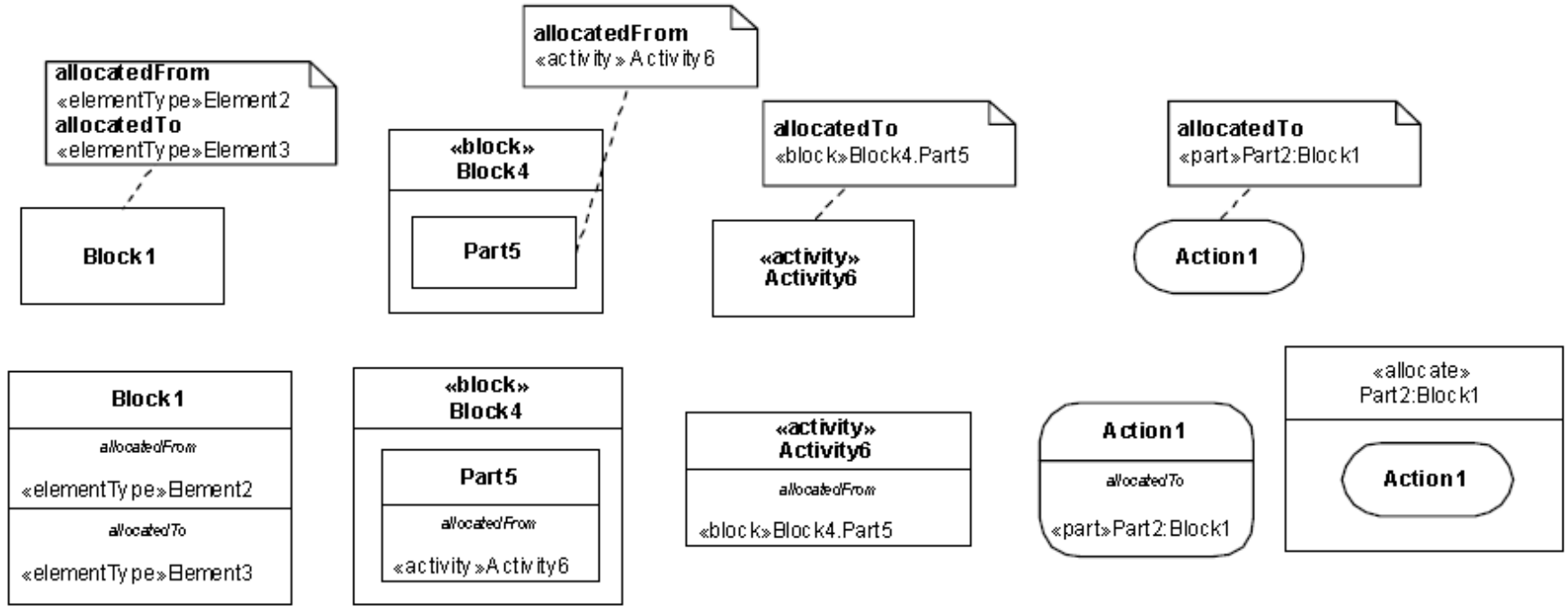
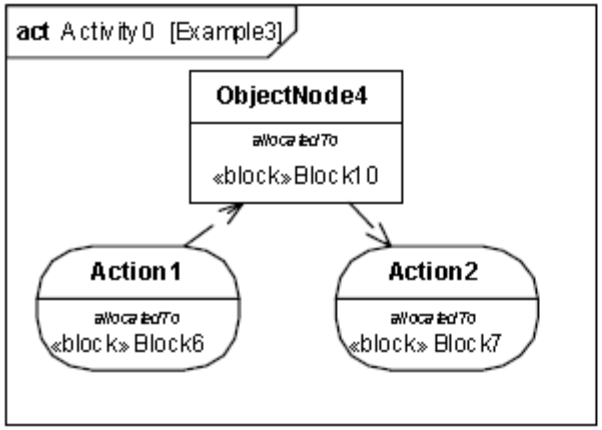


