FROM RESEARCH TO INDUSTRY

cea tech

REQUIREMENTS ENGINEERING

list

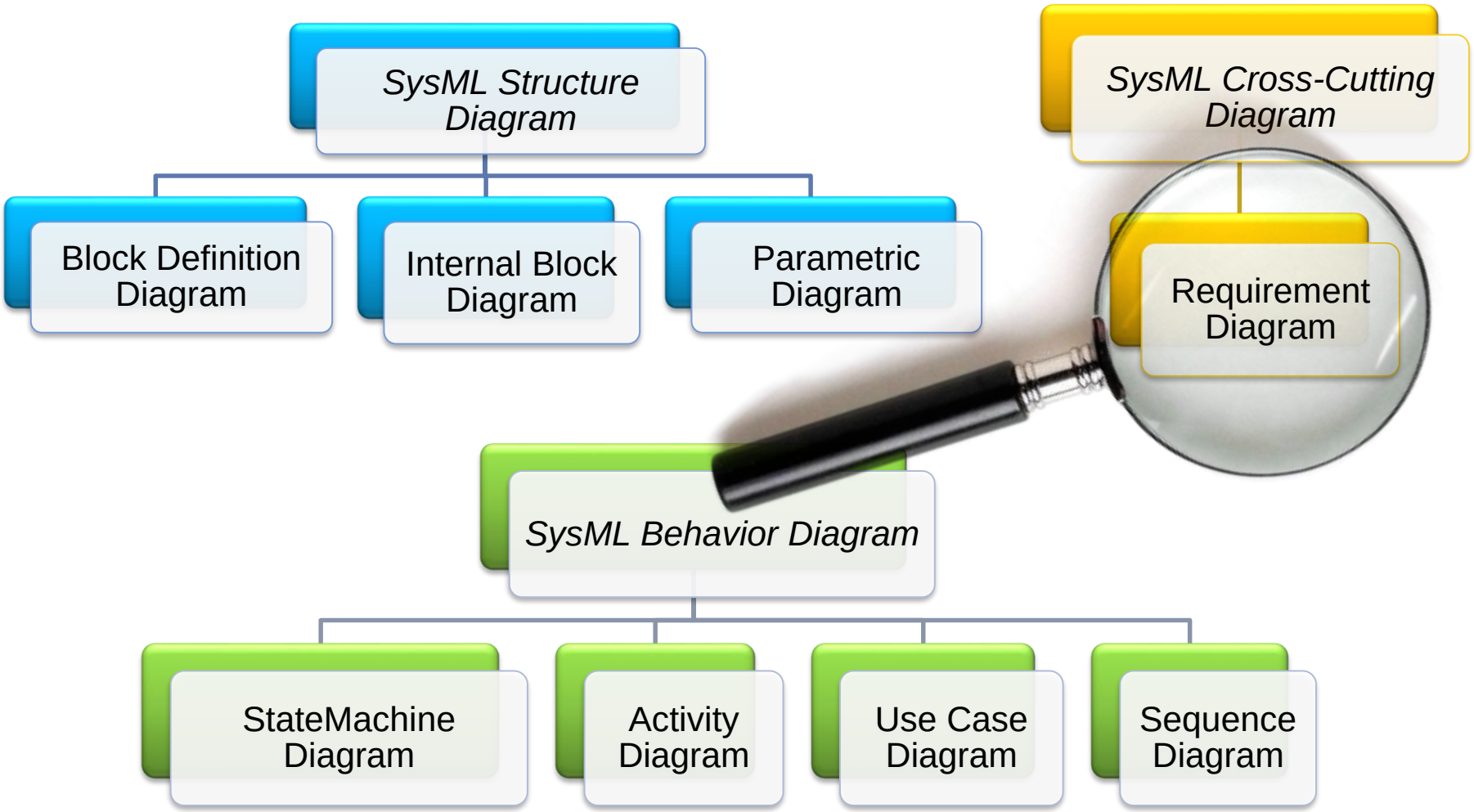
Software and system engineering department (DILS)

AGENDA

- Requirements modeling
- Use Cases modeling
- System level Interaction modeling

REQUIREMENTS

ABOUT THE CONCEPTS RELATED TO REQUIREMENTS MODELING



OUTLINES OF THE SYSML REQUIREMENTS

Scope

- SysML provides modeling constructs to represent text-based requirements and relate them to other modeling elements (Traceability).

A bridge between models and “traditional” req. management paradigms

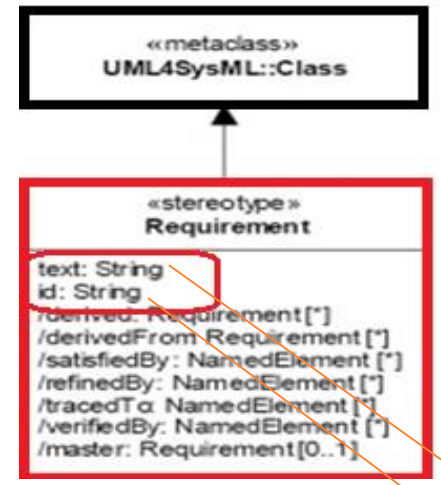
- A SysML requirement can be shown on all SysML diagrams to show its relationship to other modeling elements.

3 kinds of visual representations

- Graphical, tabular, or tree-based views.

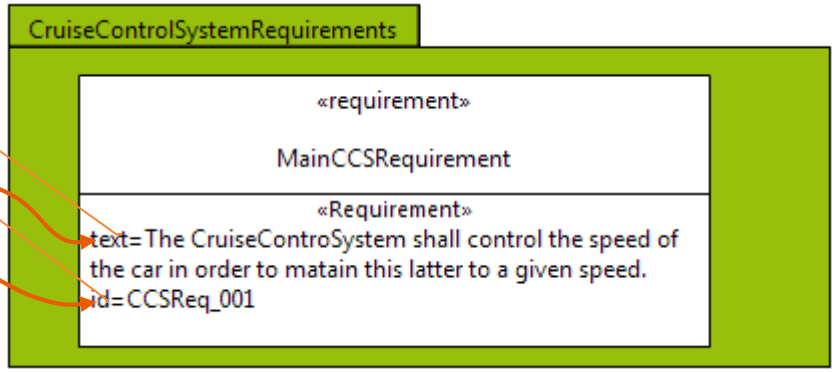
DETAILS OF THE «*REQUIREMENT*» CONCEPT

« Requirement » extends the meta-class Class (SysML profile)



Profile Level

User Level



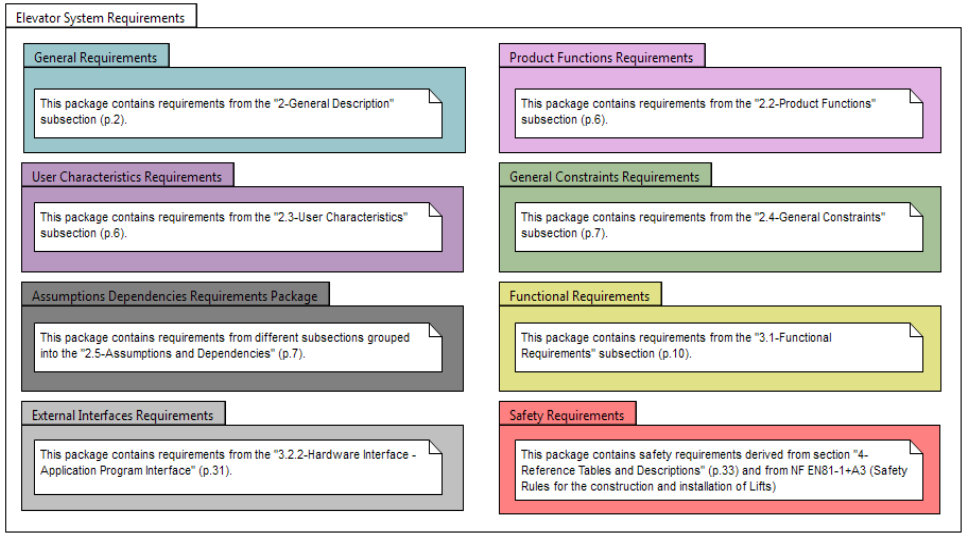
« Requirement » properties:

- A text requirement description
- A unique identifier (defined as a string)

The *Package* concept

- A Package is a namespace for its members, which are said to be owned or contained.
- A package can consist of packages (➔ hierarchical structuration).
- A package can import either other packages, or directly model elements owned in other packages.

Example



CONCEPTS FOR TRACEABILITY

« *Requirement traceability* includes several complementary notions. [From Gotel, 1994]
It includes:

- **Relations among requirements (behaviors refining requirements):**

  Answer the question: « *How a requirement can evolve?* »
  Decomposition, **refinement**, **copy, derive**

- **Relations between the requirements and the system**

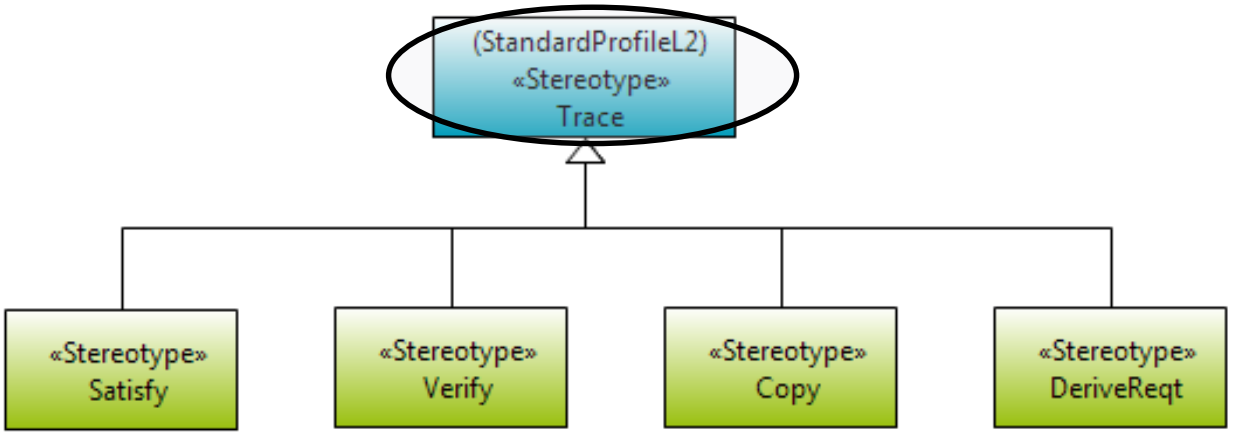
  Answer the question: « *How a system can be defined to ensure requirements?* »
  Relations such as: **satisfy**

- **Relations between requirements and V&V:**

  Answer the question: « *How a system meets the requirements?* »
  A system can be proved with respect to its requirements: test cases, code review, proofs,
  Relations such as **verify**

ABOUT REQUIREMENTS TRACEABILITY: THE « UML4SYSML::TRACE » CONCEPT

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

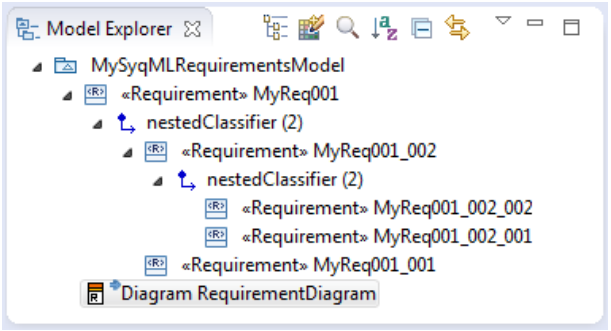


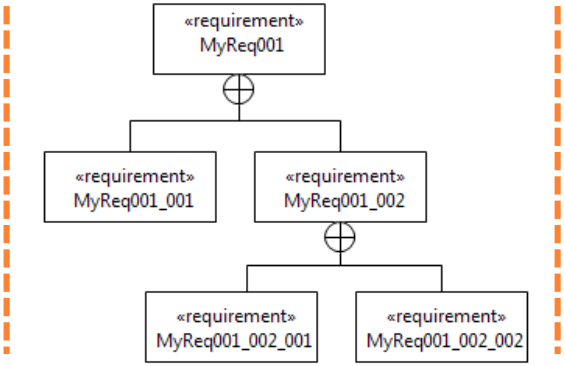
About the generic « UML::trace » relationship:
very generic (i.e., weak) semantics!

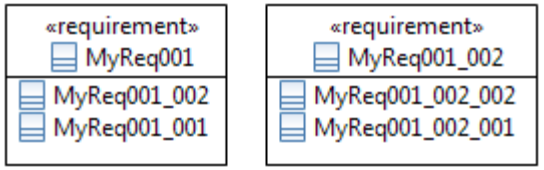
➔ not recommended to be used in conjunction with other SysML requirements relationships.

