SysML v1 – Requirements

Ansgar Radermacher / Asma Smaoui
ansgar.radermacher@cea.fr

- **Acknowledgments** – contains material from our CEA colleagues Shuai Li, Jérémie Tatibouët, François Terrier, Sébastien Gérard

Agenda – SysML v1

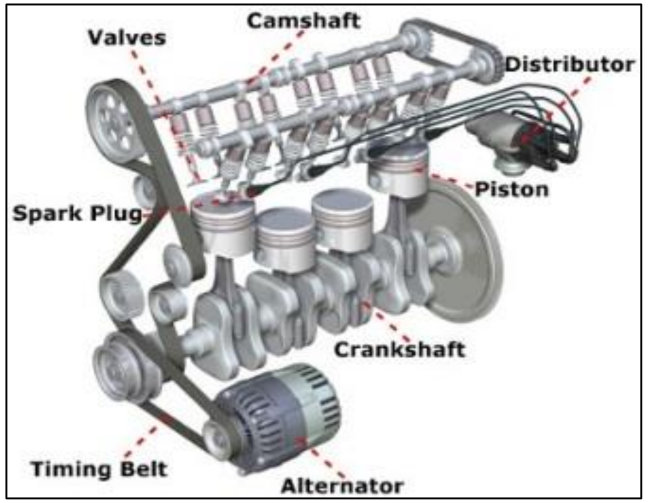
Motivation, history

SysML requirements

Exercise

Motivation

Need a modeling language to model **system engineering** aspects that can not be expressed with UML



© http://www.driving-test-success.com

Points of interest: piston diameter, cylinder volume, weight of components, etc.

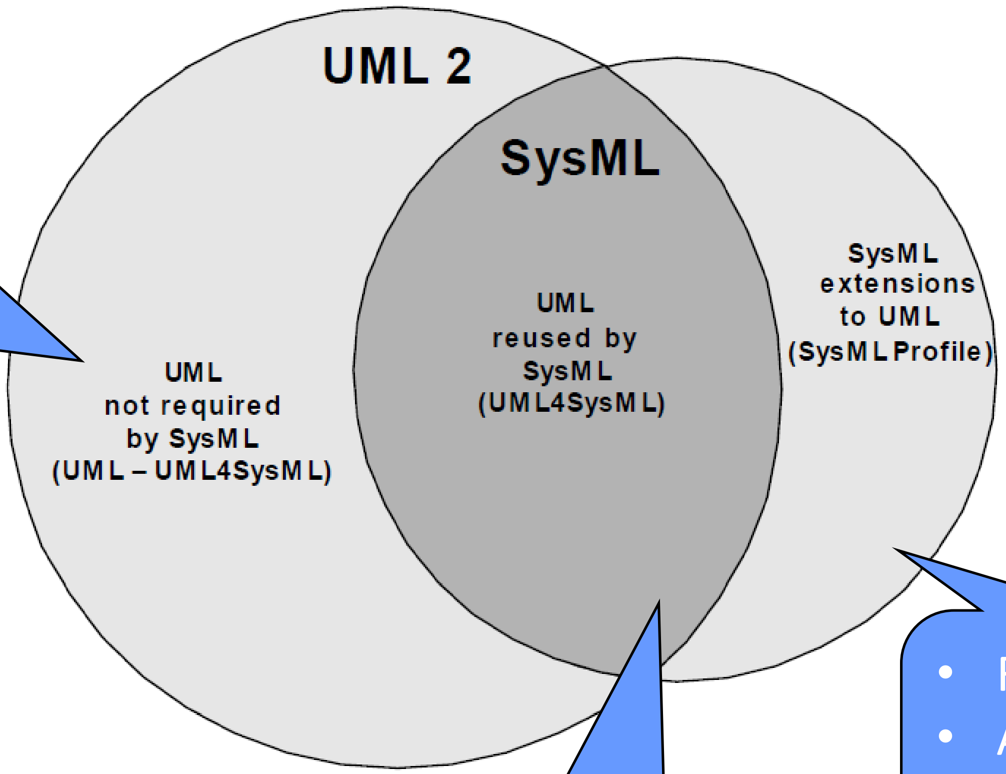
History

- 01/2001: International Council on Systems Engineering (INCOSE) start Model Driven Systems Design workgroup to customize UML for systems engineering applications.
- 07/2001: INCOSE and OMG charter Systems Engineering Domain Special Interest Group (SE DSIG)
- 03/2003: SE DSIG develops requirements for the modeling language => *UML for Systems Engineering Request for Proposal* (RFP)
- 2003: Joint response to UML for SE RFP, “Systems Modeling Language”, aka SysML, logo, language design team
- 01/2005: SysML v0.9 draft, *open source specification*

History

- 11/2005: SysML v1.0 alpha, but competing SysML specification proposals
- 07/2006: Merge Team was proposed and voted
- 01/2007: OMG SysML v1.0, as SysML is derived from open source SysML, it also includes an open source license for distribution and use.
- … several revisions
- 12/2019: OMG SysML v1.6
- 2017: Published by ISO as international standard, ISO/IEC 19514:2017 (Information technology – OMG systems modeling language)
- > 2018: OMG works on next generation and issues an RFP for **SysML v2**

