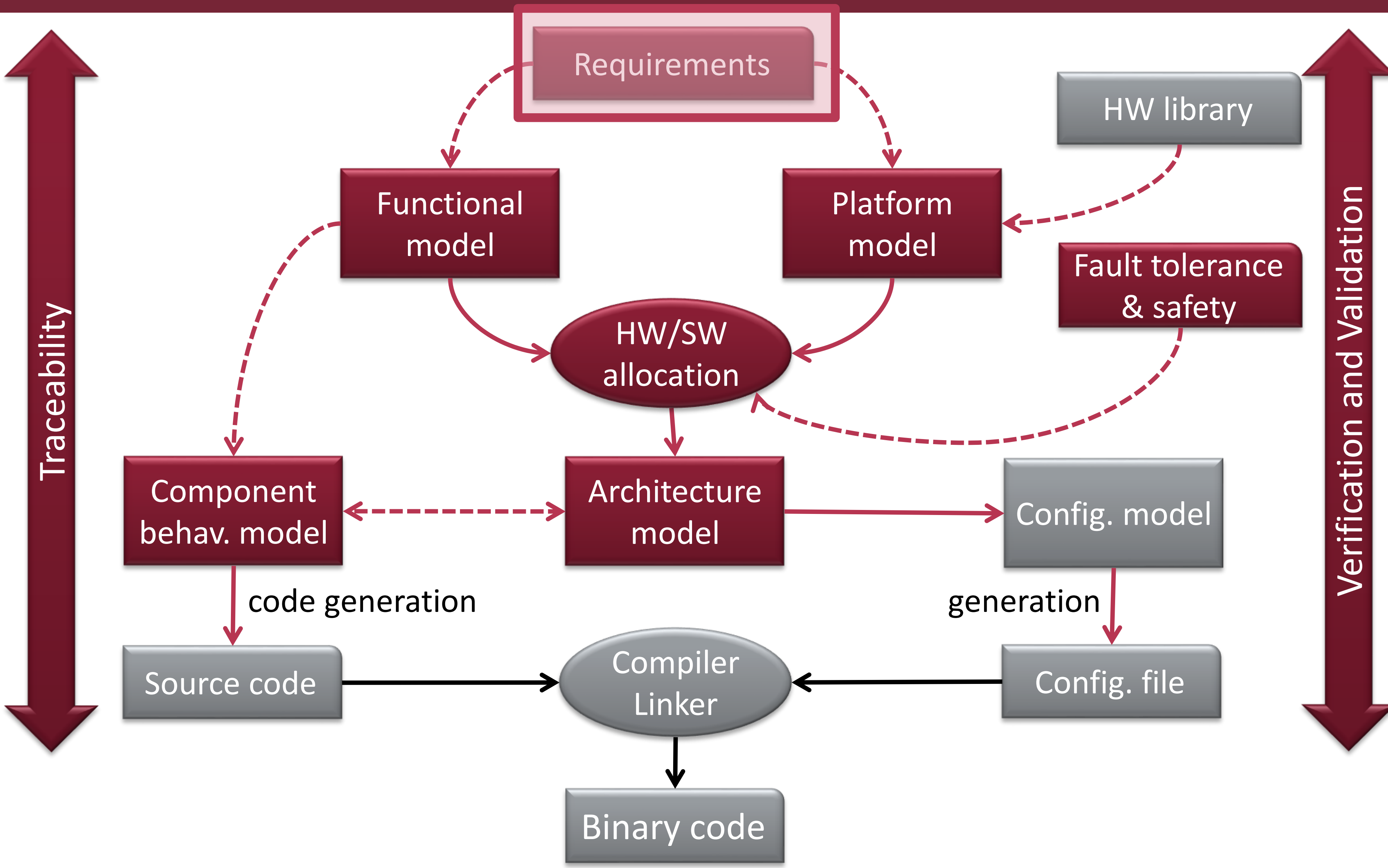
Modeling Textual Requirements

Systems Engineering BSc Course

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Platform-based systems design



Learning Objectives

Requirements

- Understand the role and major challenges of requirements engineering in systems design
- Write precise textual requirements
- Understand requirements written by others
- Capture requirements using the SysML language
- Understand the goal of traceability
- Identify relations between requirements

Use cases (System Functions)

- Understand the concepts of actors and use cases
- Capture system functions in use case diagrams
- Identify relations between actors and use cases

Why are Requirements Needed?

Project Kick-off

- **Business Case**: Why the project is needed?
  - Revenue? Units to be Sold?
- **Constraints** and Rationale:
  - Time: deadlines, iteration cycles
  - Budget & Costs: HW, unit cost, development
- **Glossary / Terms:**
  - Identify existing documents, standards
  - Identify experts: who knows what?
  - Prepare inventory
- **Teams**
- **Context (see: use case diagrams)**
- **Requirements**

Teams

- *Customer team*
  - Product manager
  - Systems engineers
  - Business analyst
  - Acceptance testing
  - Customer service, End user
  - Role:
    - We want this (one voice!)
- *Stakeholders:*
  - Anyone interested in the project
  - Regulation bodies
  - Competitors
  - Other managers / divisions …
- *Development team*
  - Systems engineers
  - Software engineers
  - Hardware/computer engineers
  - Mechanical, etc.
  - Role:
    - Implement features upon customer demand
    - Give advise on feasibility
- *Expert*
  - Knows technical details of how something works
  - Expensive and busy

Types of Communication

	How many people?	Direction?	Style?
Email	Multiple	Unidirectional	Asynchronous
Phone call	Two	Full duplex	Synchronous
Instant messaging	Two/Multiple	Nearly full duplex	Asynchronous
Group chat	Multiple	At will	Asynchronous
Web meeting	Multiple	Full multiplex	Synchronous (Scheduled)
Shared screen	Few	Full duplex	Synchronous
Whiteboard	Multiple	At will	Asynchronous

Face-to-face meeting is most effective, but:

- large overhead and effort: takes everyone's time
- geographical distribution
- long product life-cycle: people no longer there

In your homework:

- **Joint team meetings (during course slot)**
- **Basecamp + Slack**
- **Magic Draw team server**
- **Skype, telephone, etc.**