Composite requirements (i.e., hierarchical description)

- Specified using the UML namespace containment mechanism.







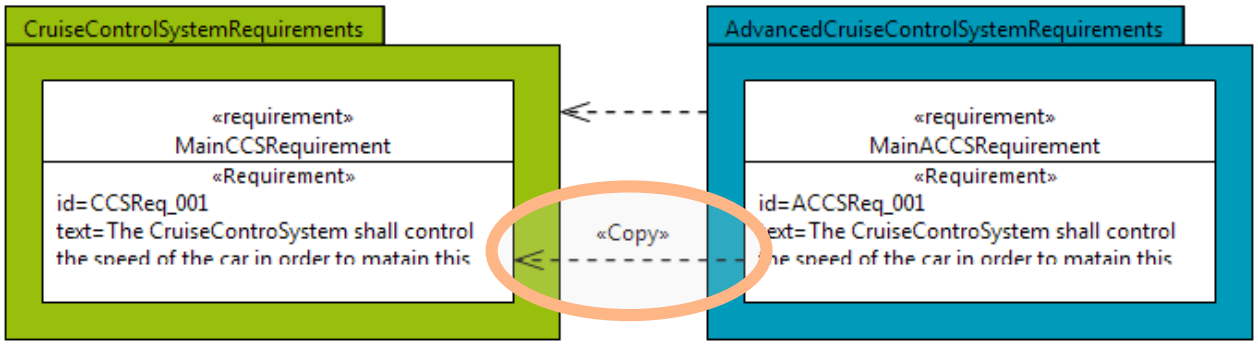
ABOUT REQUIREMENTS REUSE: THE « COPY » CONCEPT

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

- Definition: “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



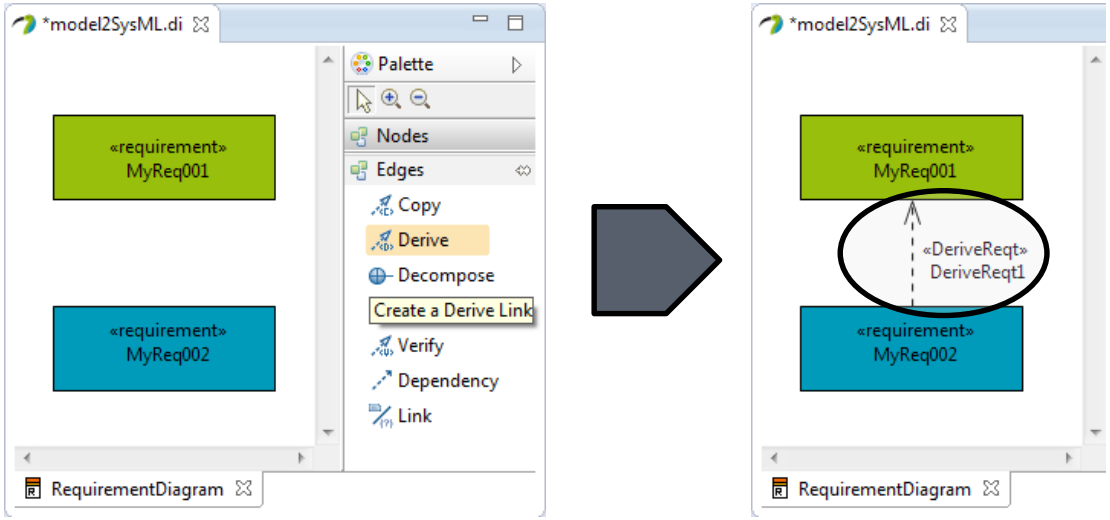
ABOUT REQUIREMENTS REUSE (SEQ.): THE « DERIVEREQ » CONCEPT

Usage

- Relates a derived requirement to its source requirement.

Example

- A vehicle acceleration requirement that is analyzed to derive requirements for engine power, vehicle weight…



Alternative tabular notation:

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

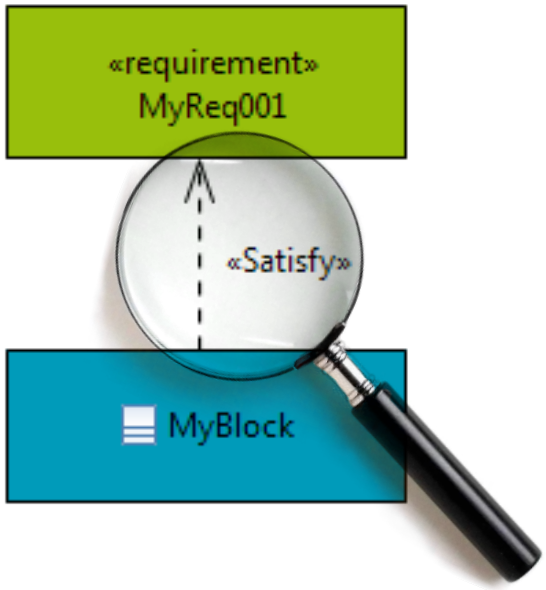
SATISFY AND REFINE

« Satisfy » Dependency

- Used for denoting a dependency between a design or implementation model element that satisfies one or more requirements.

Example

- A block of the design model satisfying a requirement.

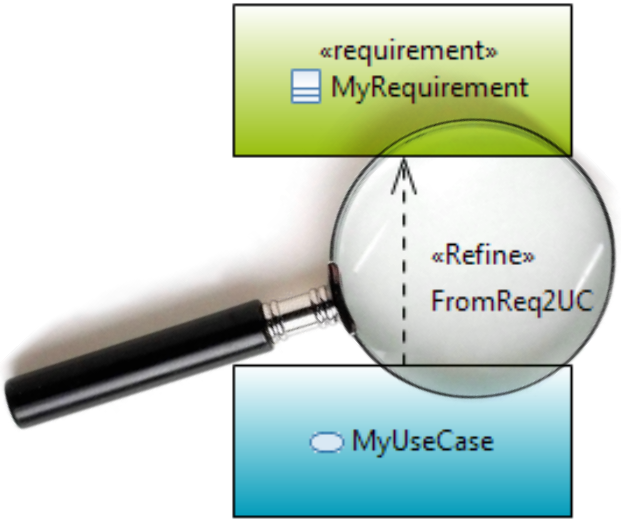


« Refine » Abstraction

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

Example

- Use case used to refine a requirement.



ABOUT REQUIREMENTS V&V: THE « VERIFY » AND « TESTCASE » CONCEPTS

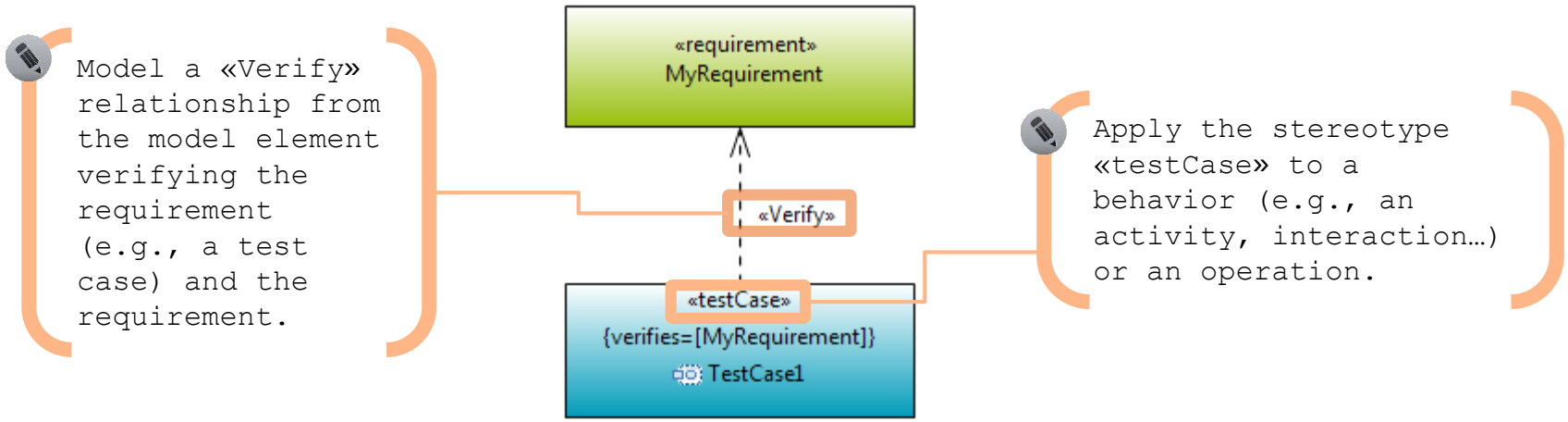
« Verify » Dependency

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

« TestCase »

- Used as a general mechanism to represent any of the standard verification methods 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/).

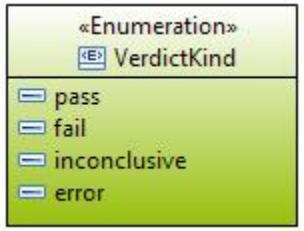
Modeling in Papyrus



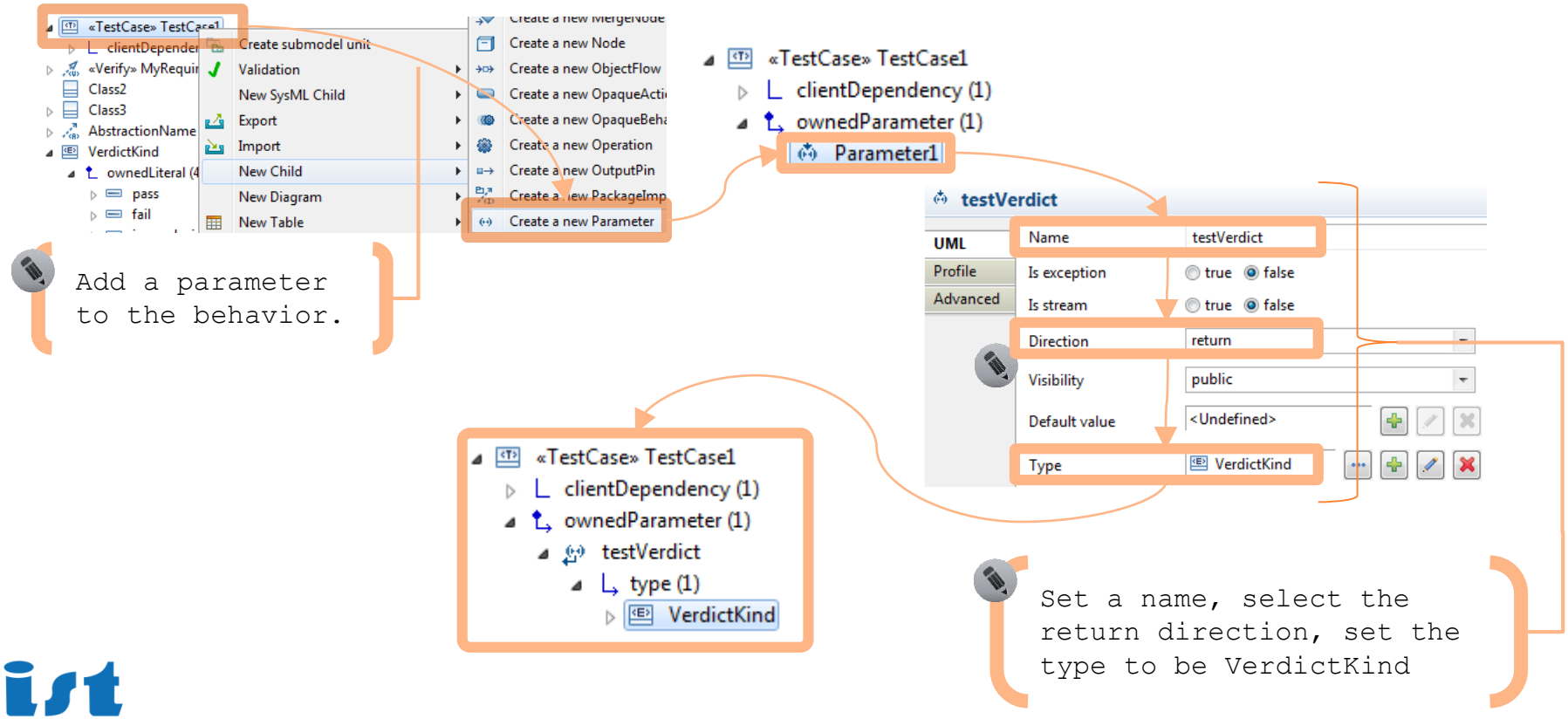
ABOUT REQUIREMENTS V&V (SEQ.): THE VERDICTKIND CONCEPT