SysML: a UML specialization for system modeling

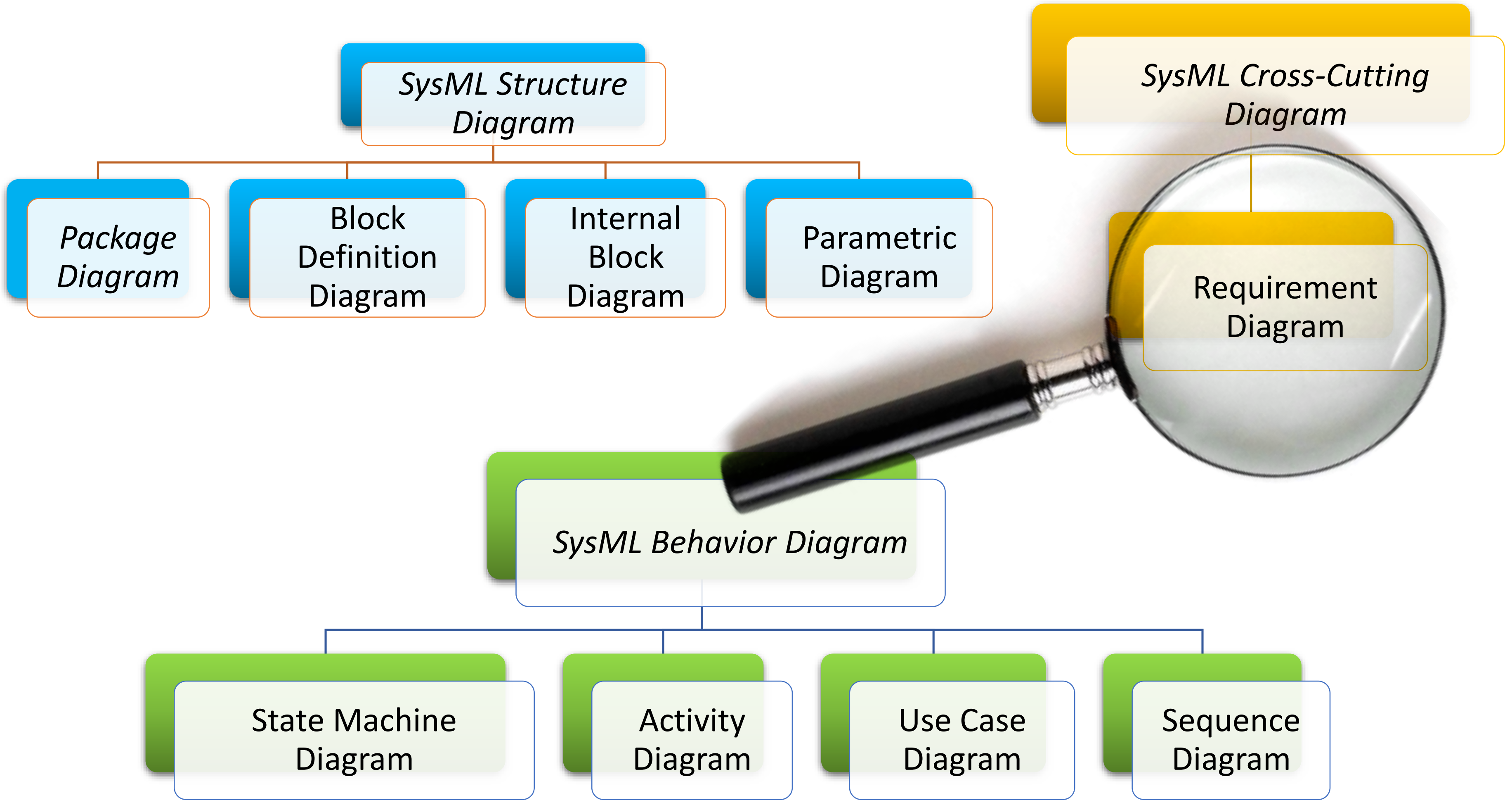

UML 2
SysML
UML not required by SysML (UML – UML4SysML)
UML reused by SysML (UML4SysML)
SysML extensions to UML (SysML Profile)