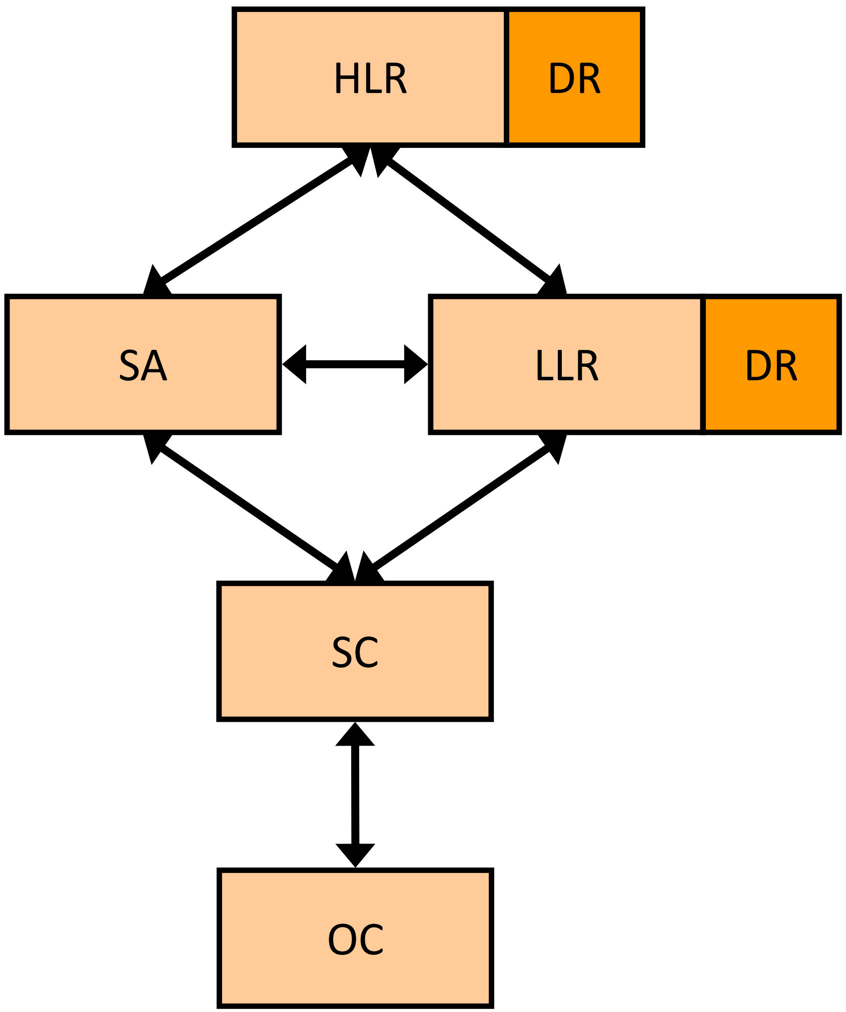
What is a Requirement?

Definition, types, traceability

Definition of a Requirement

- **Definitions**
  - A condition or capability a system must conform to (IBM Rational)
  - A statement of the functions required of the system (Mentor Graphics)
- Each requirements needs to be
  - **Identifiable + Unique**: unique IDs
  - **Consistent**: no contradiction
  - **Unambiguous**: one interpretation
  - **Verifiable**: e.g. testable to decide if met
- Captured with special statements and vocabulary

The Certification Perspective: High-level vs Low-Level



Concepts from DO-178C standard

- High-level Requirements (HLR):
  - customer-oriented
  - black-box view of the software,
  - captured in a natural language (e.g. using shall statements)
- Derived Requirements (DR)
  - Capture design decisions
  - *Derive from* customer reqs
- Low-level Requirements (LLR):
  - SC can be implemented without further information
- Software Architecture (SA)
  - Interfaces, information flow of SW components
- Source Code (SC)
- Executable Object Code (EOC)

Functional vs Extra-functional

Functional

- Core technical goal

The train shall close its doors upon remote request by the operator

Extra-functional

- Performance
- Dependability
- Safety
- Security
- …

The closing of doors should take no more than 4 sec

The mechanism operating the doors shall endure 5 years of continuous use without maintenance

The door must never hurt a passenger when closing

A secure cover shall protect the mechanism against vandalism

Functional vs Extra-functional

- Typical scope (not always true)
  - Functional req.: specific to a given component
  - Extra-functional: fulfilled by the system as a whole
- **Derivation** possible across different kinds
  - Customer HLR safety:
    „The door must never hurt a passenger when closing”
    →
  - Derived HLR functional:
    „The door must be able to detect obstruction”

How to Write Requirement?

Good practices and antipatterns

Good practices for writing textual requirements

- A textual requirement contains
  - a short description(stand-alone sentence / paragraph)
  - of the problem and not the solution
- English phrasing:
  - Pattern: Subject Auxiliary Verb Object Conditions
    - Example:
      The railway operator shall create a direct route between any two points on the track
  - Be precise! (Quantitative is better than qualitative)
  - Avoid passive sentences
- Use of auxiliaries:
  - Positive: shall/must > should > may
  - Negative: must not > may not
  - They specify priorities!

Examples

Functional:

- The operator shall be able to change the direction of turnouts
- Train equipments shall periodically log sensor data with a timestamp

Safety:

- The system shall ensure safe traffic within a zone
- The system shall stop two trains if they are closer than a minimal distance
- No single faults shall result in system failure

Performance:

- The system should allow five trains per every 10 minutes

Dependability:

- The allowed downtime of the system should be less than 1 hour per year
- The system shall continue normal operation within 10 minutes after a failure

Supportability:

- The system shall allow remote access for maintenance

Security:

- The system shall provide remote access only to authorized personnel

Usability:

- The user interface should contain only 3 alerts at a time

Anti-patterns

1. The system should be safe
2. The system shall use Fast Fourier Transformation to calculate signal value.
3. The system shall continue normal operation soon after a failure.
4. Sensor data shall be logged by a timestamp
5. Unauthorized personnel could not access the system

Too general / high-level

Describes a solution (and not only the problem)

Imprecise (how to verify „soon”?)

Passive should be avoided!

Use specific auxiliaries!

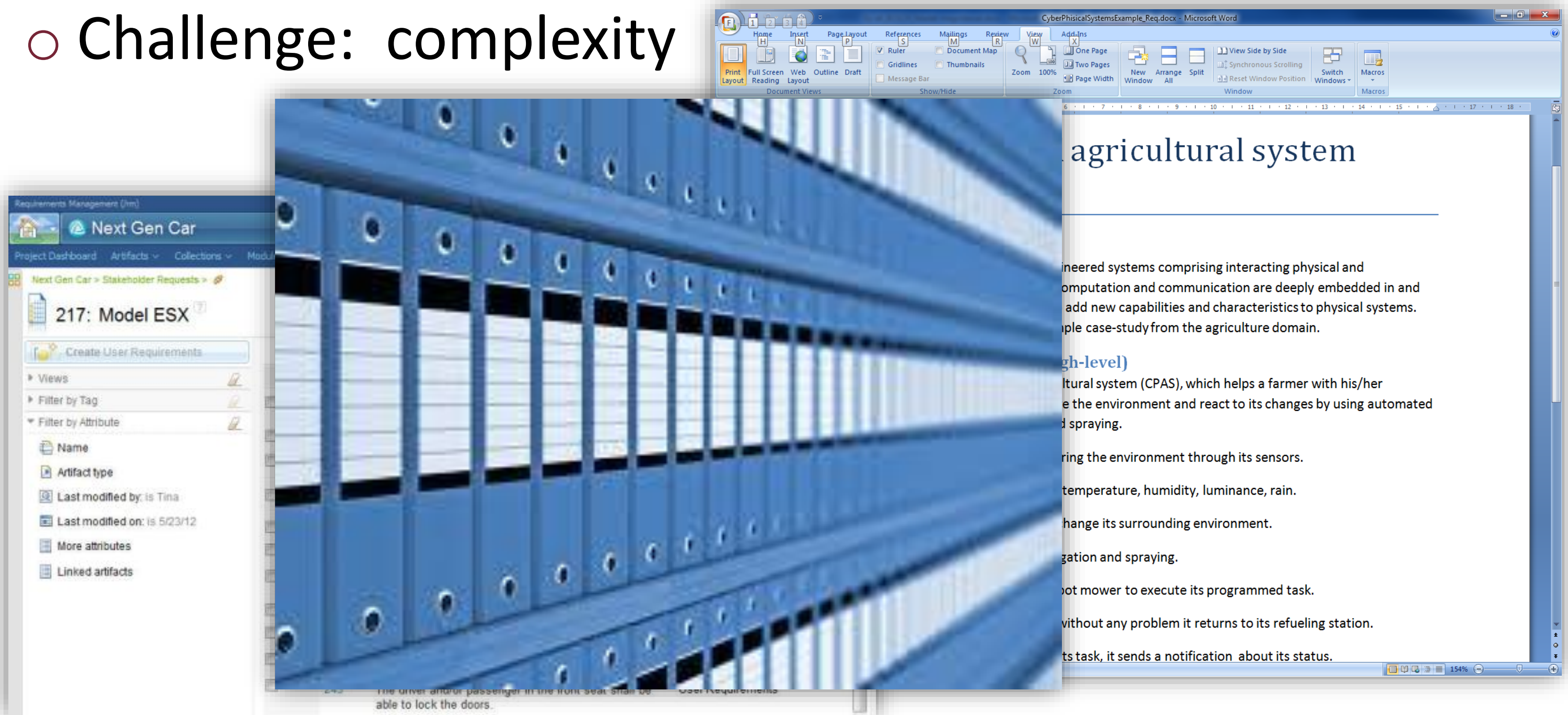
How to identify missing or inconsistent requirements?