VerdictKind (Enumeration)

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



Modeling in Papyrus



ABOUT « 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)

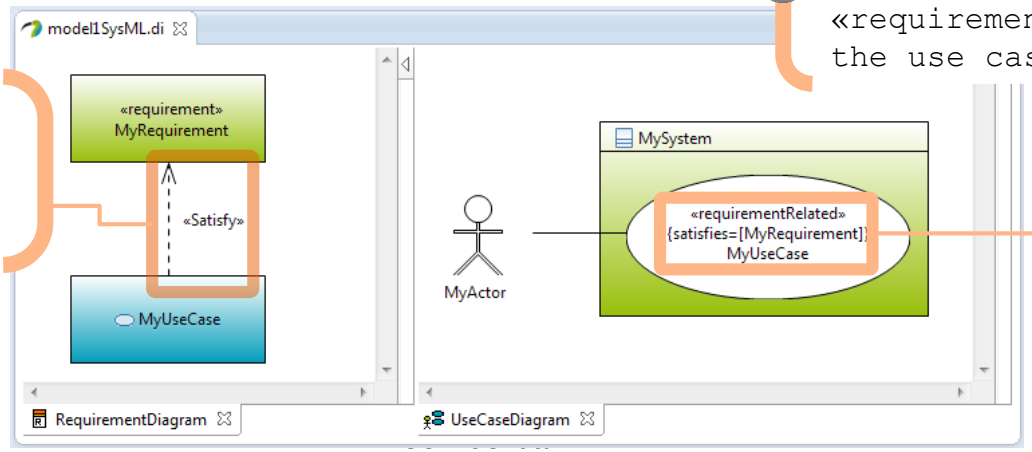
Derived properties

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

Modeling in Papyrus

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

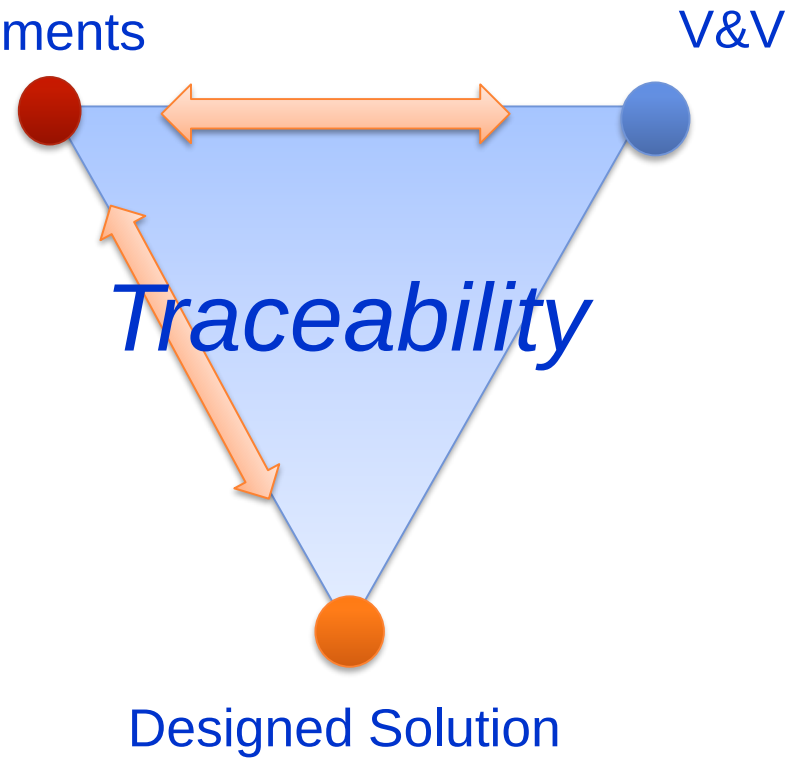
Apply the stereotype «requirementRelated» to the use case MyUseCase.



REQUIREMENT TRACEABILITY

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

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



REQUIREMENT DIAGRAM

Your turn:

- Capture the requirements using a SysML requirements diagram.
- Identify the various 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:

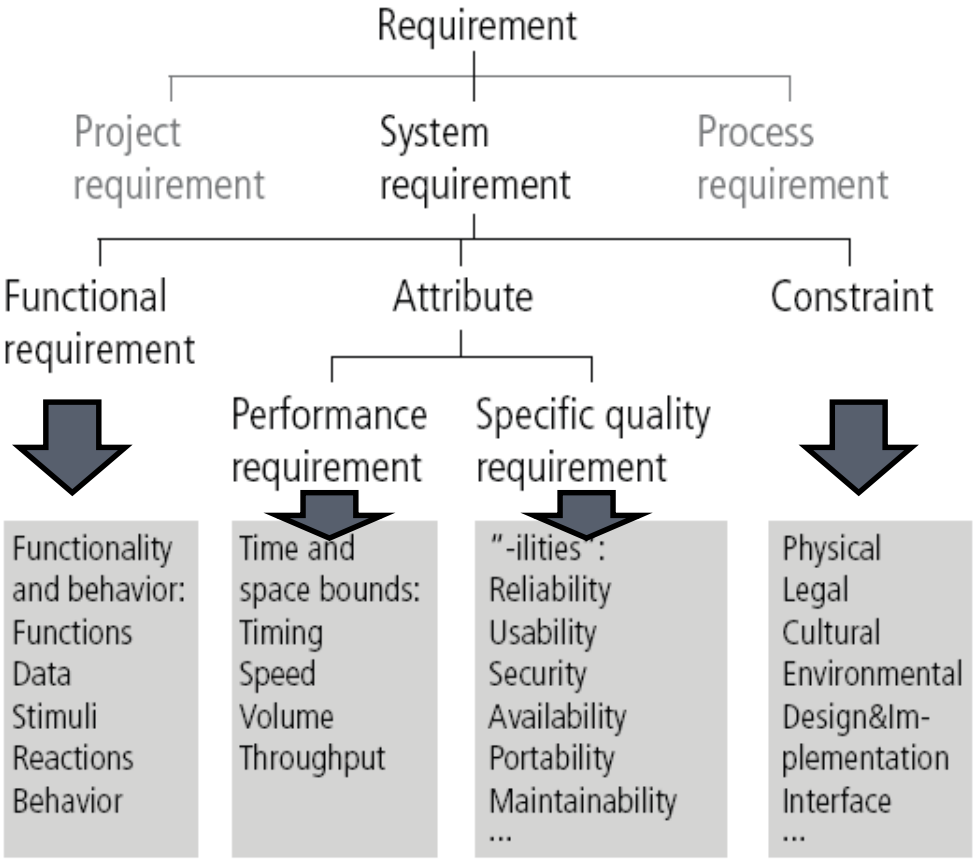
- Create a package “Requirements” at the root of the model and a requirement diagram in this package.
- Create a requirement table corresponding to these requirements.

REQUIREMENTS CLASSIFICATION

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.**

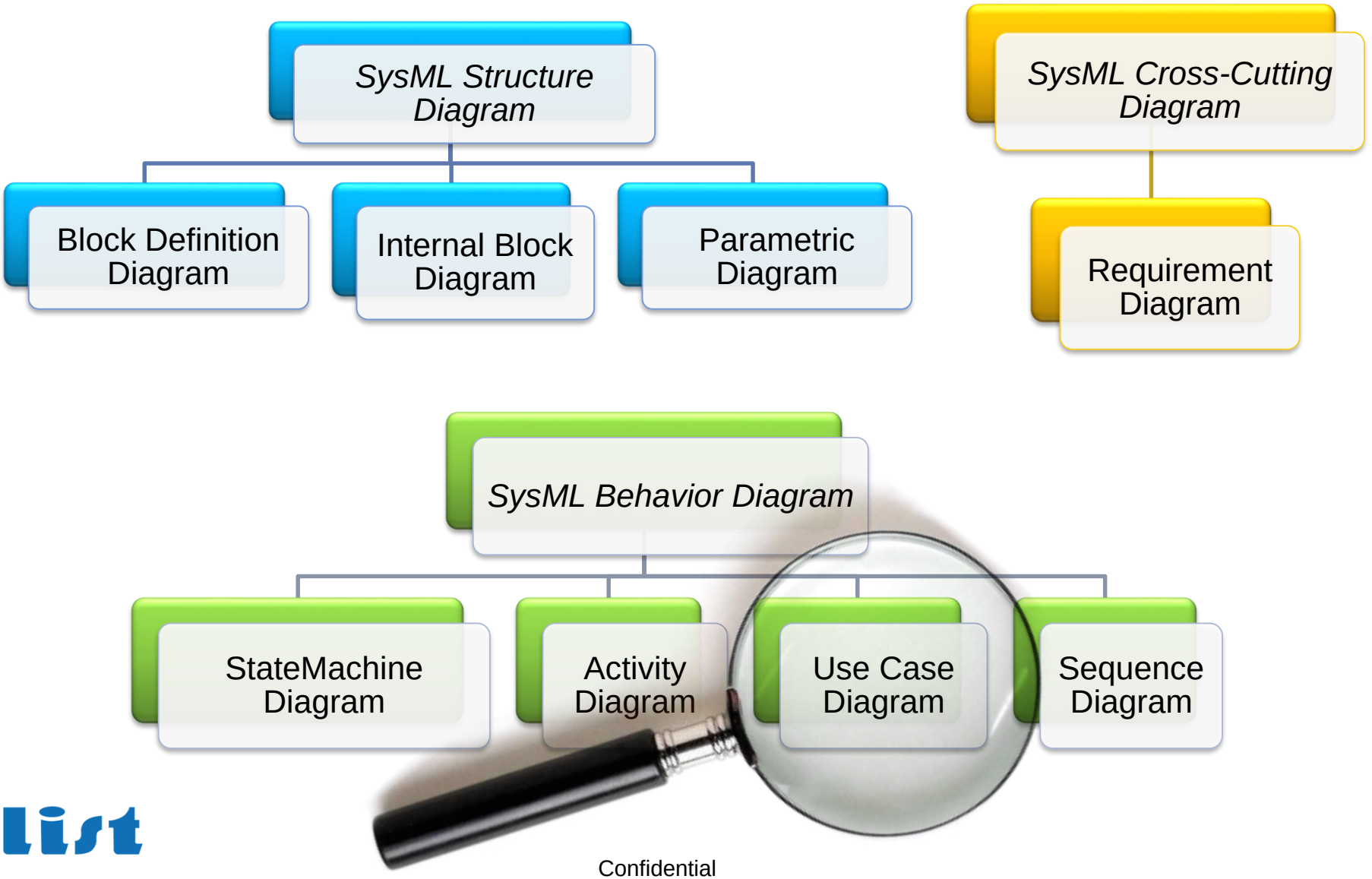
REQUIREMENTS CLASSIFICATION



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

USE CASES MODELING

ABOUT THE CONCEPTS RELATED TO USE CASE MODELING



OUTLINES OF THE USE CASE DIAGRAM

Scope

- Mainly dedicated to describe system high-level requirements in a functional manner and considering the system as a black box.

Usages

- Identification of the main system offered functionalities
- Interaction points between the system and its environment
- Clear systems boundaries identification

Consists of:

- One subject ➔ the system itself
- Actors ➔ the elements of the environment interacting with the system
- Use cases ➔ the functions provided by the system to its users

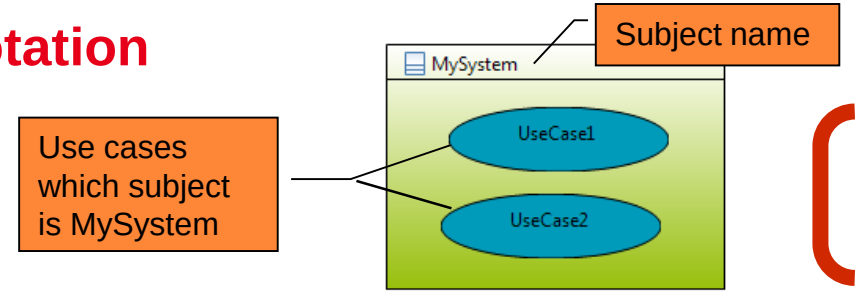
A UseCase is a specification of behavior, which is often described by one or many Interactions.