- Component
- Deployment
- Collaboration
- Template

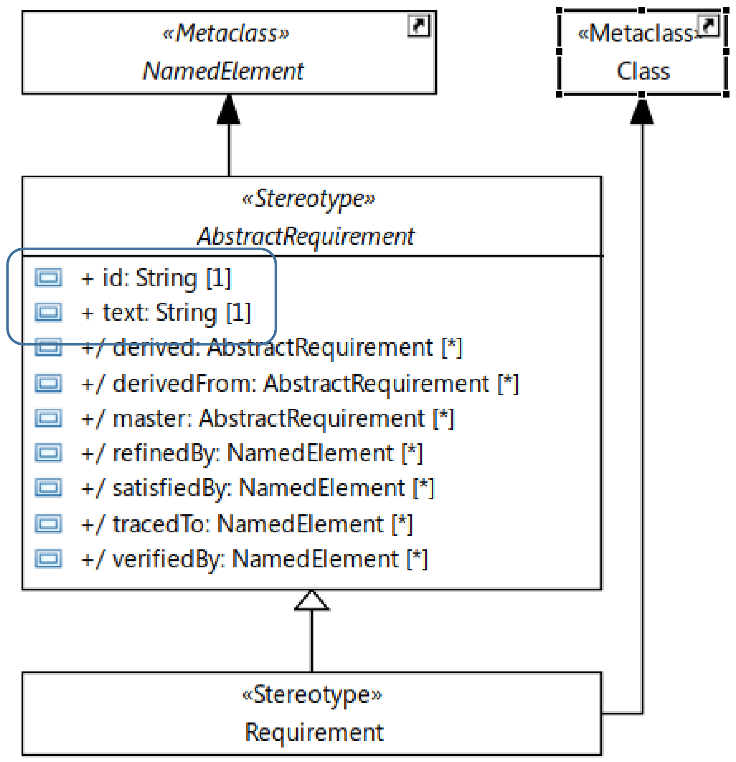
- Behavior
- Types

- Requirements
- Allocation
- Blocks
- Parametrics

Requirement Diagram



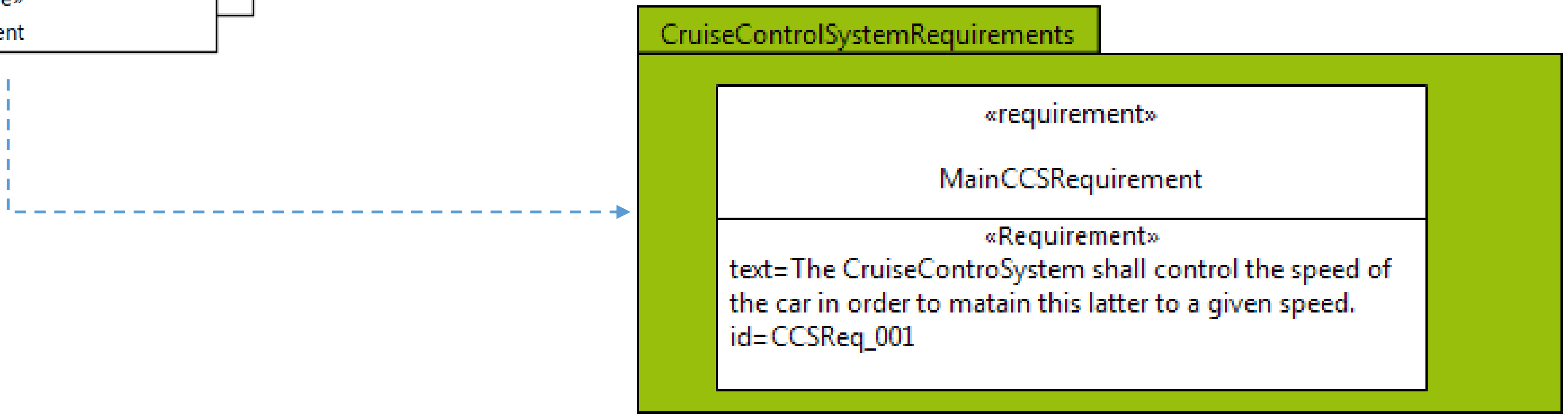
Details of the «Requirement» Concept


«Metaclass»
NamedElement
«Metaclass»
Class
«Stereotype»
AbstractRequirement
+ id: String [1]
+ text: String [1]
+/ derived: AbstractRequirement [*]
+/ derivedFrom: AbstractRequirement [*]
+/ master: AbstractRequirement [*]
+/ refinedBy: NamedElement [*]
+/ satisfiedBy: NamedElement [*]
+/ tracedTo: NamedElement [*]
+/ verifiedBy: NamedElement [*]
«Stereotype»
Requirement

- Extends meta-class Class
- Inherits attributes from «AbstractRequirement»
- Properties
  - id = unique identifier (defined as a string)
  - text = requirement description

Profile Level

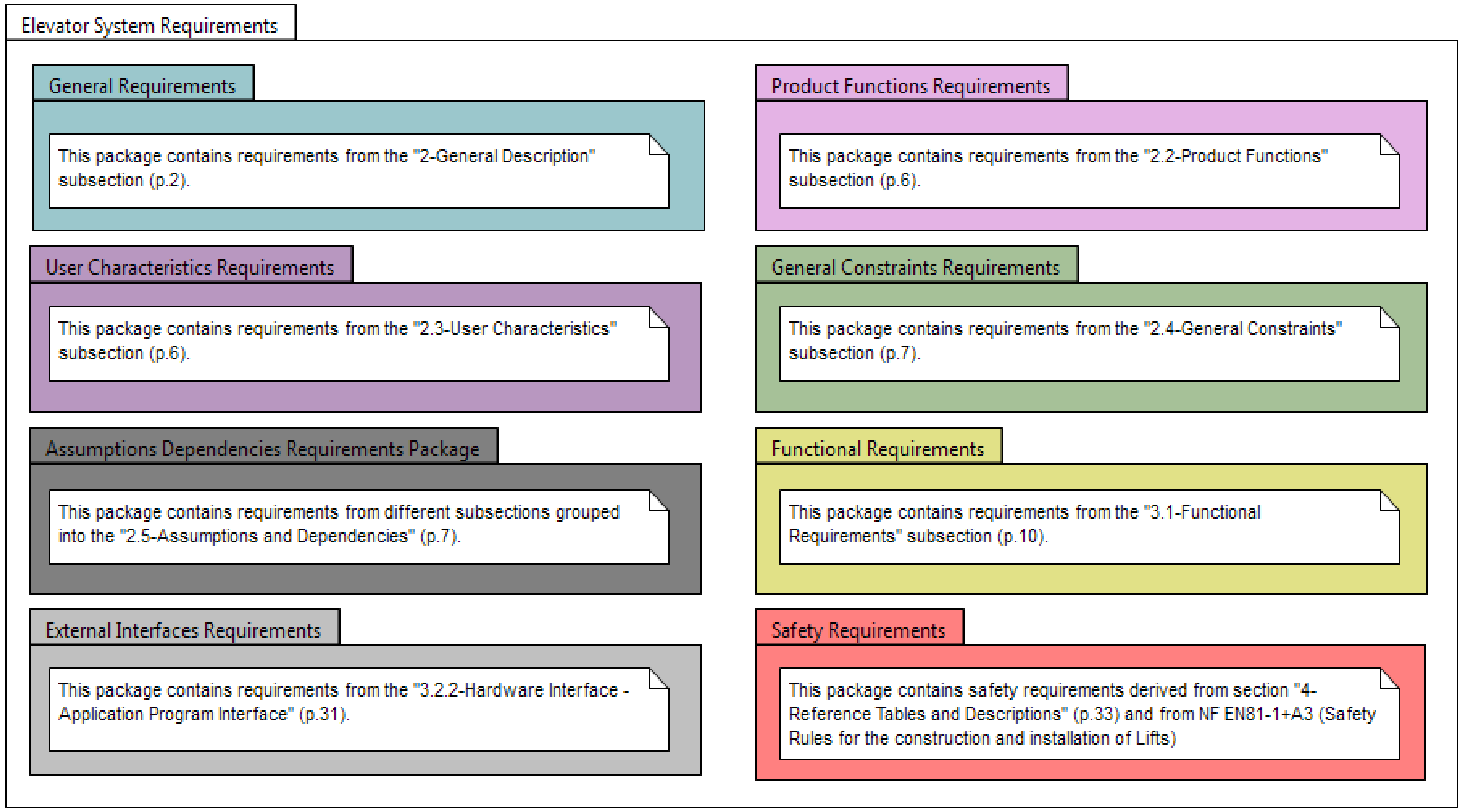
User Level


CruiseControlSystemRequirements
«requirement»
MainCCSRequirement
«Requirement»
text=The CruiseControSystem shall control the speed of the car in order to matain this latter to a given speed.
id=CCSReq_001

Organization/Structuration of Requirements

- Organize requirements in packages (hierarchical structuration).
  - Can import other packages or model elements in other packages

- Example


Elevator System Requirements
General Requirements
This package contains requirements from the "2-General Description" subsection (p.2).
Product Functions Requirements
This package contains requirements from the "2.2-Product Functions" subsection (p.6).
User Characteristics Requirements
This package contains requirements from the "2.3-User Characteristics" subsection (p.6).
General Constraints Requirements
This package contains requirements from the "2.4-General Constraints" subsection (p.7).
Assumptions Dependencies Requirements Package
This package contains requirements from different subsections grouped into the "2.5-Assumptions and Dependencies" (p.7).
Functional Requirements
This package contains requirements from the "3.1-Functional Requirements" subsection (p.10).
External Interfaces Requirements
This package contains requirements from the "3.2.2-Hardware Interface - Application Program Interface" (p.31).
Safety Requirements
This package contains safety requirements derived from section "4-Reference Tables and Descriptions" (p.33) and from NF EN81-1+A3 (Safety Rules for the construction and installation of Lifts)

Concepts for Traceability

- **Traceability** includes several complementary notions [From Gotel, 1994]
  - Relations among requirements (behaviors refining requirements):