Modeling Requirements in SysML

SysML overview, Requirements Diagram

Roots & Relations

- Document based system development
  - Formulated requirements textually (e.g. in Word)
  - Handled by Req. management tools (e.g. DOORS)
  - Challenge: complexity



SysML overview (System Modeling Language)

- „UML for Systems Engineering”
  - Supports the specification, analysis, design, verification and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- Developed by OMG and INCOSE (International Council on Systems Engineering)
- OMG SysML™ (http://www.omgsysml.org)
  - RFP – March 2003
  - Version 1.0 – September 2007
  - Version 1.1 – November 2008
  - Version 1.2 – June 2010
  - Version 1.3 – June 2012
  - Version 1.4 – September 2015
  - Version 1.5 – May 2017

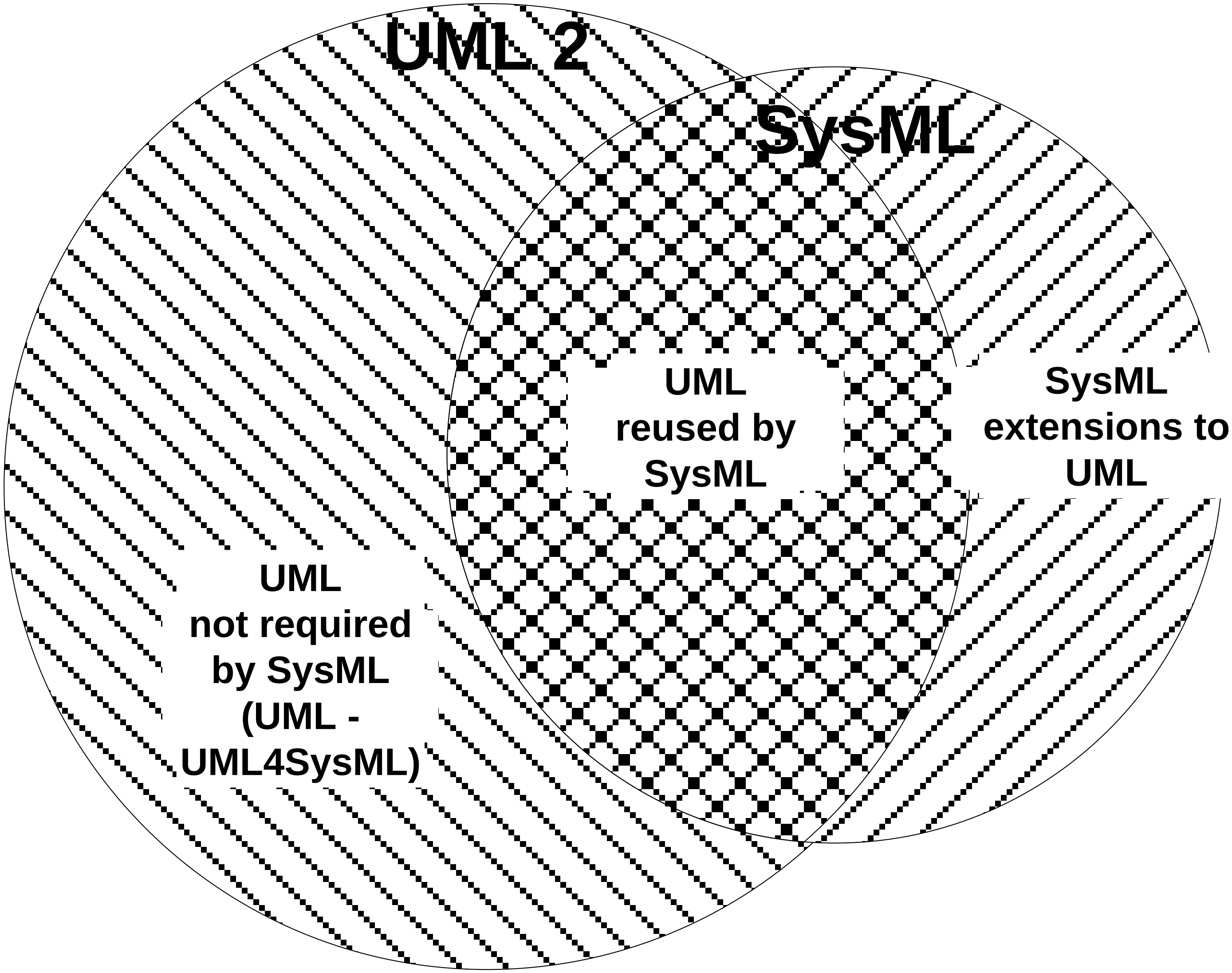
SysML good to know

- SysML is for interdisciplinary systems
- Examples for systems:
  - Railway, Automobile, Spacecraft, Factory, etc.
  - Thirty Meter Telescope is designed with SysML (tmt.org)
- SysML is only a language, how it is used is another question – model only what is important
- Methodologies (recommendations, best practices)
  - SYSMOD
  - NASA System Engineering Handbook
  - OOSEM (Object-Oriented Systems Engineering Method)
  - ESEM (Executable System Engineering Method)