Figure 15.4 - Behavior allocation

SYSTEMVARUHUSET™

Flow allocations



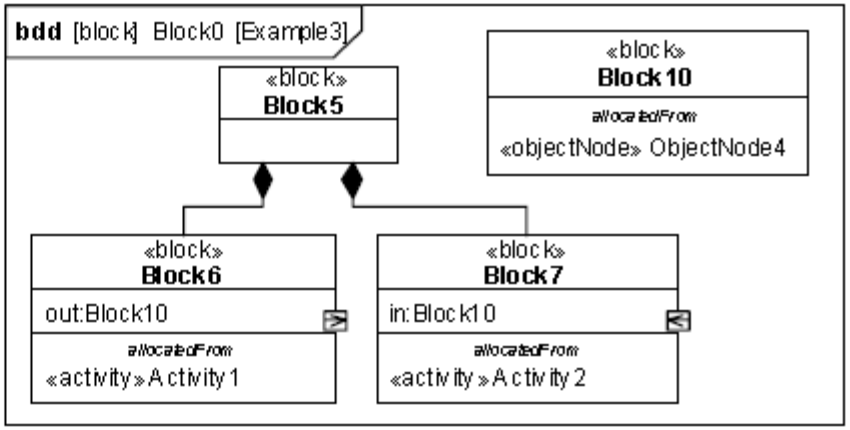


Figure 15.7 - Example of flow allocation