    Answer the question: “*How can a requirement evolve?*”

    Decomposition, **refinement**, **copy, derive**
  - Relations between the requirements and the system

    Answer the question: “*How can a system be defined to ensure requirements?*”

    Relations such as: **satisfy**
  - Relations between requirements and V&V:

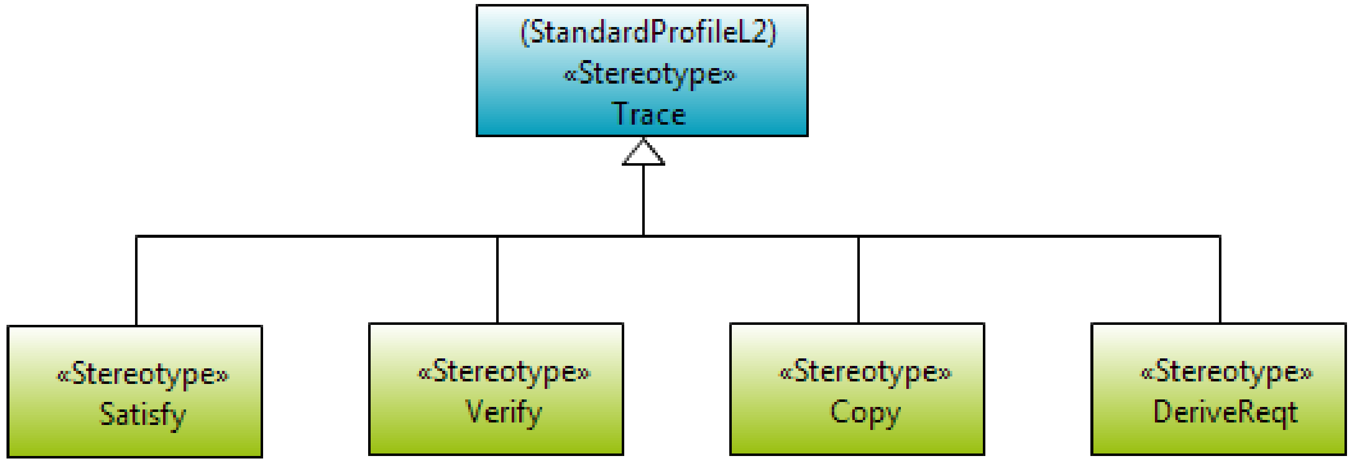
    Answer the question: “*How does a system meet its requirements?*”

    A system can be proved with respect to requirements: test cases, code review, …

    Relations such as **verify**

Requirements Traceability: «Trace» concept

Refine and decomposition already present in UML. New SysML stereotypes for satisfy, verify, copy and derive.



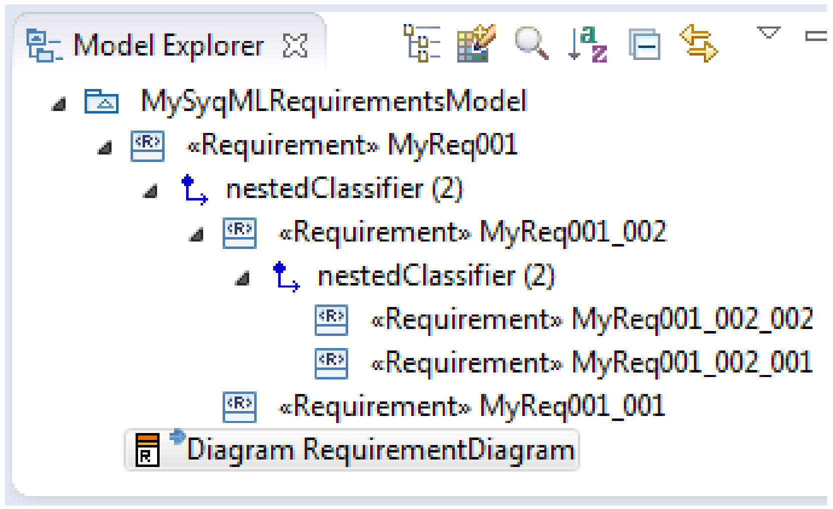
The generic «UML::trace» relationship has *very generic (i.e., weak) semantics!*

➔ Not recommended to use in conjunction with other SysML requirements relationships.

Decomposition

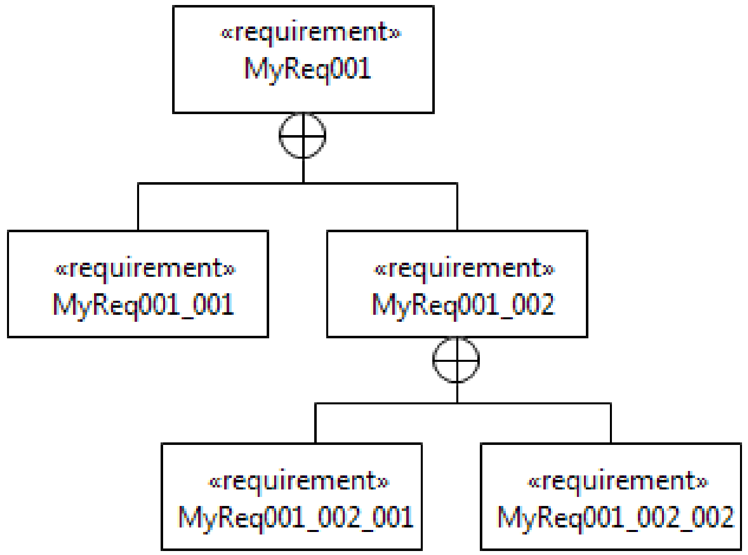
Composite requirements (i.e., hierarchical description)

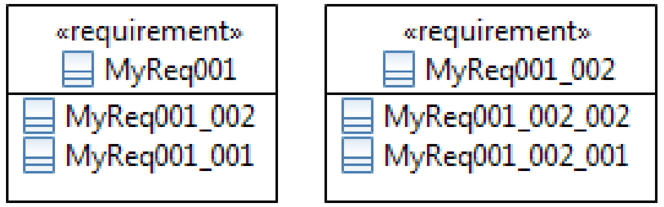
- Use UML namespace containment mechanism.