ABOUT THE *SUBJECT* OF A USE CASE DIAGRAM

Usage

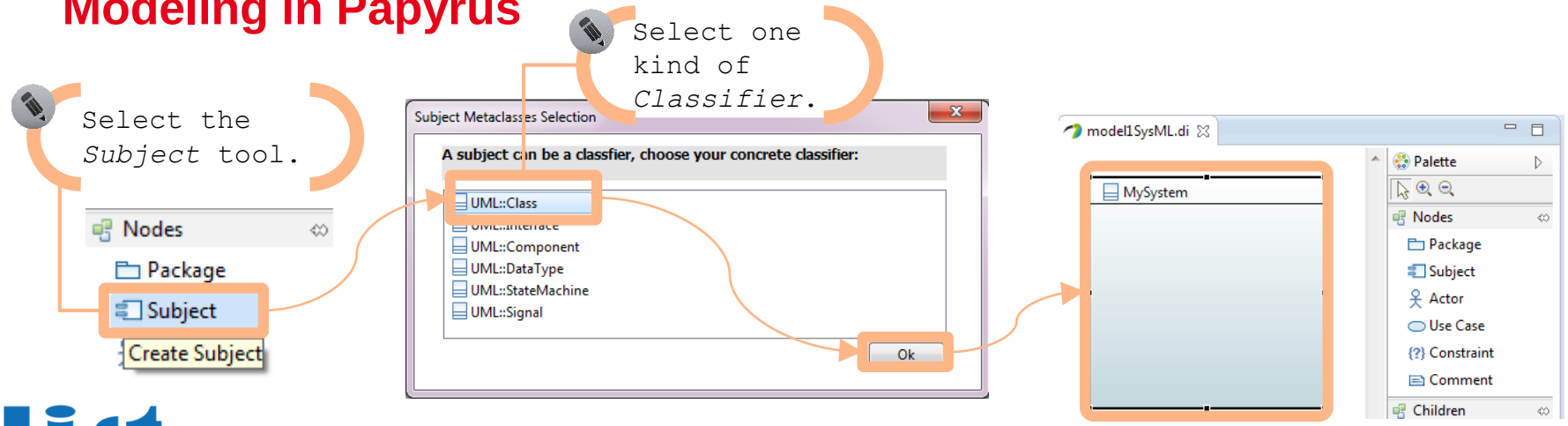
- Used to denote the *system (sub system) under study*
- *Any UML4SysML::Classifier can be a subject*
  - ► E.g., Class, Interface, and Component.

Notation



Use cases shown in the rectangle denoting the subject <u>might not be owned</u> by the subject itself!

Modeling in Papyrus

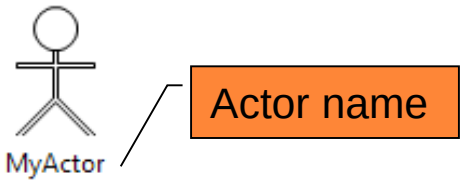


DETAILS ON THE *UML4SYSML::ACTOR* CONCEPT

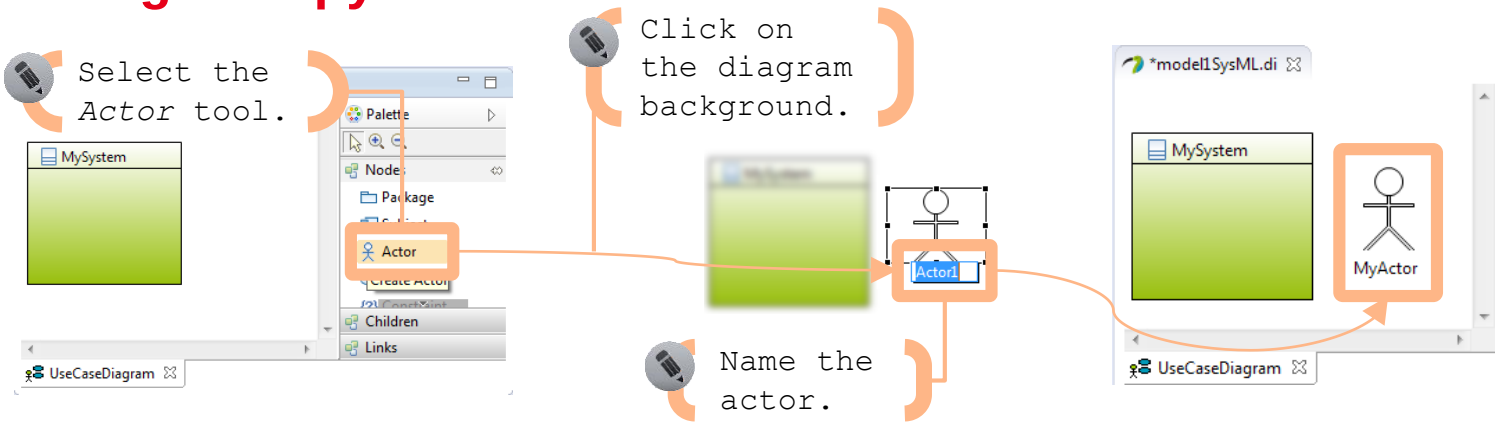
Usage

- Used to model categories of role played by entities interacting with a subject within the context of its related use cases.
- May be for example human users, external hardware devices, or any other systems.

Notation



Modeling in Papyrus

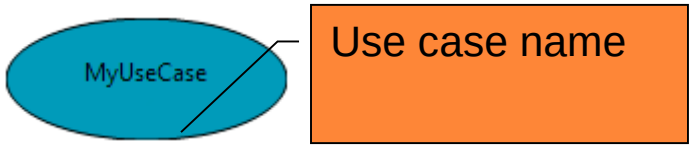


DETAILS ON THE *UML4SYSML::USECASE* CONCEPT

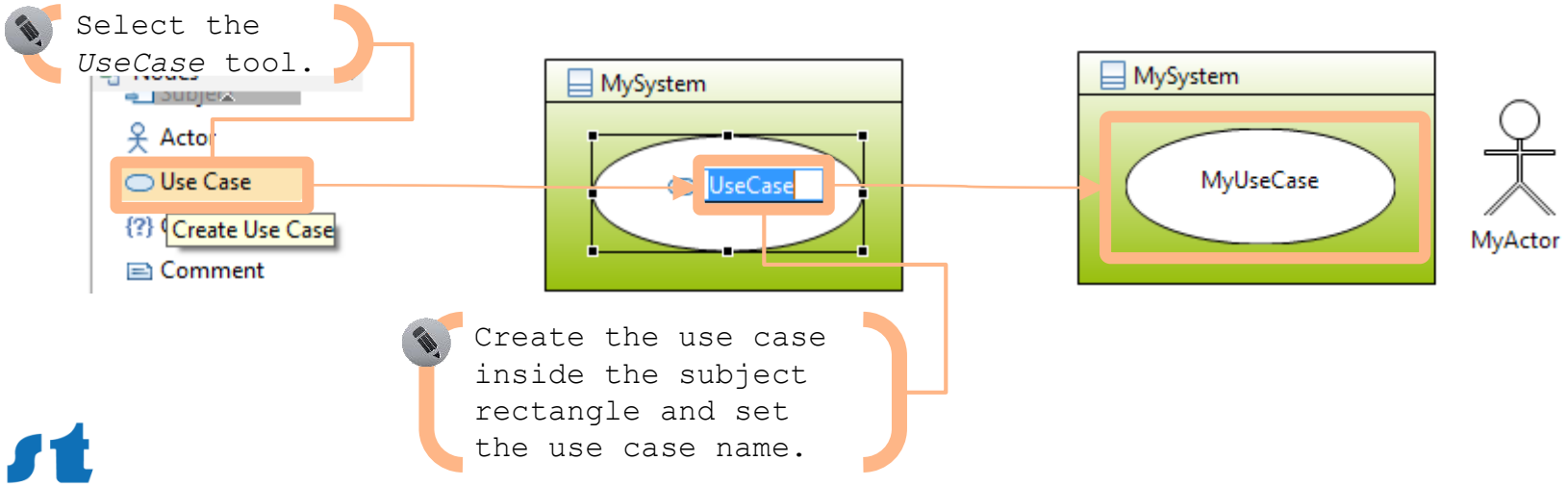
Usage

- Denote a set of behaviors performed by a subject, which yields an observable result that is of value for one or more of its Actors.
- Black-box view of the subject => only externally observable behavior are denoted (i.e., no internal details).

Notation



Modeling in Papyrus



ABOUT USECASE AND ACTORS ASSOCIATIONS

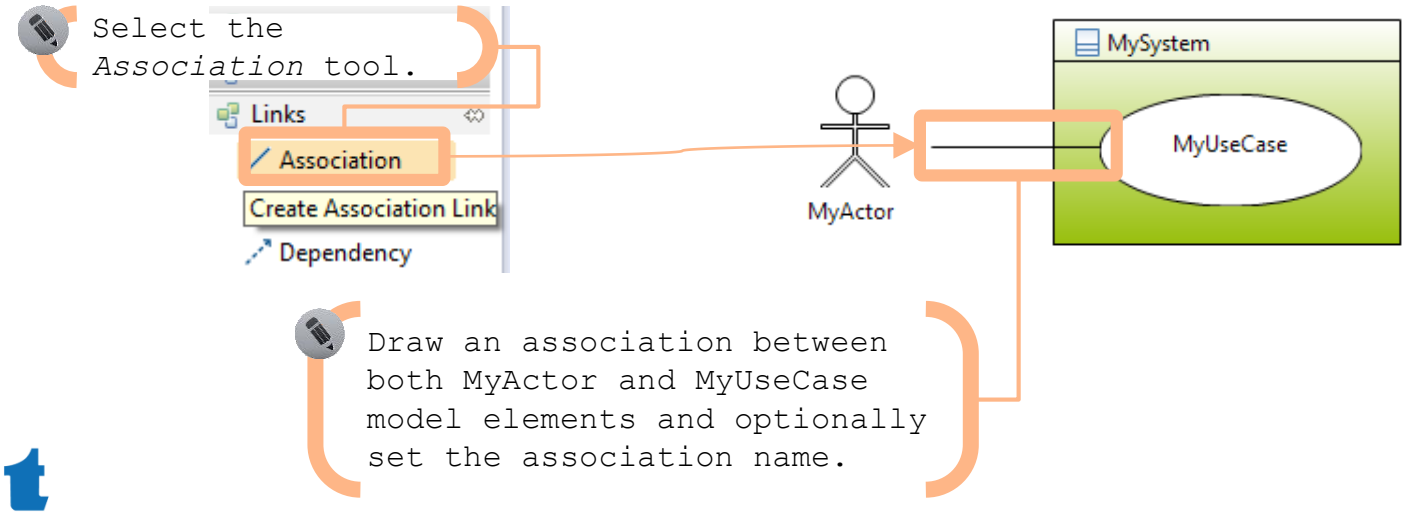
Usage

- Used to denote the existence of interactions between a use case and the system environment (i.e., the actors)

Specificities

- When a use case has an association to an actor with a multiplicity that is greater than one at the actor end, it means that more than one actor instance is involved in initiating the use case.

Modeling in Papyrus

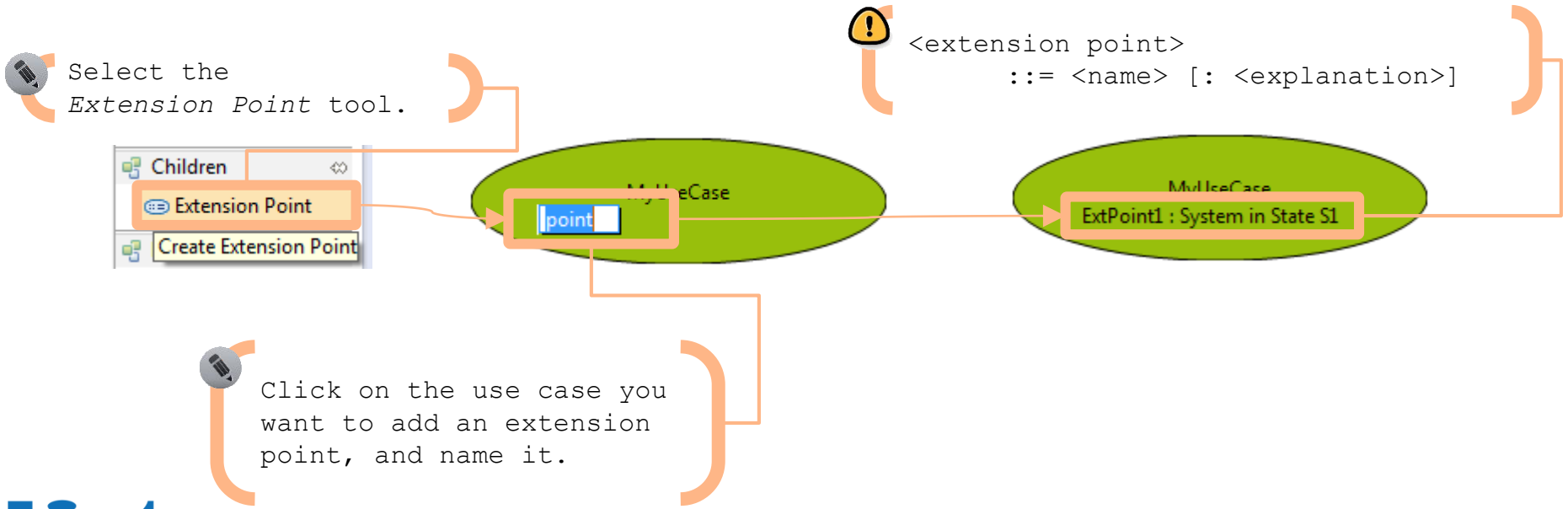


ABOUT EXTENSION POINTS OF USE CASES

Usage

- Define a location within a use case where behavior of other use cases may be inserted as extensions (see next slides).
- Extension point name are unique within a same use case