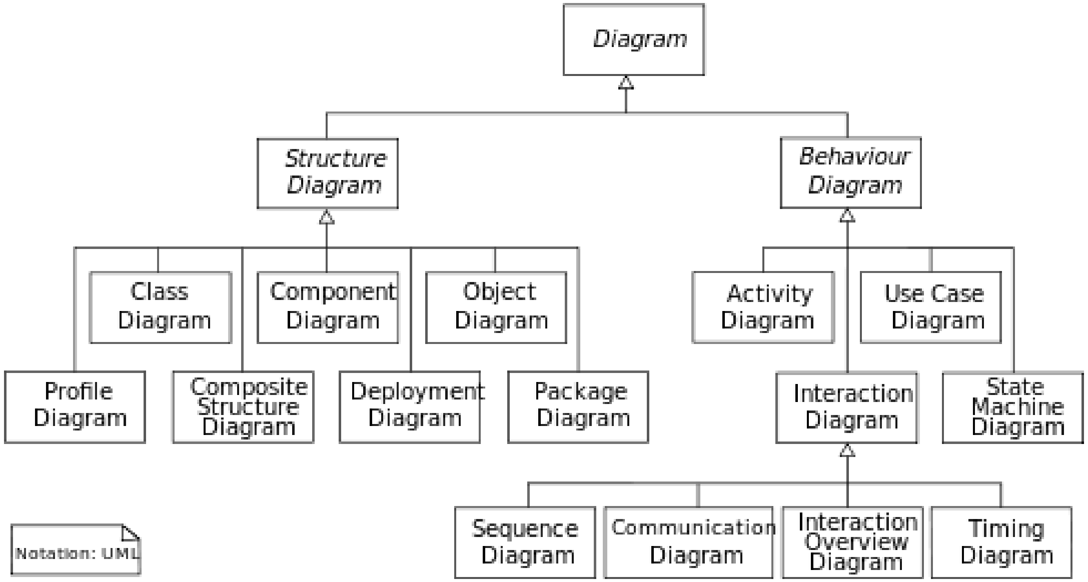
Recommended materials

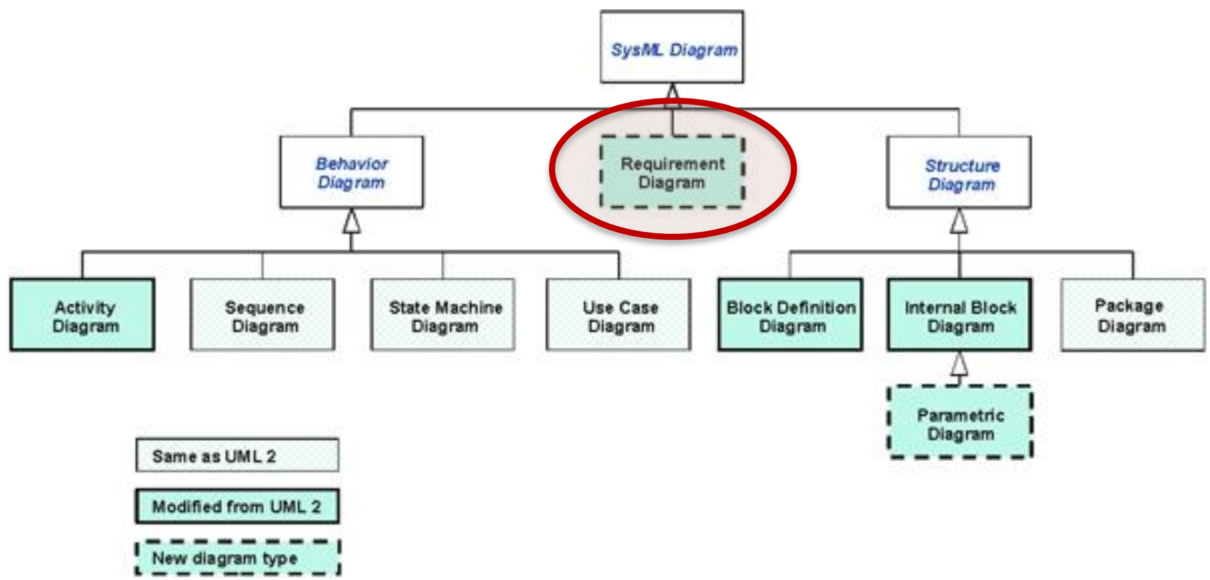
- Books
  - Tim Weilkiens:
    - SYSMOD – The System Modeling Toolbox
    - Systems Engineering with SysML/UML (older version)
  - Sanford Friedenthal, Alan Moore, Rick Steiner: A Practical Guide to SysML
    - More precise with the syntax, good examples, practices
- Web pages
  - http://www.uml-diagrams.org/
    - Good references to notations, but **only UML**