from ObjectNode to FlowProperty

SYSTEMVARUHUSET™

Structural allocations

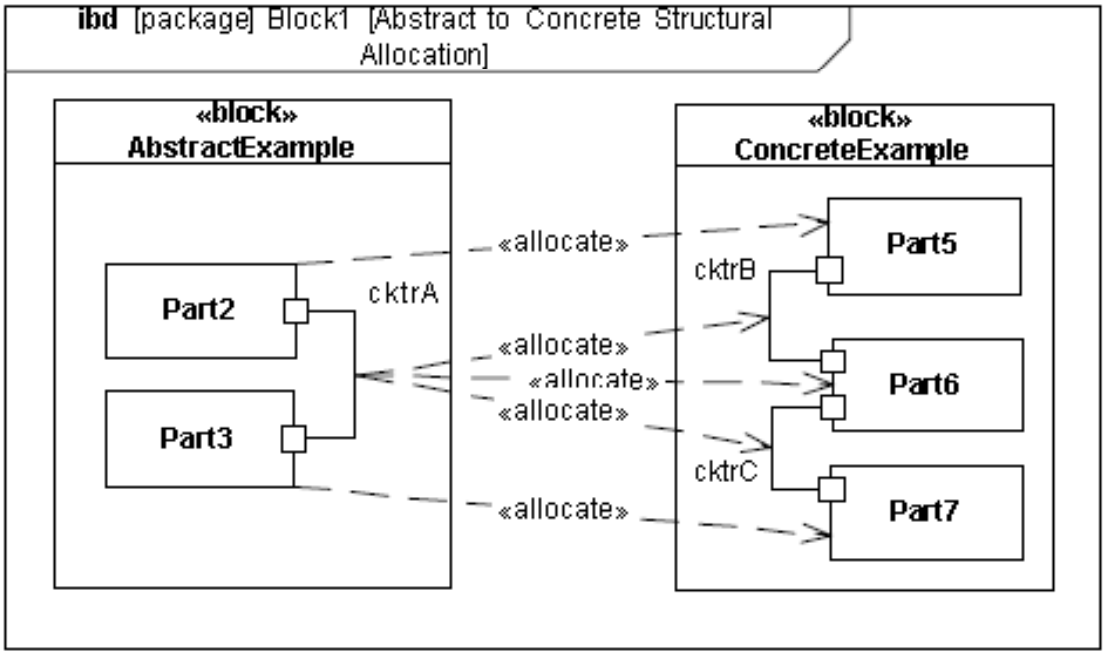
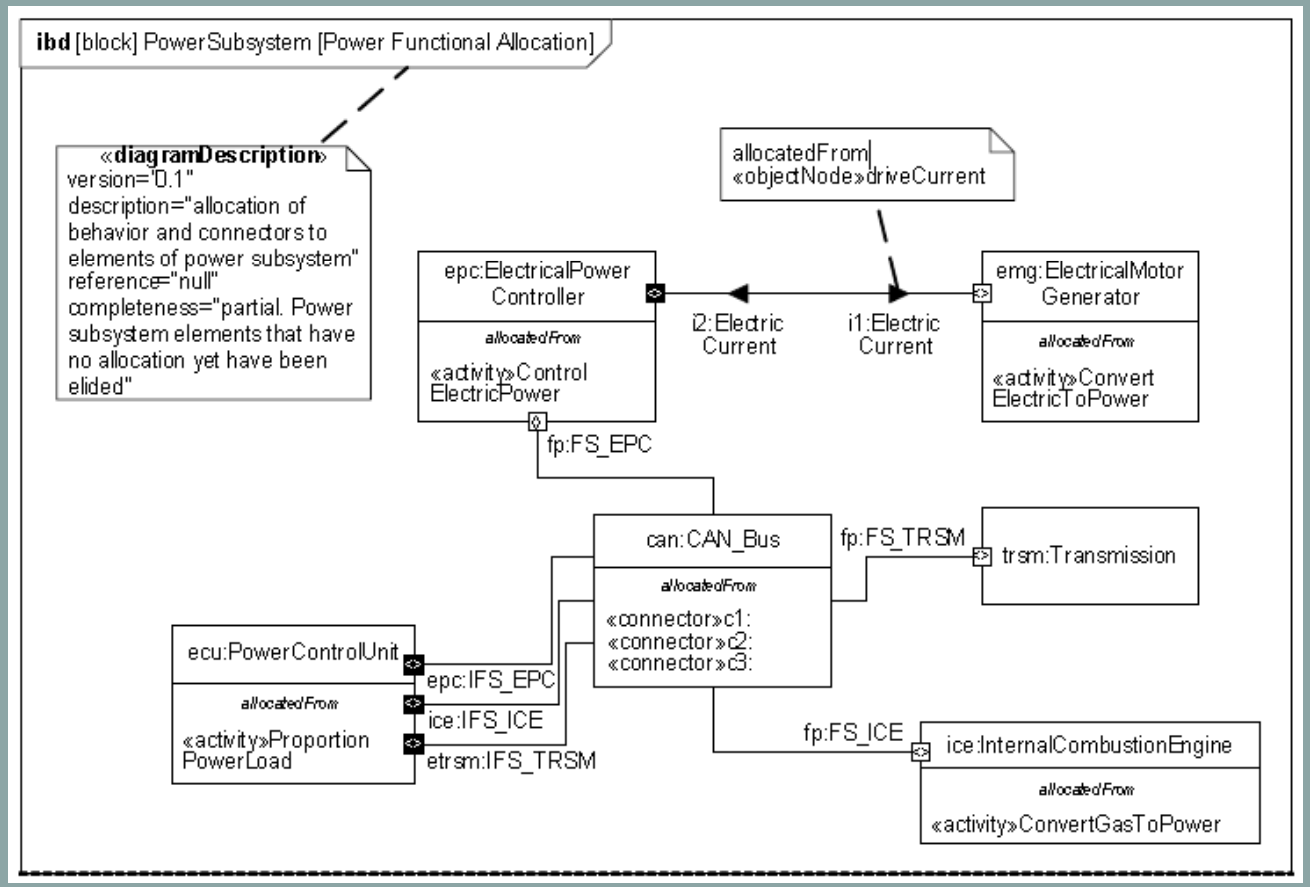