Modeling in Papyrus



THE EXTEND RELATIONSHIP

Usage

- Relationships used to model how a use case may be extended by one or many other use cases.
- Two properties:

1) condition : Constraint [0..1]

2) extensionLocation : ExtensionPoint [1..*] {ordered}

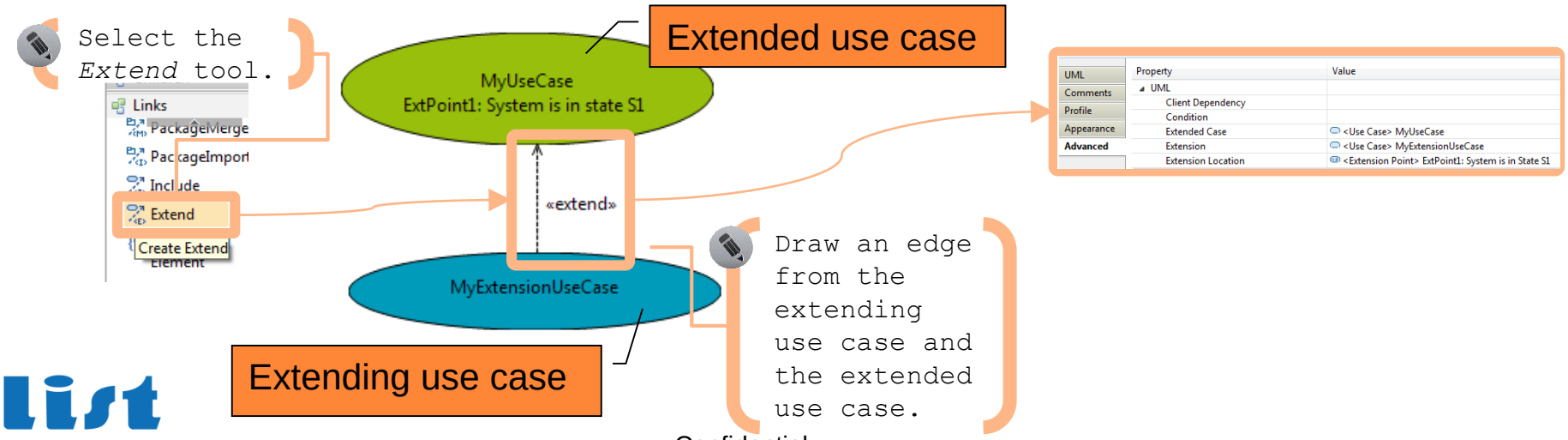
Extended use case

- Defined independently of the extending use case

Extending use case

- Not necessarily meaningful per se
- May extend several use cases
- May be also extended

Modeling in Papyrus



THE INCLUDE RELATIONSHIP

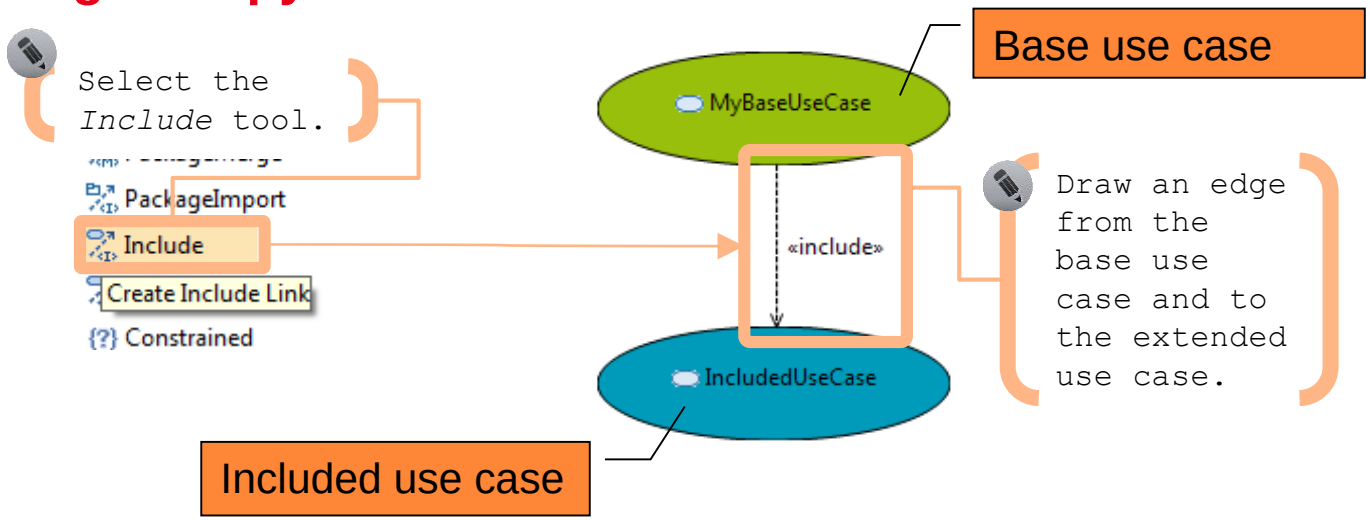
Usage

- Specify that the behavior of the included use case is part of the including use case (also called “base use case”).
- Used to enable reuse between use case.

Main feature

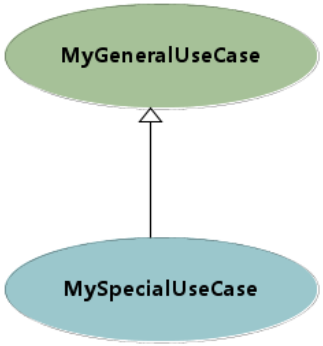
- Contrary to extend relationship:
  - » Insertion of included use case is not optional
  - » Base use case needs included use cases to be meaningful
- A use case may be included in several base use cases

Modeling in Papyrus



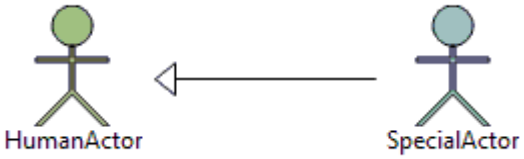
Uses cases

- A use case is a classifier
- The specialized use case inherits the features of the general use case (extend and include relationships)



Actors

- An actor is a classifier
- The specialized actor inherits the features of the general actor (attributes, behaviors)



USECASE ARE ALSO BEHAVIOREDCLASSIFIER!

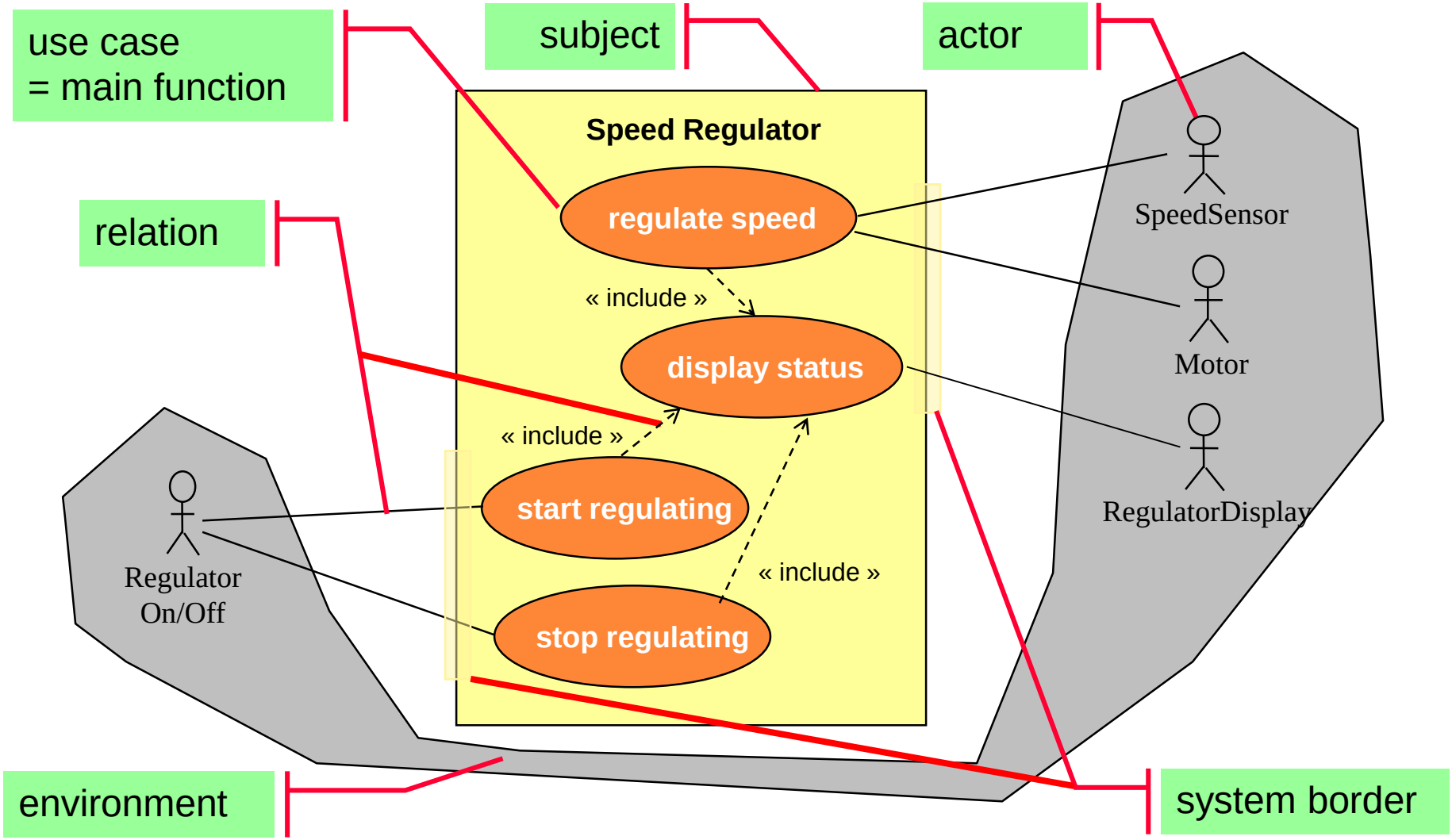
BehavioredClassifier ➔ May own one or many behaviors

- Enable to model variations of the basic behavior of a use case, including as for example exceptional behavior and error handling.

Use case behaviors may be:

- Interactions, activities, or state machines,
- Or pre-conditions and post-conditions,
- Or natural language description,
- Or indirectly through a Collaboration that uses the use case and its actors as the classifiers that type its parts.

SUMMARY ON USE CASE DIAGRAMS



SUMMARY ON USE CASE DIAGRAMS

When to use a use case diagrams?

- For describing the requirements/expected functions of a system in the analysis, design, implementation and documentation stages.

When constructing use case diagrams, think about:

- What tasks the subject must perform?
- What resources the subject requires to perform its tasks?
- What actor(s) triggers the tasks?

A use case model is a view of a system that emphasizes its functionalities and boundaries without revealing details of the system inside:
➔ System abstracted under a “black box” point if view.