Relationship Between SysML and UML



Requirements Diagram

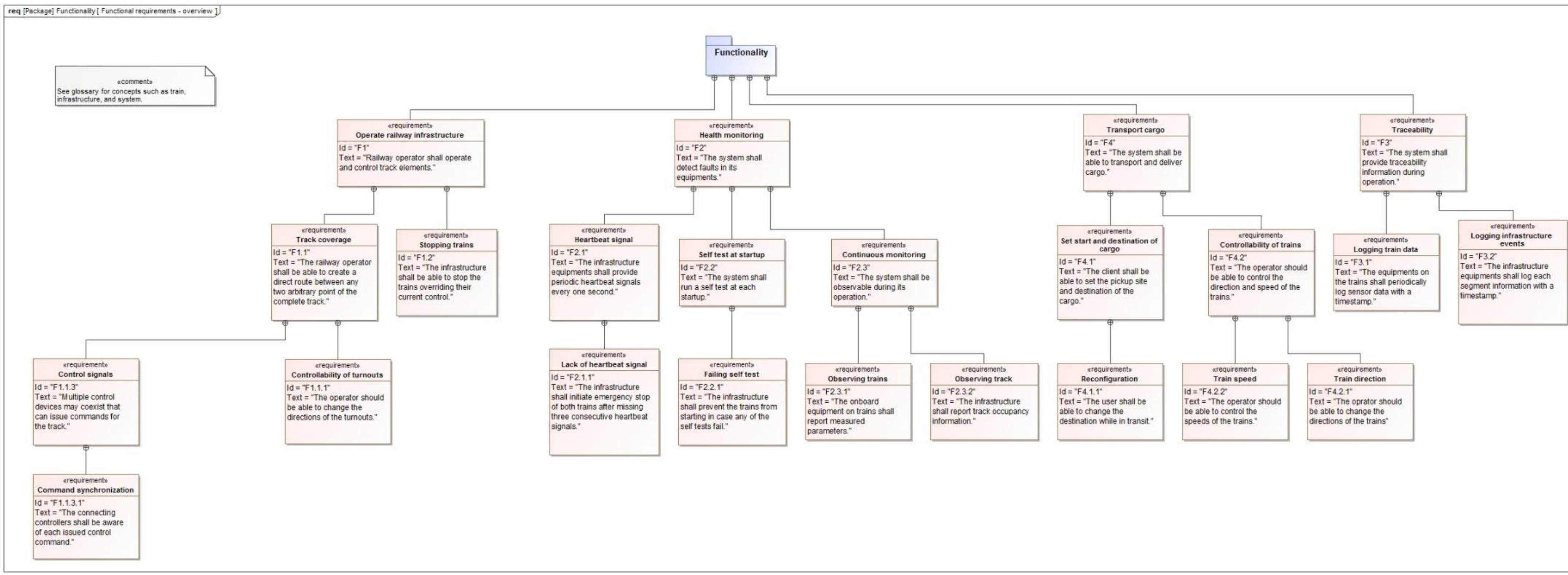




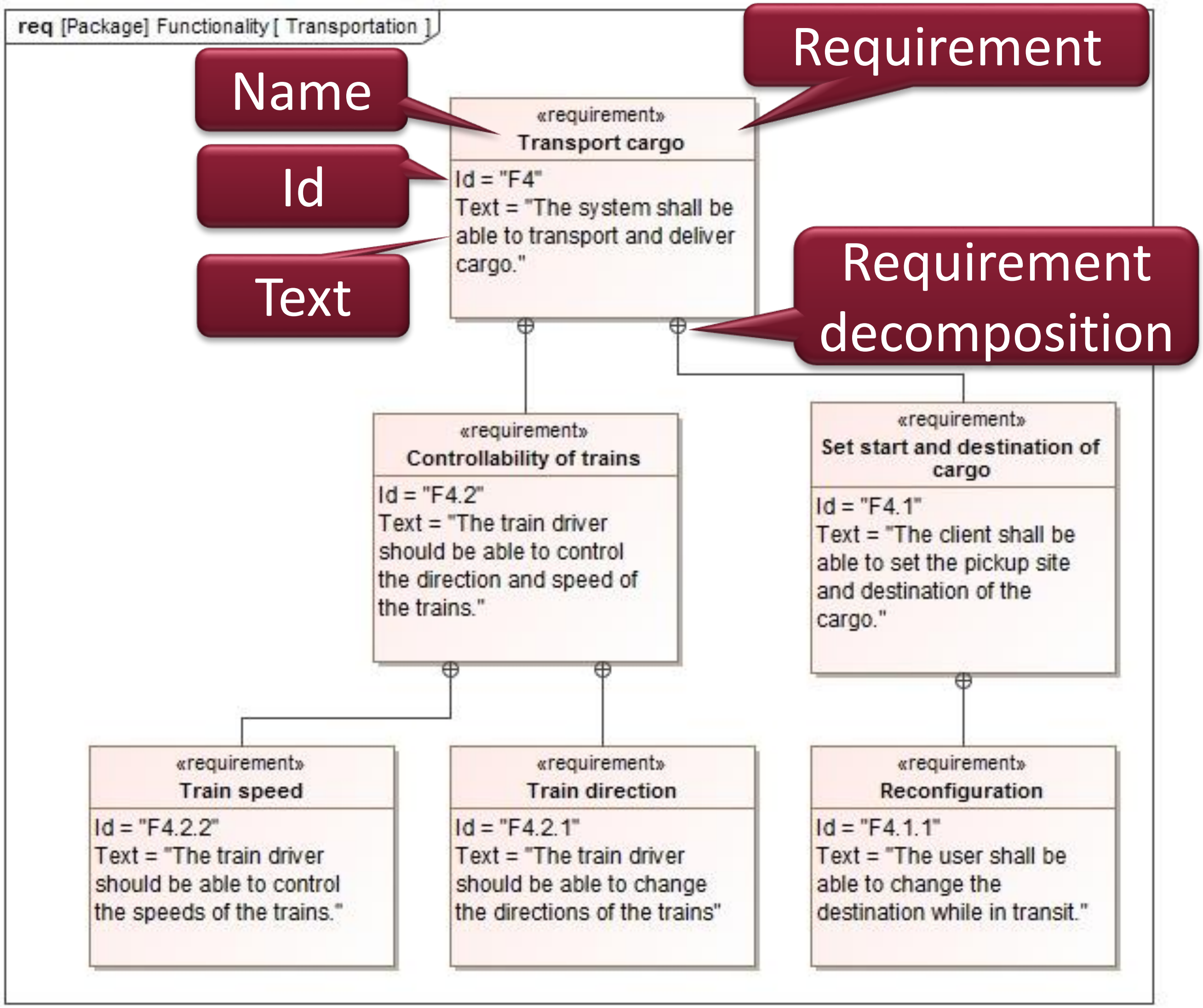
Main Goal of Requirements Diagram

What are the main textual requirements?

What is their hierarchy?