Model Explorer

- MySyqMLRequirementsModel
  - «Requirement» MyReq001
    - nestedClassifier (2)
      - «Requirement» MyReq001_002
        - nestedClassifier (2)
          - «Requirement» MyReq001_002_002
          - «Requirement» MyReq001_002_001
      - «Requirement» MyReq001_001
  - Diagram RequirementDiagram


«requirement» MyReq001
«requirement» MyReq001_001
«requirement» MyReq001_002
«requirement» MyReq001_002_001
«requirement» MyReq001_002_002



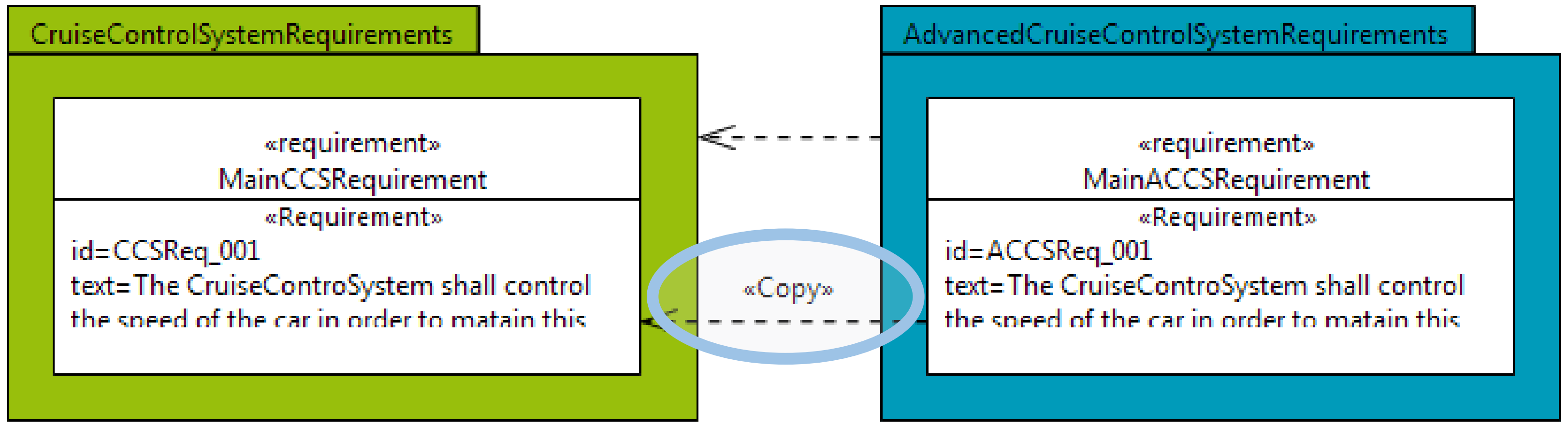
Requirements reuse: the «Copy» concept

⚠ The use of namespace containment to specify requirements hierarchies precludes reusing requirements in different contexts since a given model element can only exist in one namespace!

Slave requirements for enabling reuse

- A **slave** requirement is a requirement whose text property is a read-only copy of the text property of a master requirement.
- The master/slave relationship is denoted via a «Copy» dependency.

Example


CruiseControlSystemRequirements
«requirement»
MainCCSRequirement
«Requirement»
id=CCSReq_001
text=The CruiseControSystem shall control the speed of the car in order to matain this
«Copy»
AdvancedCruiseControlSystemRequirements
«requirement»
MainACCSRequirement
«Requirement»
id=ACCSReq_001
text=The CruiseControSystem shall control the speed of the car in order to matain this

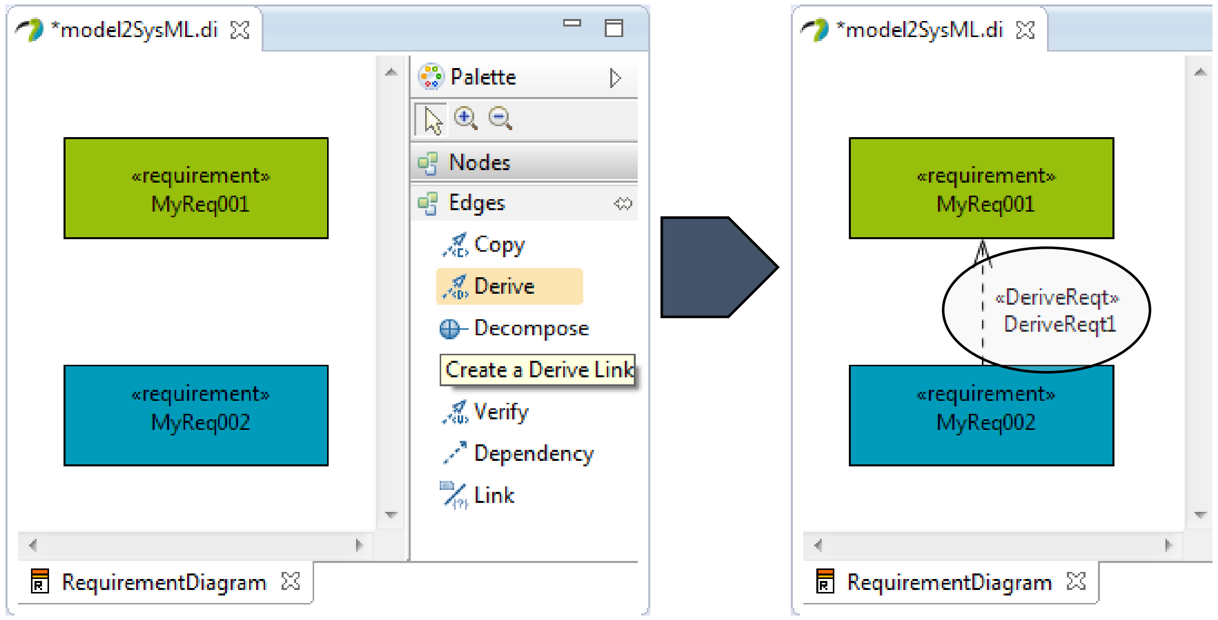
Requirements reuse: the «DeriveReq» concept

Usage

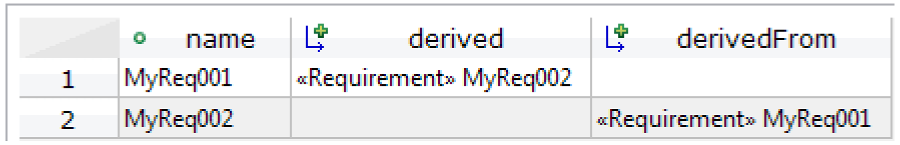
- Relates a derived requirement to its source requirement.