USE CASE DIAGRAM

Your turn:

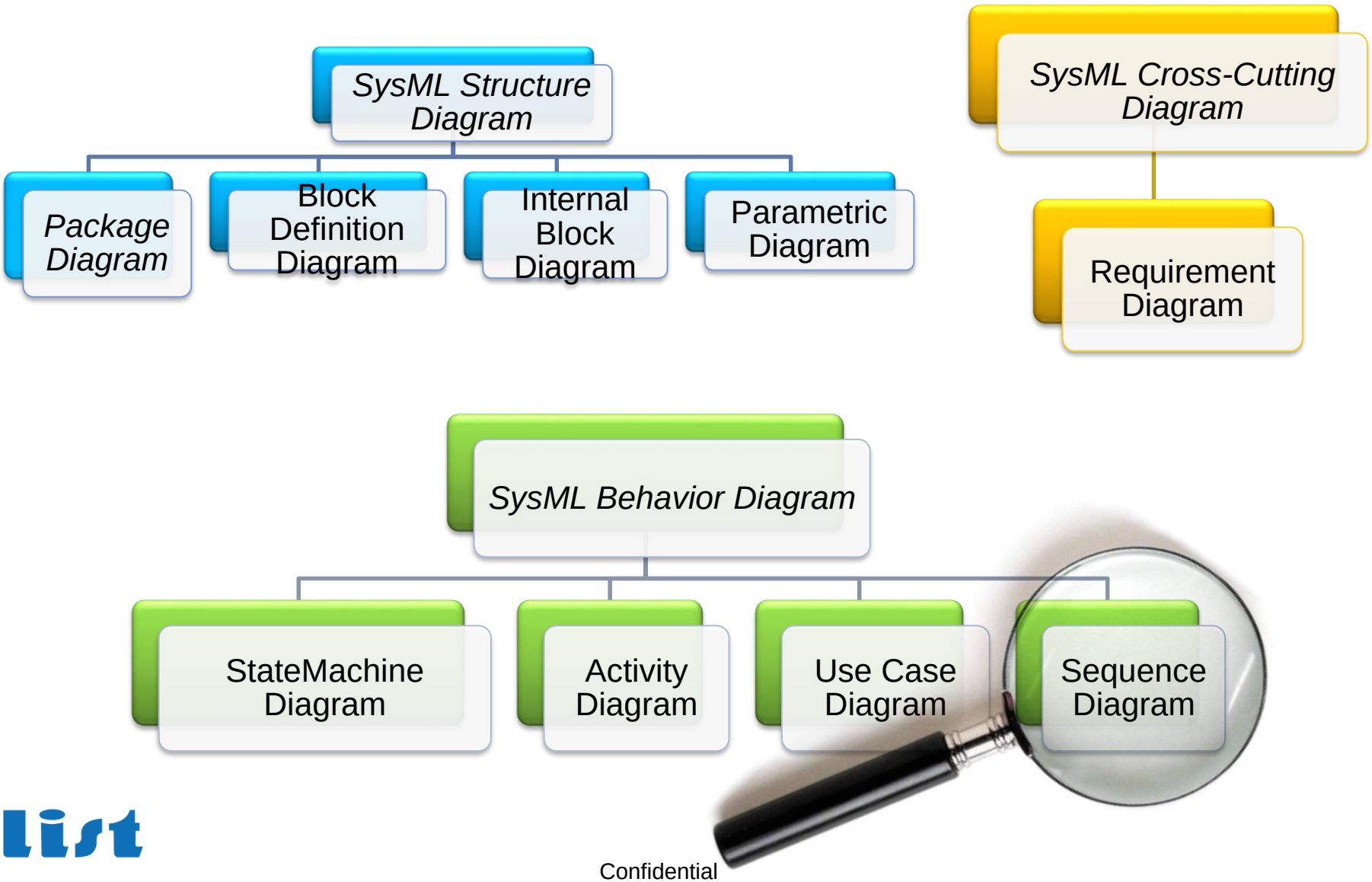
- Identify the actors and the use cases.
- Relate the actors with the use cases they participate in.
- Mind the “includes” and “extends” semantic relationship that may exist between the use cases.

Do it in Papyrus:

- Create a package “UseCases” at the root of the model and a Use case diagram in this package

SYSTEM/ENVIRONMENT INTERACTIONS MODELING

SEQUENCE DIAGRAM



INTERACTIONS – OBJECTIVE AND USAGES

Describe the messages sequences exchanged between structural elements composing a system

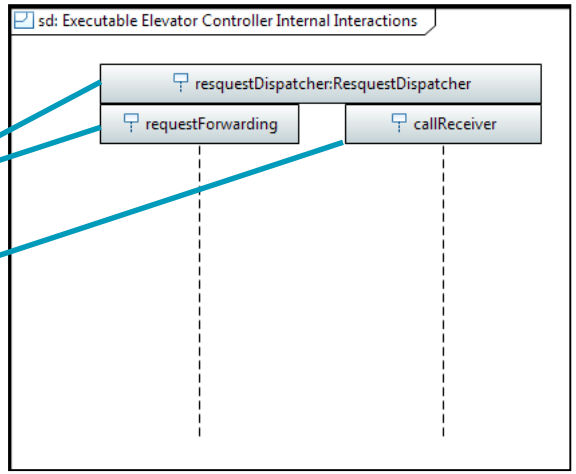
Interactions are used

- during [analysis], to understand
  - Interaction between structural elements (parts) composing the system
  - Interactions between the system and its environment (relation with use cases)
- during [design], to refine
  - Interactions identified between structural elements (parts / ports) of the system
- during [testing], to compare
  - Sequence of behavior initially specified with traces generated by simulation

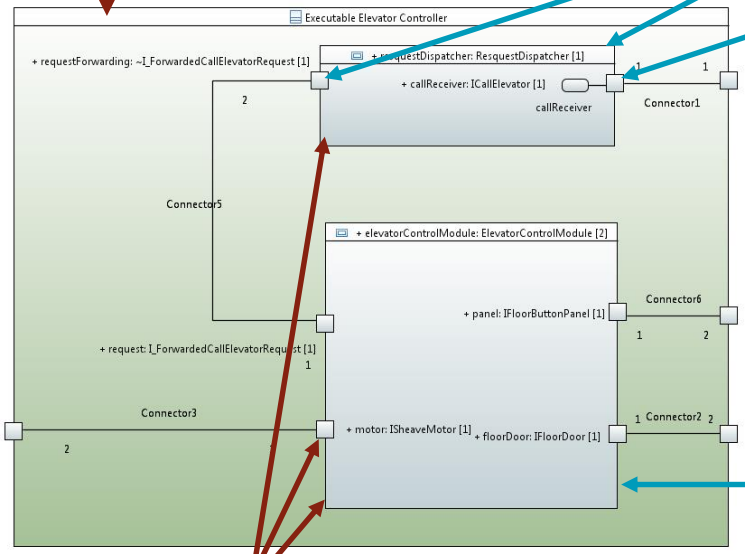
# DENOTES INTERACTION PARTICIPANT
LIFELINES