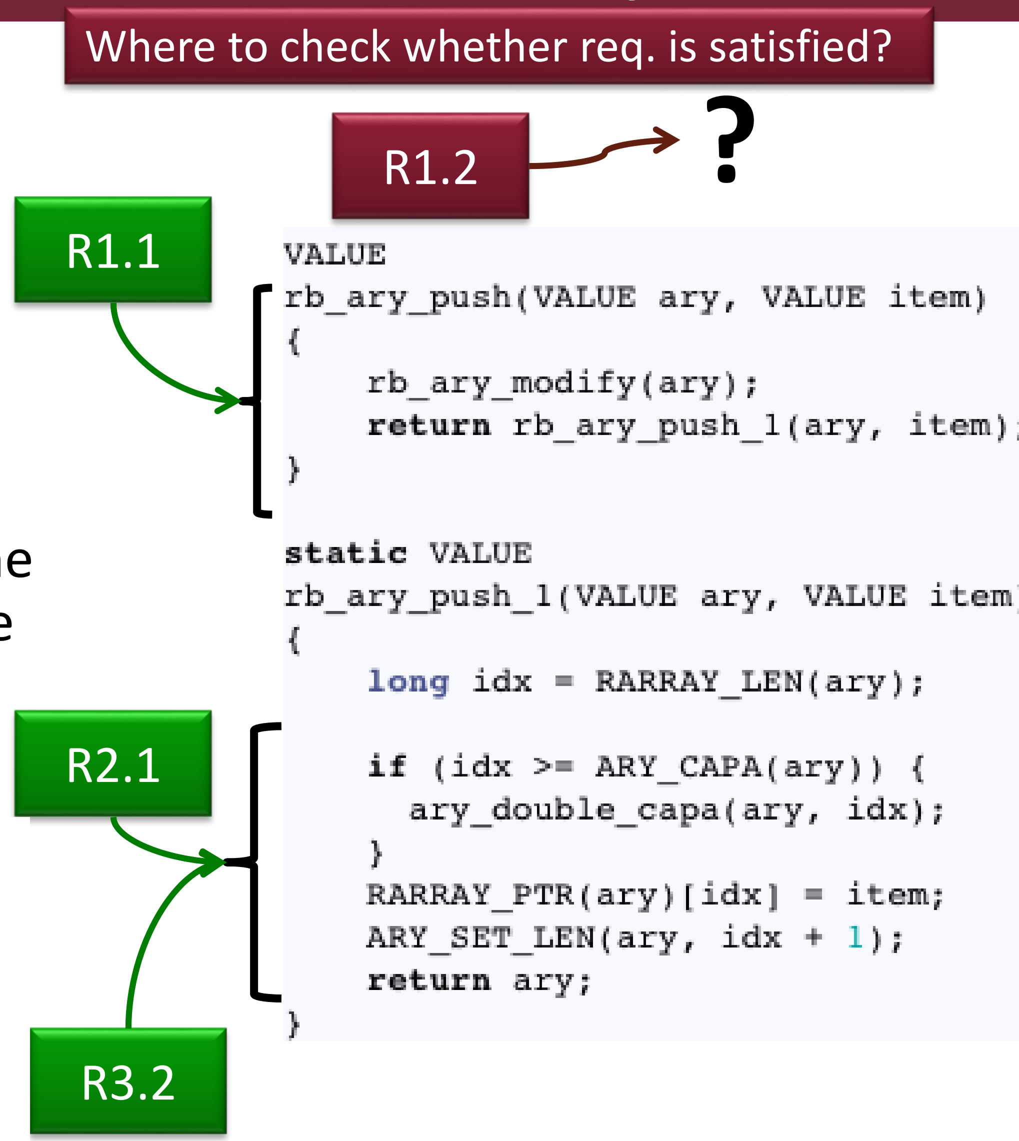


SysML Example – Requirements



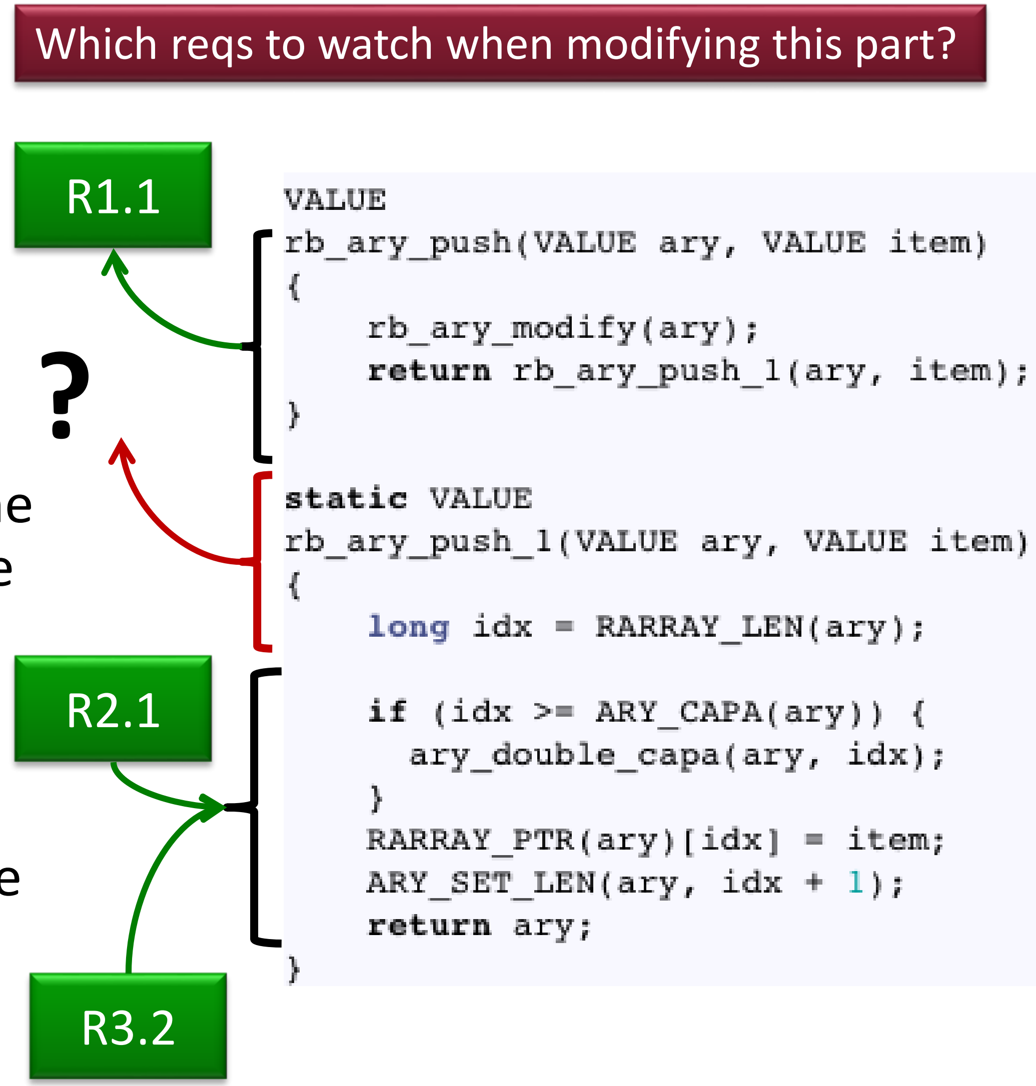
The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility



The Concept of Traceability

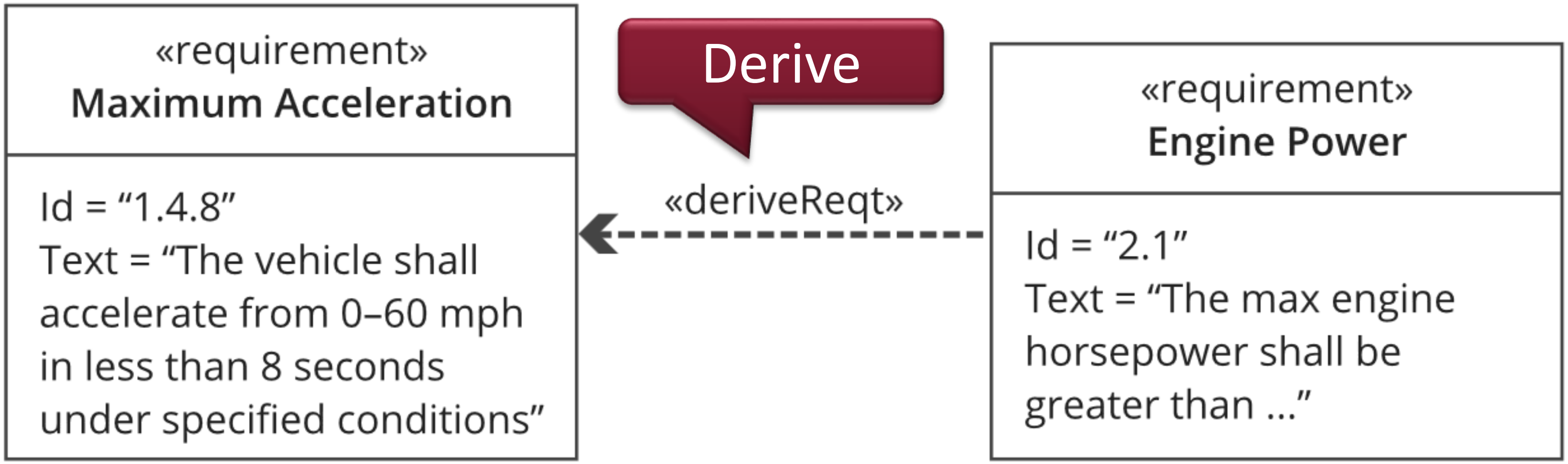
- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility
- **Backward traceability**:
  - From any lines of source code to one ore more corresponding requirements
  - No extra functionality