Example

- Vehicle **acceleration** requirement implies derived requirements for engine power, vehicle weight…



Alternative tabular notation:



	name	derived	derivedFrom
1	MyReq001	«Requirement» MyReq002	
2	MyReq002		«Requirement» MyReq001

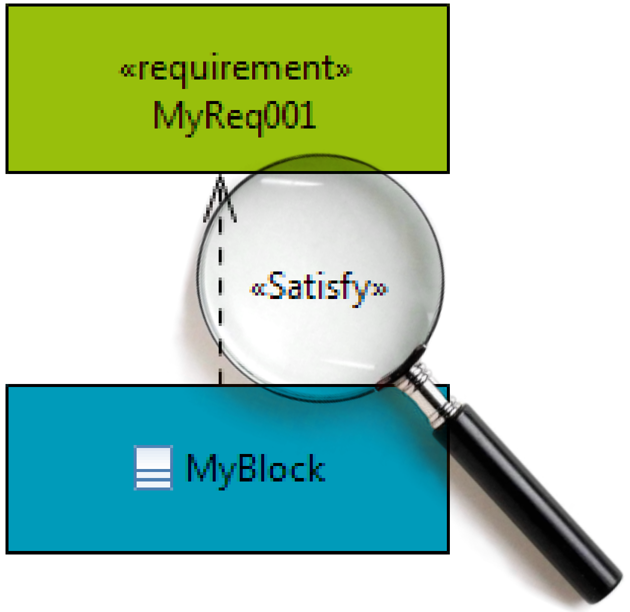
Satisfy and Refine

«Satisfy» Dependency

- Dependency from design or implementation model element to the requirement that it satisfies.

Example

- A block of the design model satisfies a requirement.

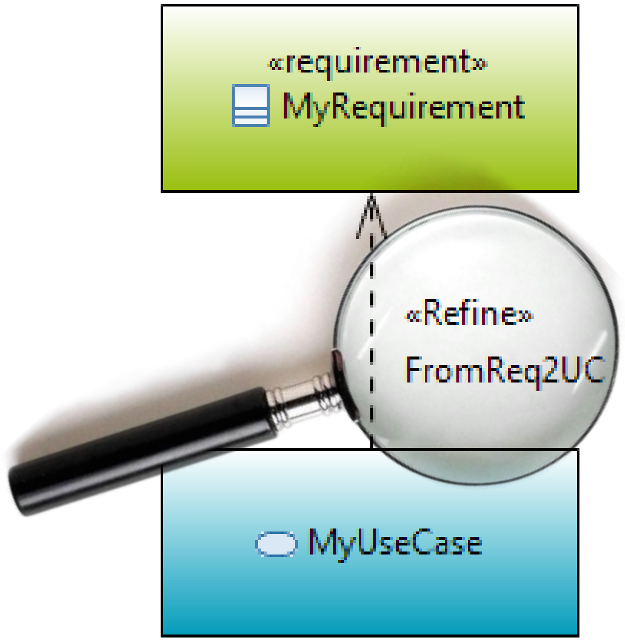


«Refine» Abstraction

- Denote how a model element or set of elements can be used to further refine a requirement.

Example

- Use case used to refine a requirement.



Requirements V&V: the «Verify» and «TestCase» concepts

«Verify» Dependency

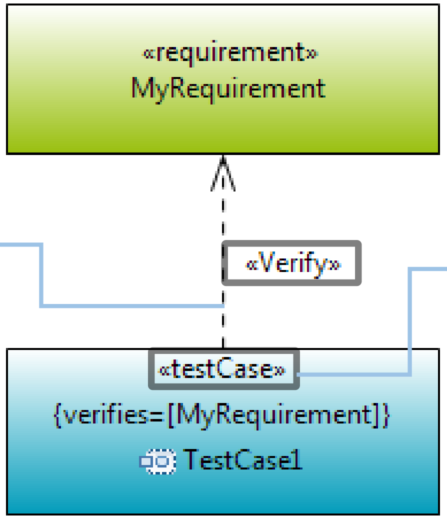
- Used for defining how a test case or other model element verifies a requirement.

«TestCase»

- Represent standard verification method for inspection, analysis, demonstration, or test.
- Extends UML::Operation and UML::Behavior.
- May be used to integrate both SysML and the UML testing profile (http://utp.omg.org/).

Modeling in Papyrus

Model a «Verify» relationship from the model element verifying the requirement (e.g., a test case) and the requirement.

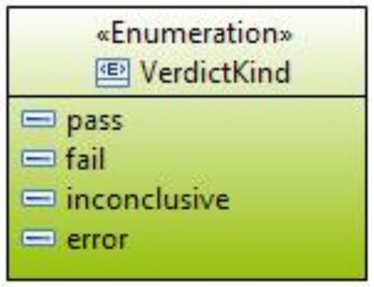


Apply the stereotype «TestCase» to a behavior (e.g., an activity, interaction…) or an operation.

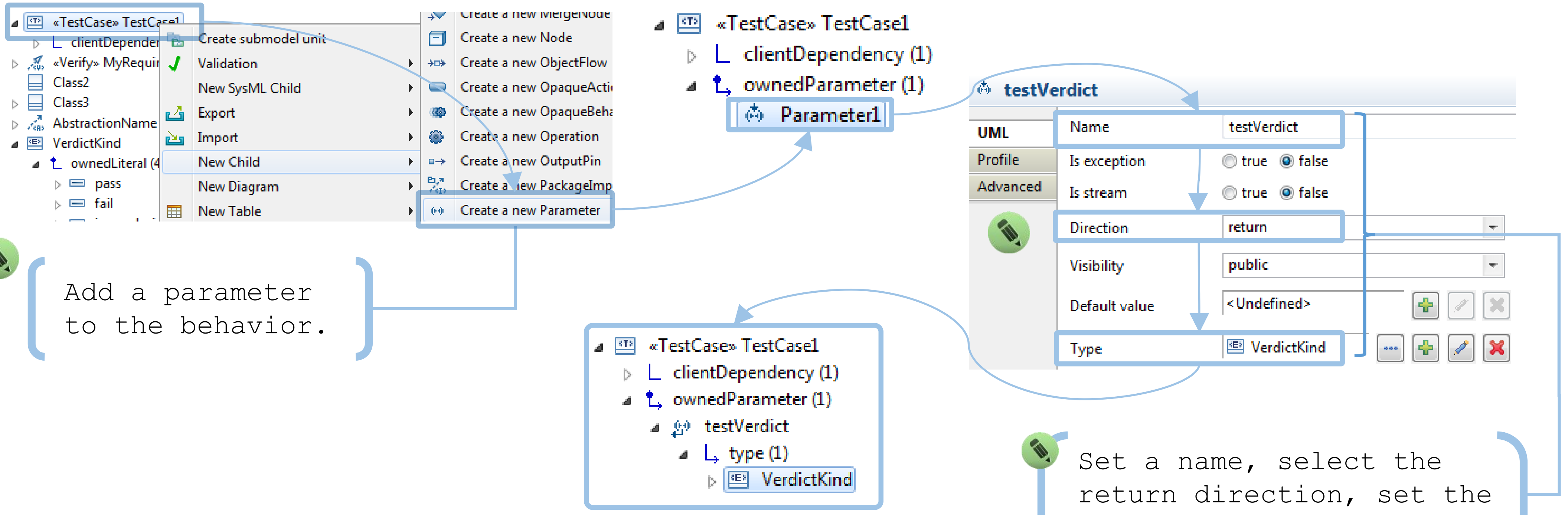
Requirements V&V (Seq.): the VerdictKind concept