A lifeline represents an individual participant in an interaction

Represents



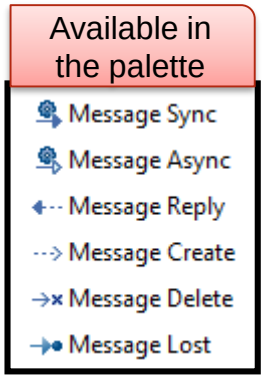
The system

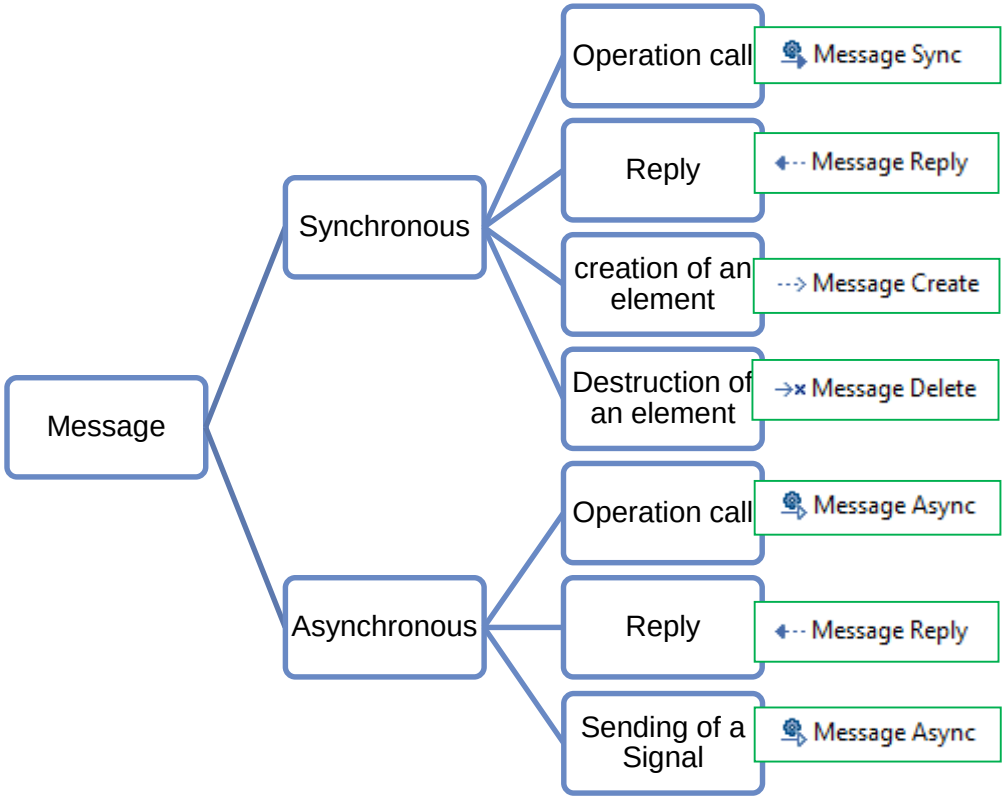


Participants

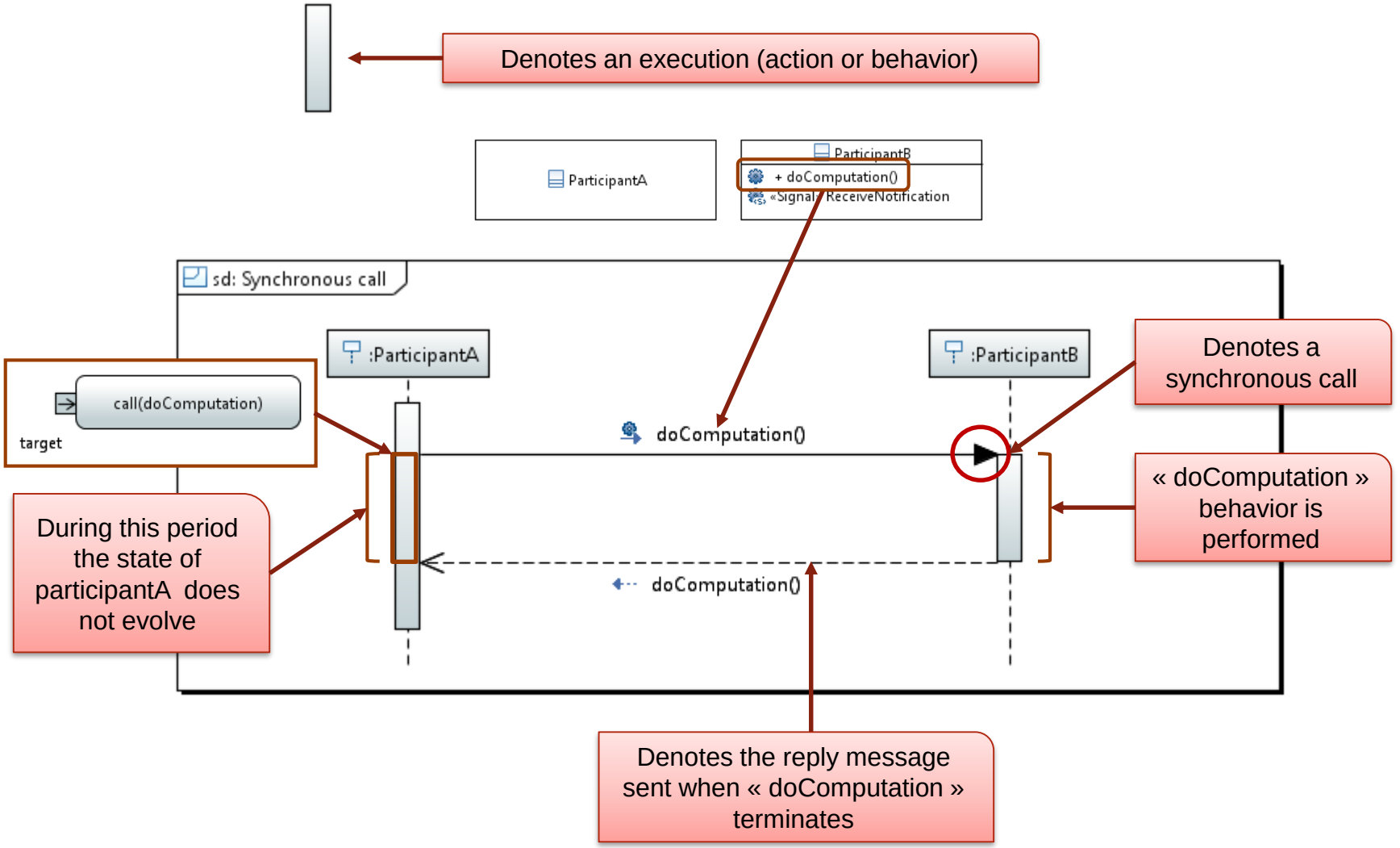
DENOTE COMMUNICATIONS (1)
MESSAGES KIND

A communication between two participants denoted as lifelines

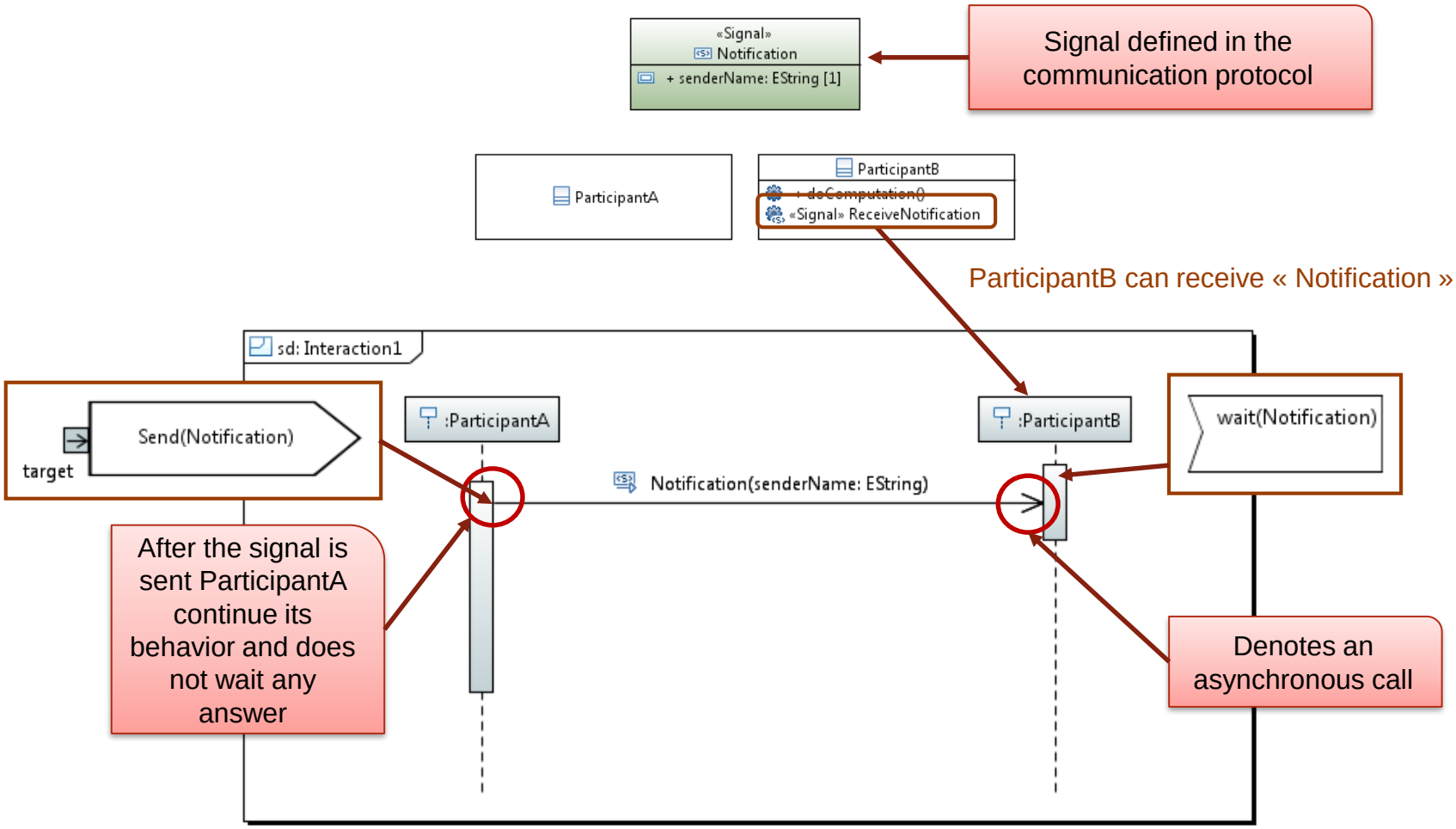




DENOTE COMMUNICATIONS (2)
SYNCHRONOUS CALL AND REPLY

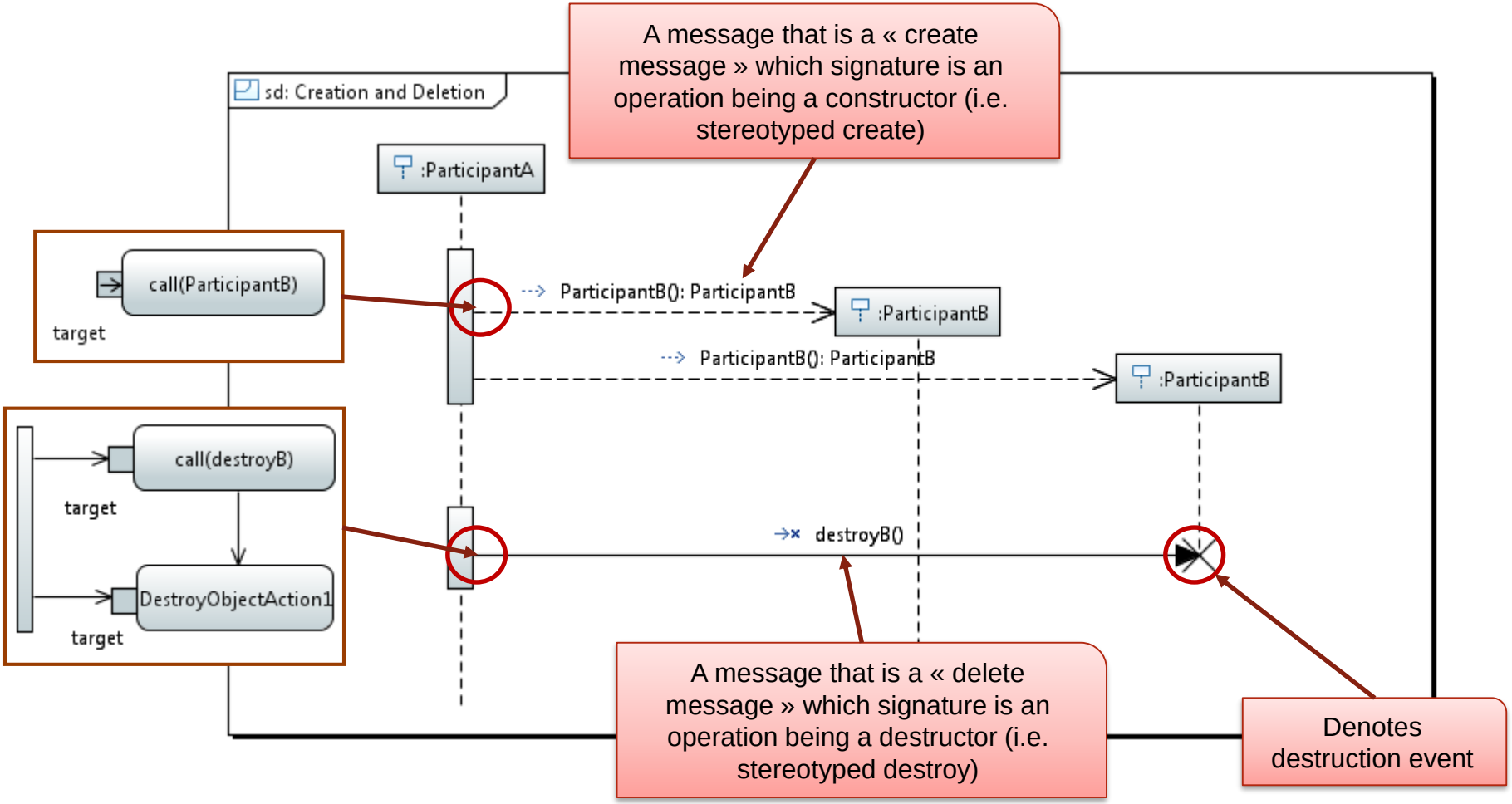


DENOTE COMMUNICATIONS (3)
ASYNCHRONOUS CALL



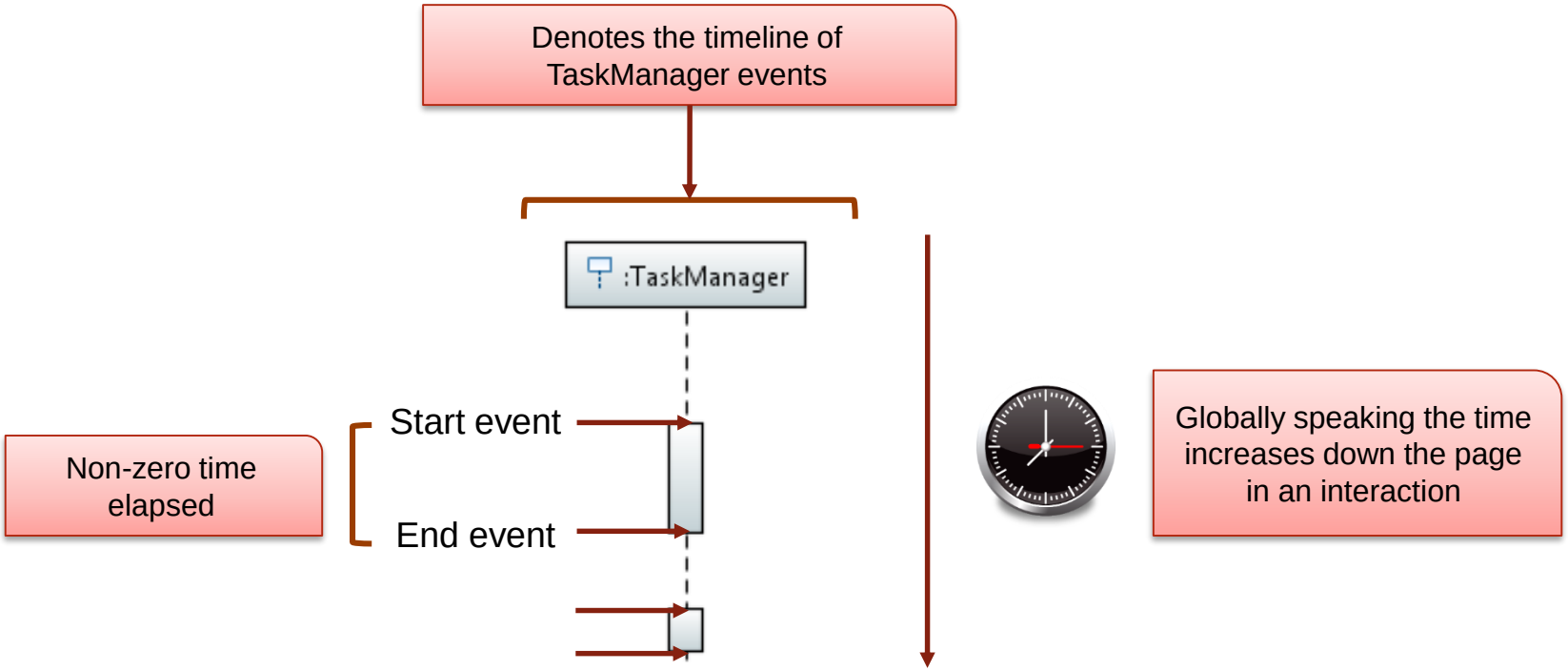
NOTE: It is fine to call an operation asynchronously

DENOTE COMMUNICATIONS (3)
CREATION AND DELETION



TIME IN INTERACTIONS (1)

HOW TIME IS EXPRESSED THROUGH LIFELINES



Observations

- **Timing observation**

  A time observation is a reference to a time instant during an execution.
- **Duration observation**

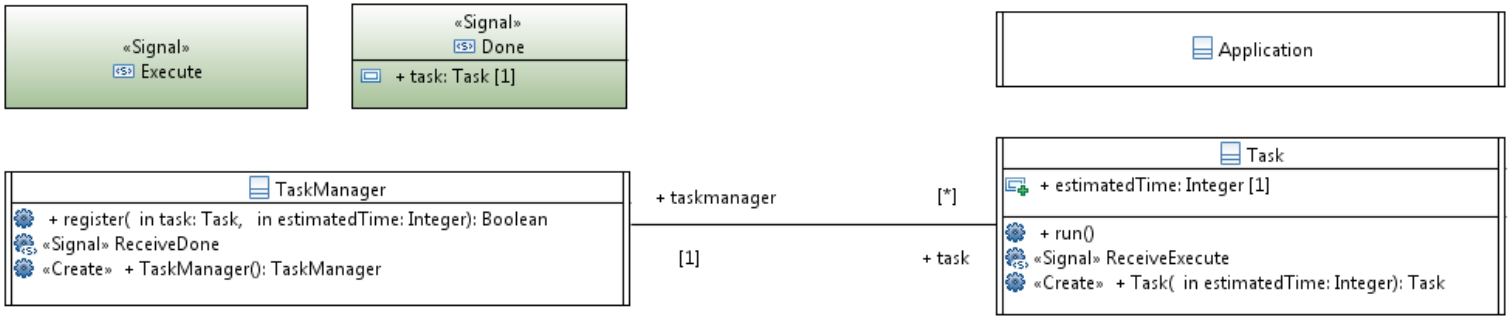
  An observation is a reference to a duration during an execution. It is denoted as an interval.