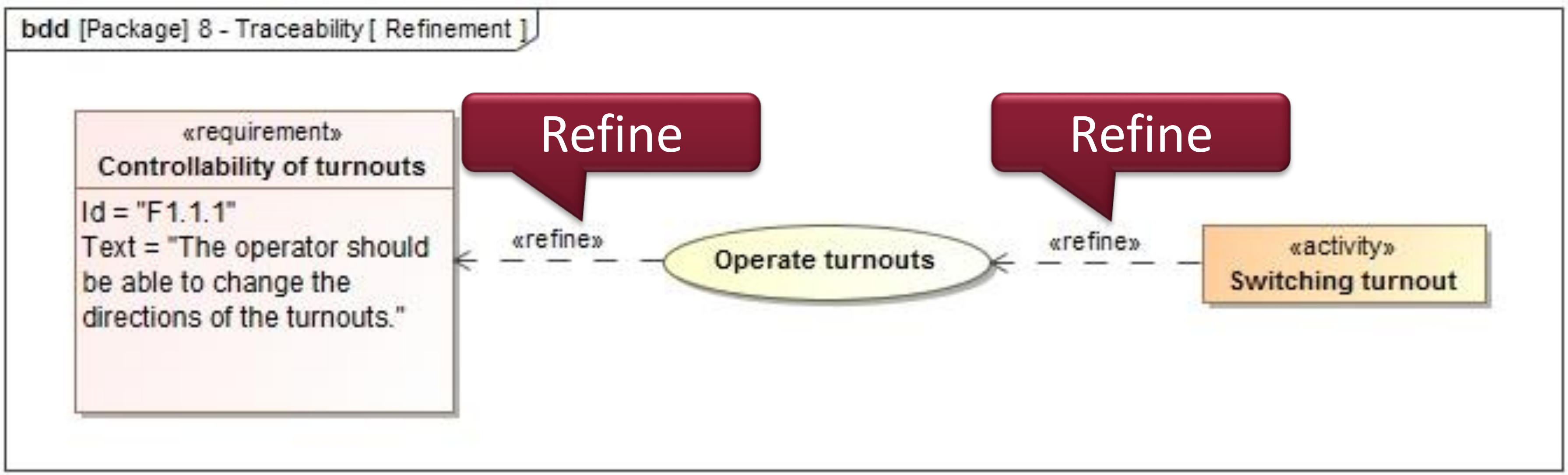


Relations between Requirements

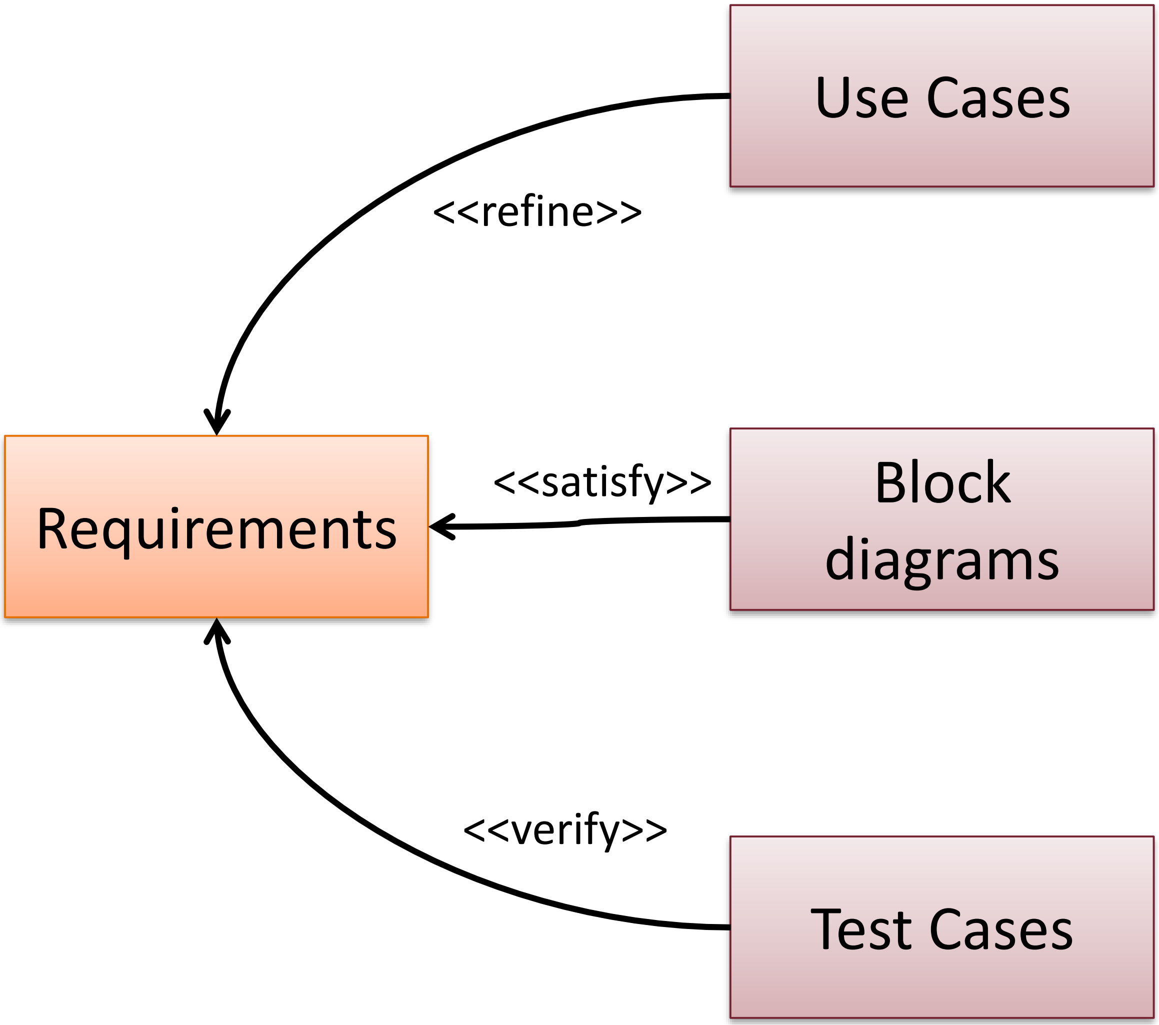
- **Trace**
  - General trace relationship
  - Between requirement and any other model element
- **Refine**
  - Depicts a model element that clarifies a requirement
  - Typically a use case or a behavior
- **Derive**
  - A requirement is derived from another requirement by analysis or decision
  - Typically at the next level of the system hierarchy
- **Copy**
  - Supports reuse by copying requirements to other namespaces
  - Master-slave relation between requirements
- **Satisfy**
  - Depicts a design or implementation model element that satisfies the requirement
- **Verify**
  - Used to depict a test case that is used to verify a requirement

Examples of Relations between Requirements





Traceability of Requirements in SysML Models



Requirements Relations in Table

#	Id	Name	Text	Traced To
24	P1	Cost efficiency	The system shall choose one of the cheapest ways of delivering the cargo to the destination in a safe way.	SAFE_1 Safe traffic
25	P2	Swift delivery	The delivery of the cargo shall be as fast as the safe operation of the railway allows and the route is economical.	P1 Cost efficiency R2 High availability
26	R2.1	Low downtime	Allowed downtime of the system is one hour per year.	
27	R2.2	Fast recovery	The system should continue normal operation withi[…] hours after a failure. (MTTR = 8h)	
28	R2	High availability	The transportation system shall provide its services […]	
29	S1.1	[…]	The system shall provide remote access to the staff […]bers.	
30	S1.2.1	[…]	[…]nnel only with extra authority may access the system.	
31	S1.2	Secure access	[…]tenance staff should access the system securely.	
32	S1	Maintainability	There shall be access points for the system for maintenance and update.	
33	SAFE_1.	Safety within a zone	The infrastructure shall ensure safe traffic within a zone.	