Figure 15.8 - Example of Structural Allocation

Allocations

SYSTEMVARUHUSET™



Allocations

SYSTEMVARUHUSET™

Exempel

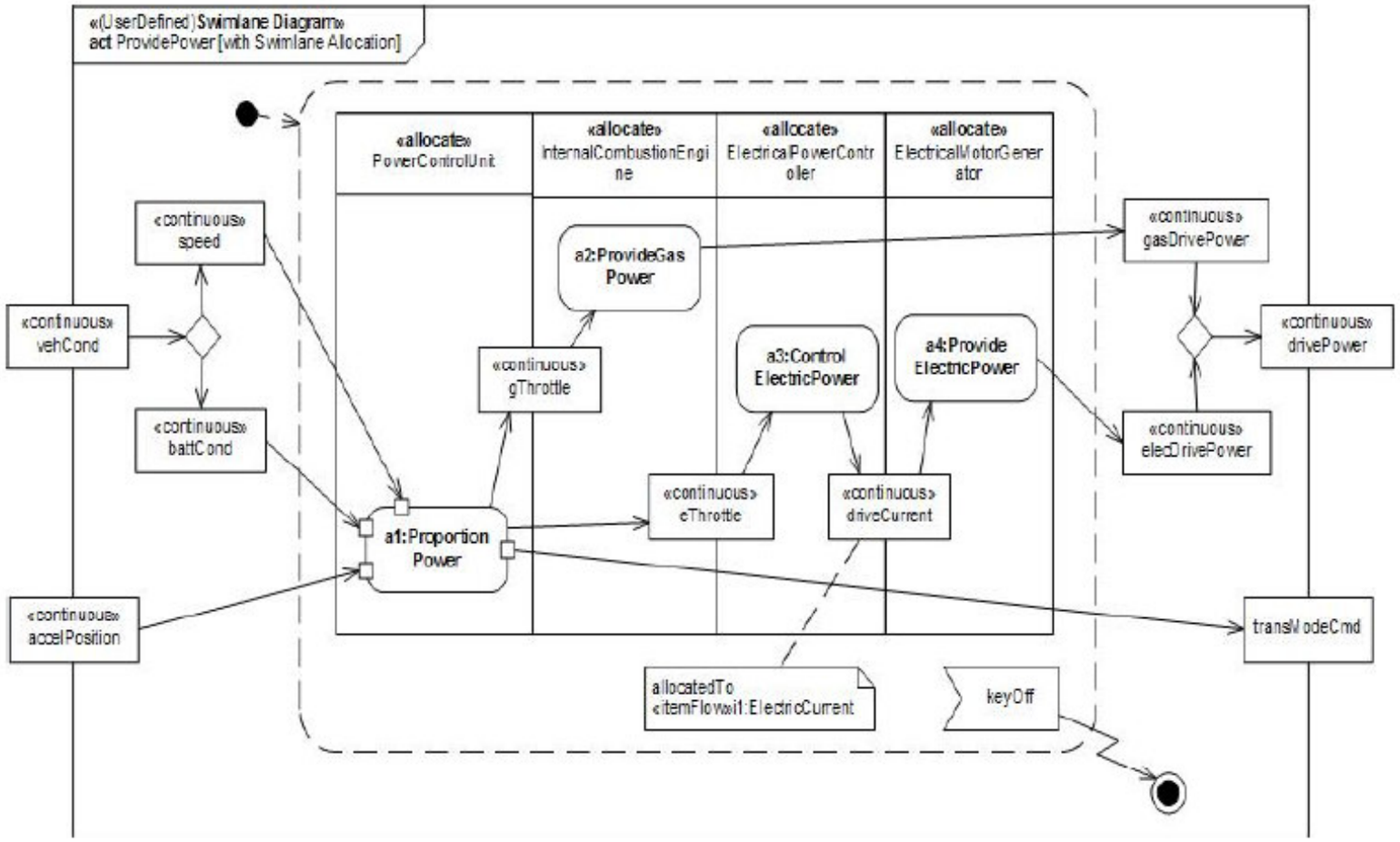


Figure 15.9 - AllocateActivityPartitions (Swimlanes) for HybridSUV Accelerate Example

SYSTEMVARUHUSET™

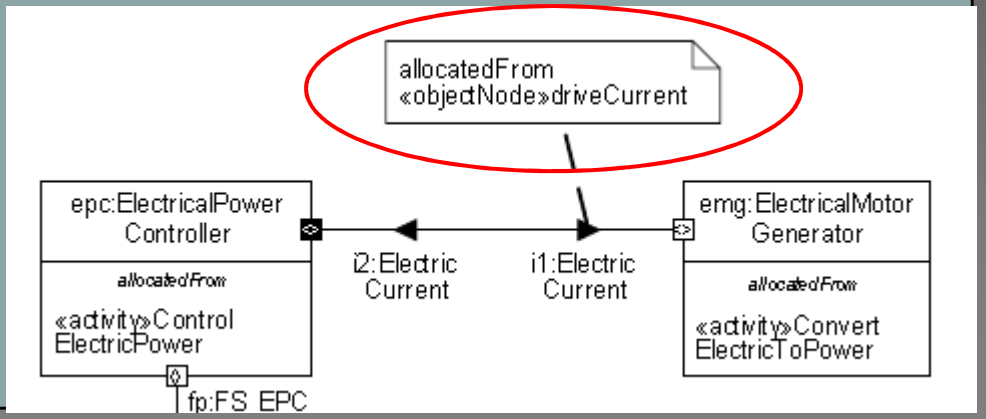
Profiles and model libraries

- The Profiles package contains mechanisms that allow metaclasses from existing metamodels to be extended to adapt them for different purposes.
- The stereotype is the primary mechanism used to create profiles to extend the metamodel. Stereotypes are defined by extending a metaclass, and then have them applied to the applicable model elements in the user model.

SYSTEMVARUHUSET™

Callout

The callout notation provides a mechanism for representing relationships between model elements that appear on different diagram kinds. In particular, they are used to represent allocations and requirements, such as the allocation of an activity to a block on a block definition diagram, or showing a part that satisfies a particular requirement on an internal block diagram.



SYSTEMVARUHUSET™

Obligatoriska ramar

Figure A.2 - Diagram Frame

SYSTEMVARUHUSET™

Diskussion

Positivt

Negativt