VerdictKind (Enumeration)

- Possible literals: pass, fail inconclusive or error
- Used to type the return values of the test case specification



Modeling in Papyrus



Name	testVerdict
Is exception	○ true [✓ false]
Is stream	○ true [✓ false]
Direction	return
Visibility	public
Default value	<Undefined>
Type	VerdictKind

Add a parameter to the behavior.

Set a name, select the return direction, set the type to be VerdictKind

«RequirementRelated» concept

Usage

- Used to add properties to those elements that are related to requirements via the various SysML requirements relationships (e.g., verify and refine).
- Can be applied on UML::NamedElement (i.e., almost on all UML model elements)

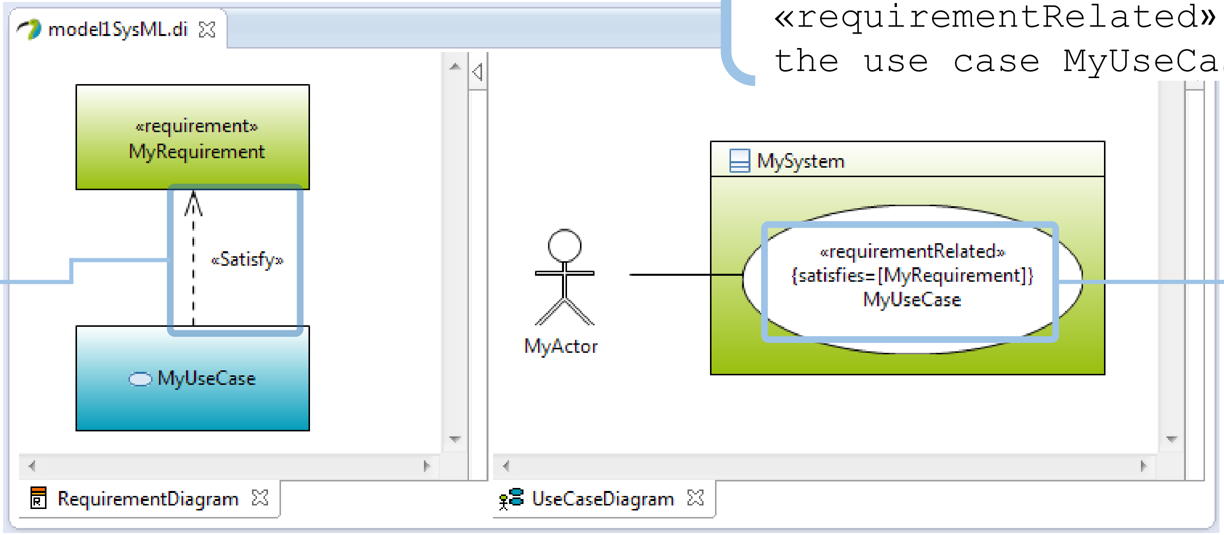
Derived properties

- / tracedFrom: Requirement [*]
- / satisfies: Requirement [*]
- / refines: Requirement [*]
- / verifies: Requirement [*]

Modeling in Papyrus

Model a «Satisfy» relationship from the use case to the requirement.

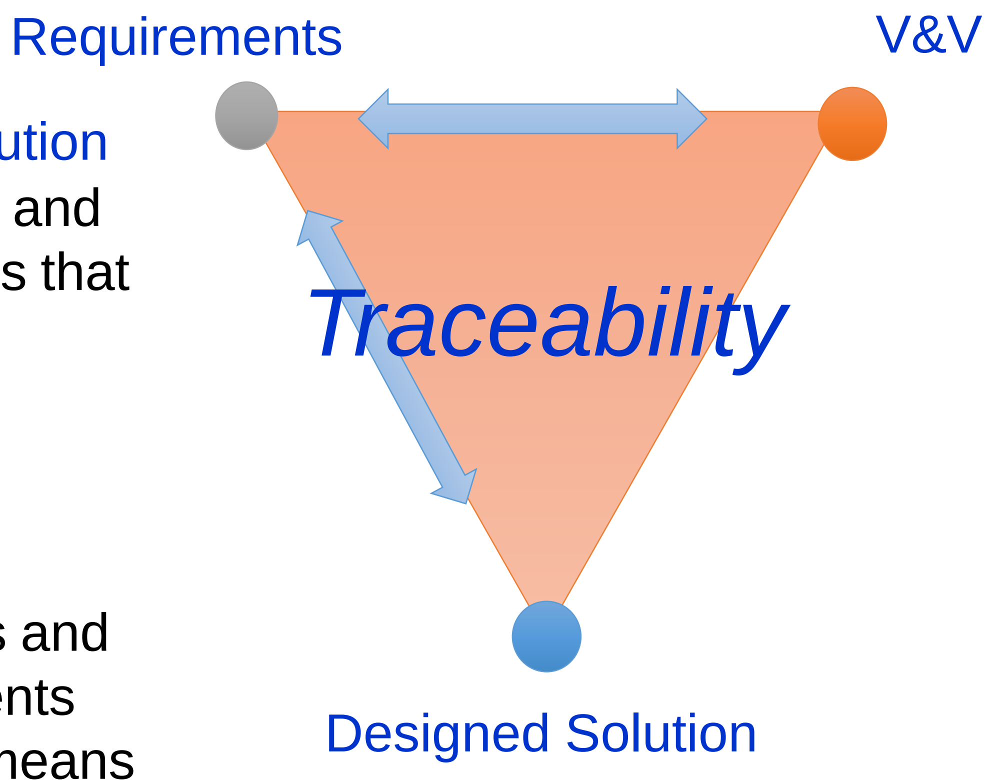
Apply the stereotype «requirementRelated» to the use case MyUseCase.