Constraints (Traces that violate the constraints are negative traces)

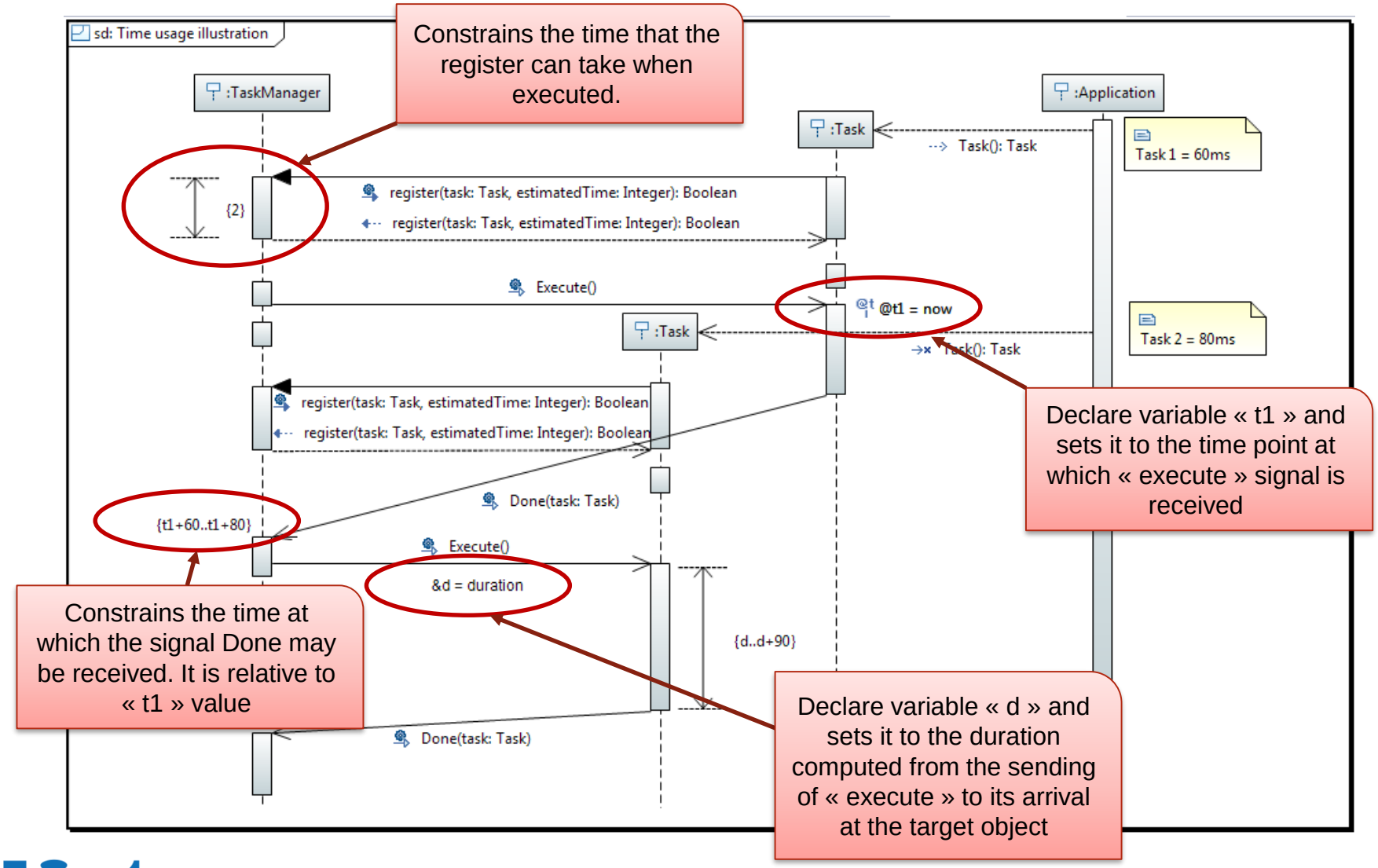
- **Timing constraint**

  Constrains the time point to which a specific event can occur. The constraint is expressed as an interval.
- **Duration constraint**

  Constrains the duration that separates two events. The constraint is expressed as an interval.



TIME IN INTERACTIONS

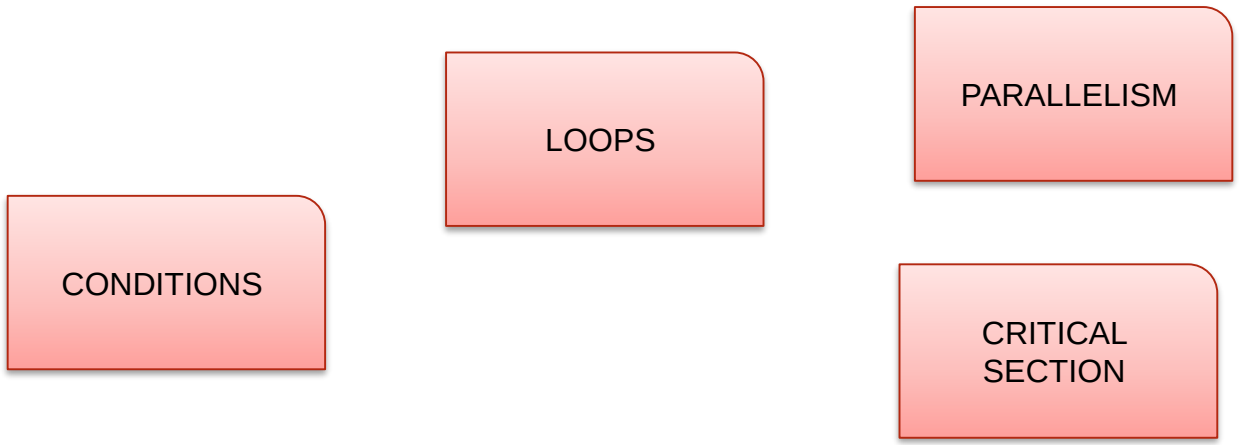


WHAT IS REQUIRED TO EXPRESS MORE COMPLEX TRACES ?

Until now we know that:

- Lifelines references instances of system structural elements
- This elements can exchanges messages that can be synchronous / asynchronous
- Time elapse during an interaction and can be modeled/manipulated explicitly
- Events occur according in predictable way (total order)

What is needed to express more realistic traces ?



TO DO WE USE COMBINED FRAGMENTS

A combined fragment is used to group sets of messages together to show conditional flow in a sequence diagram

A combined fragment has

- An interaction operator
- A set of interaction operands

12 interaction operators for combined fragments

- Alt – alternatives with conditions for each operand
- Opt – optional with condition (unique operand)
- Par – parallel merge between the behavior of the operands
- Loop – loop with condition (unique operand)

COMBINED FRAGMENTS - NOTATION

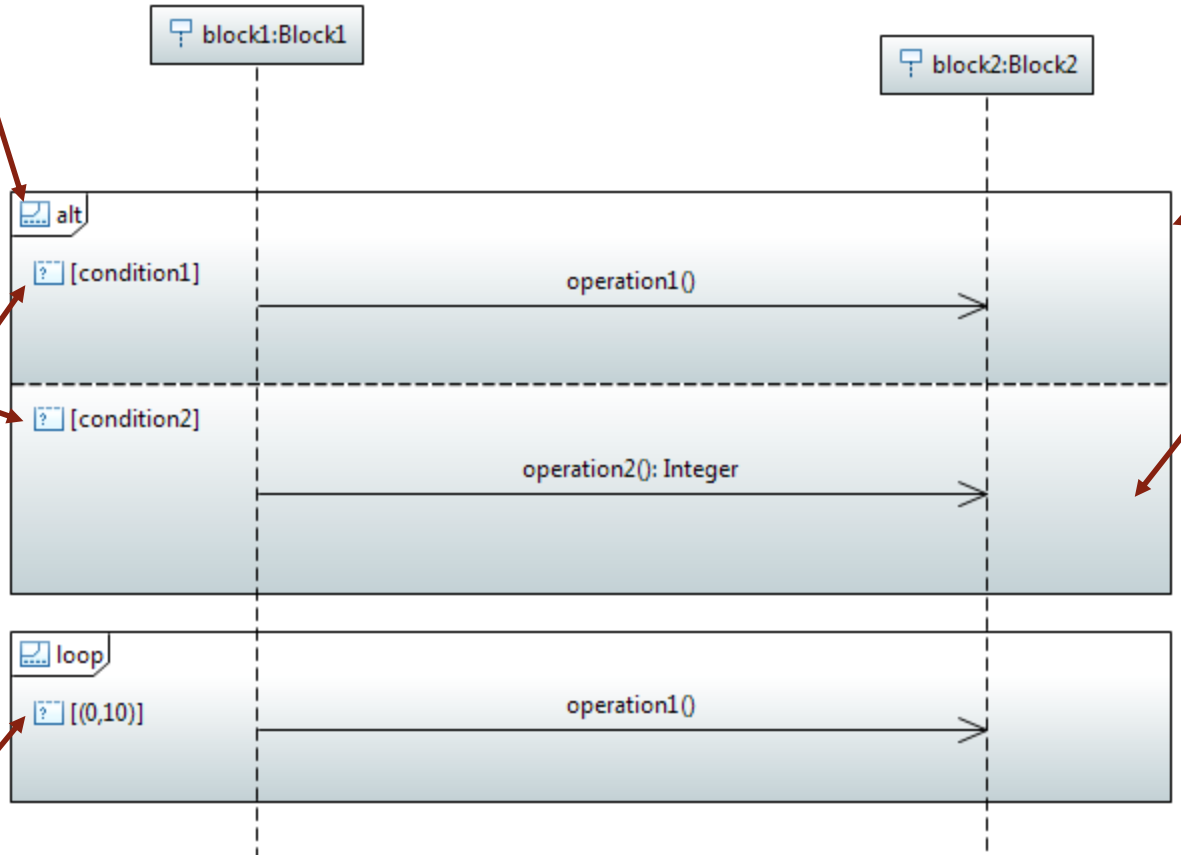
interactionOperator:
ALT, LOOP, OPT, PAR, CRITICAL, NEG, ASSERT, STRICT, SEQ, IGNORE, CONSIDER…

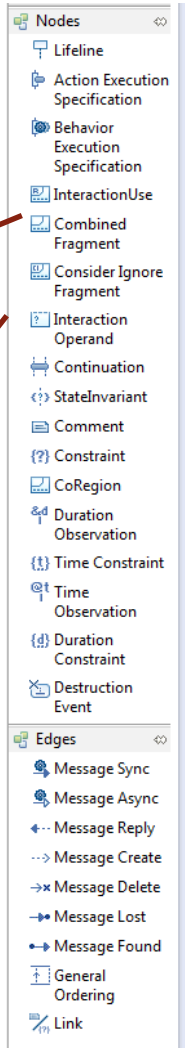
Guard:

Condition to enter the operands

Guard:

Condition to loop on this fragment





SEQUENCE DIAGRAM

Your turn:

- **Describe the interaction that exists between the Maintenance Operator and the system for the Steering Calibration use case.**
- **Describe the interaction that exists between the Rover Operator and the system for the Steering Control use case.**

Do it in Papyrus:

- **Contextualize the system thanks to a BDD diagram in a "Contextualization" Package at the root of the model**
  - Reify the actors with Blocks
  - Contextualize the system in it environment: relate the system and the reified actors to the context.
- **Create a package "EnvironmentSystemInteractions" at the root of the model and Sequence Diagrams in this package**

www.eclipse.org/papyrus

Commissariat à l'énergie atomique et aux énergies alternatives
Institut Carnot CEA LIST
Centre de Saclay | 91191 Gif-sur-Yvette Cedex
T. +33 (0)169 077 093 | M. +33 (0)688 200 047

Direction DRT
Département DILS
Laboratoire LISE