Requirement TRACEABILITY

✓ Requirements ⟺ Designed Solution
Traceability between requirements and the designed solution demonstrates that the solution satisfies user needs.

✓ Requirements ⟺ V&V
Traceability between requirements and test cases shows which requirements are covered by tests. Tests are a means to verify requirements.



Sample driver assistance system (ADAS)

- CruiseControl: keep set speed
  - Activate via set button
  - Deactivate via cancel button or brake pedal
- EmergencyBreak
  - Execute emergency break if obstacle (e.g. pedestrian) is detected within n meters (5 seconds x current speed)
- Environment
  - Speed sensor,
  - Pedals (acceleration & break)
  - ObstacleDetector: distance to obstacle (if detected)

Hands-On: SysML requirements

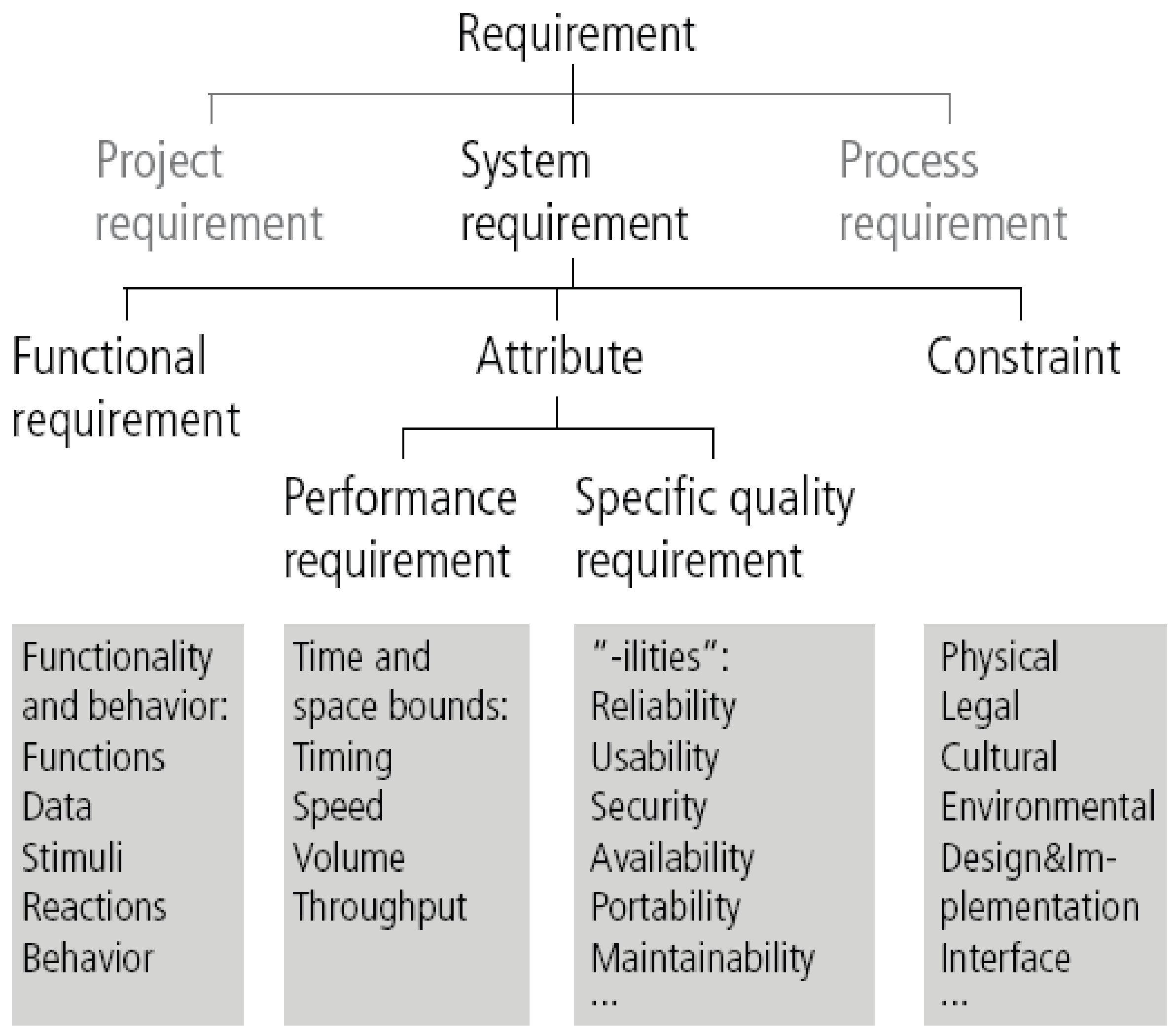
- Your turn
  - Capture the requirements using a SysML requirements diagram.
  - Identify the requirements and decompose them in logically independent requirements.
  - Should the requirements be semantically related to each other, use appropriate relationships (containment, derived requirements…).
  - Should you have to motivate some requirements definitions, use the Rationale stereotype

- Do it in Papyrus
  - Create a package “Requirements” at the root of the model and a requirement diagram in this package.
  - Create a requirement table corresponding to these requirements.

Hands-On: extend SysML requirements

- Create a profile to capture the classification of the following slide
  - Create a Papyrus project « RequirementProfile »
    (choose Profile in the language wizard)
  - Create stereotypes for every class (FunctionalRequirement …) – extend the SysML::Requirements::Requirement stereotype
  - Create an Enumeration for every class and add literals (function, behavior…)
  - In each stereotype create a property "kind" typed with the corresponding Enumeration
  - Save the profile to define it.
  - Apply the profile to the former model and apply appropriate stereotypes to requirements.

Further Requirement classification


Requirement
Project requirement
System requirement
Process requirement
Functional requirement
Attribute
Constraint
Performance requirement
Specific quality requirement
Functionality and behavior: Functions Data Stimuli Reactions Behavior
Time and space bounds: Timing Speed Volume Throughput
"-ilities": Reliability Usability Security Availability Portability Maintainability …
Physical Legal Cultural Environmental Design&Im-plementation Interface …

Extract from M.Glinz. *On Non-Functional Requirements*. Proc. of the 15th IEEE Int. RE Conference, 2007.