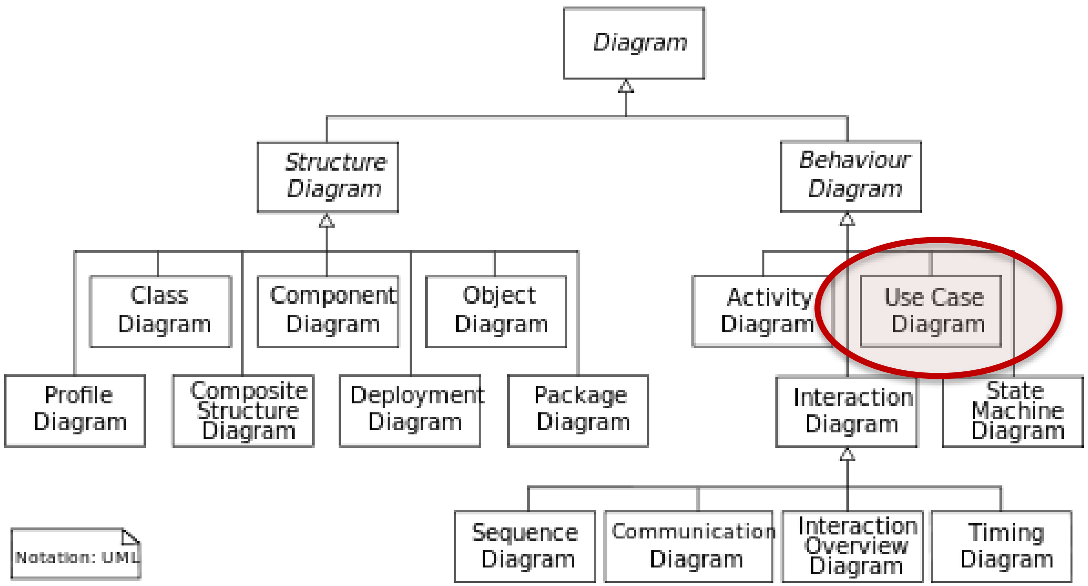
Traceability links

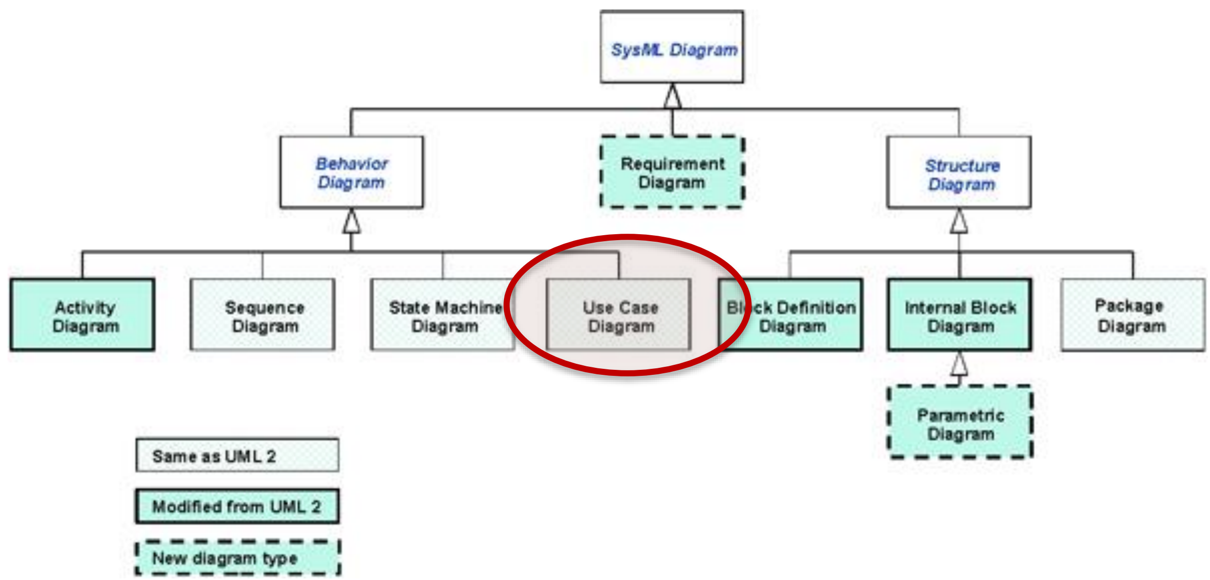
Hierarchical numbering

Modeling System Functions with Use Cases

Use Case Diagrams, System Context, Actors

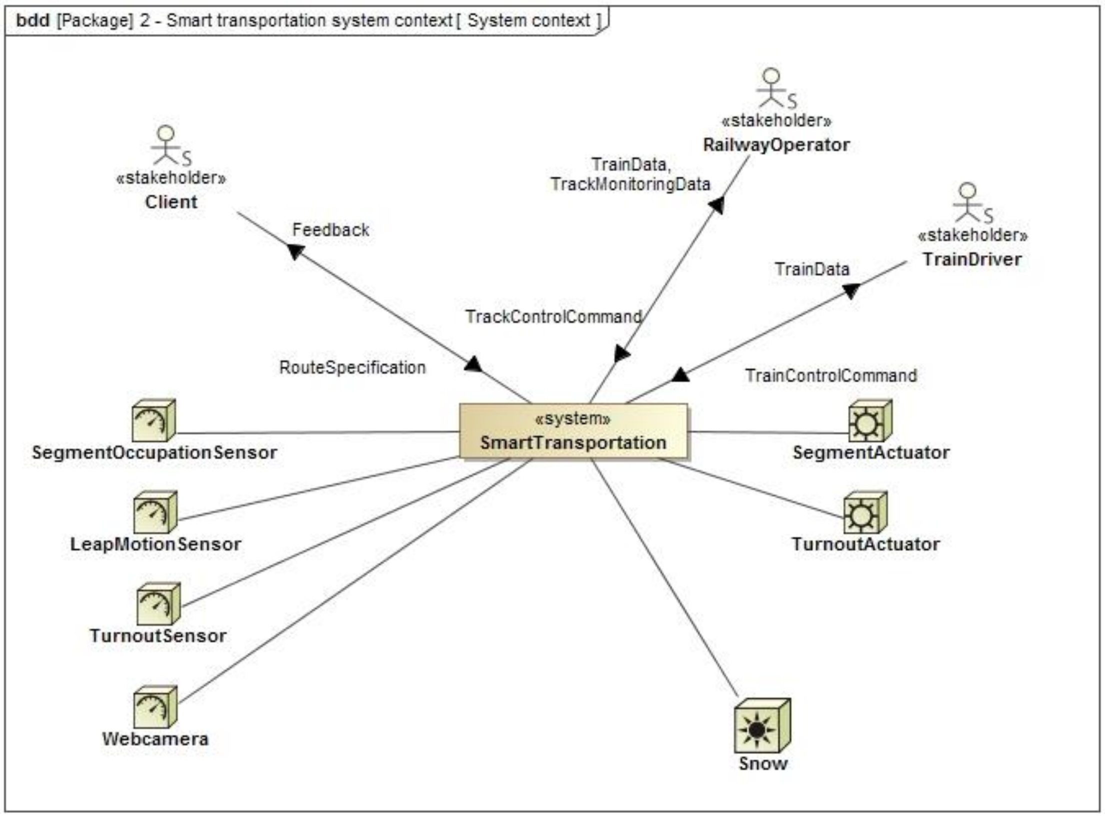
Use Case Diagrams





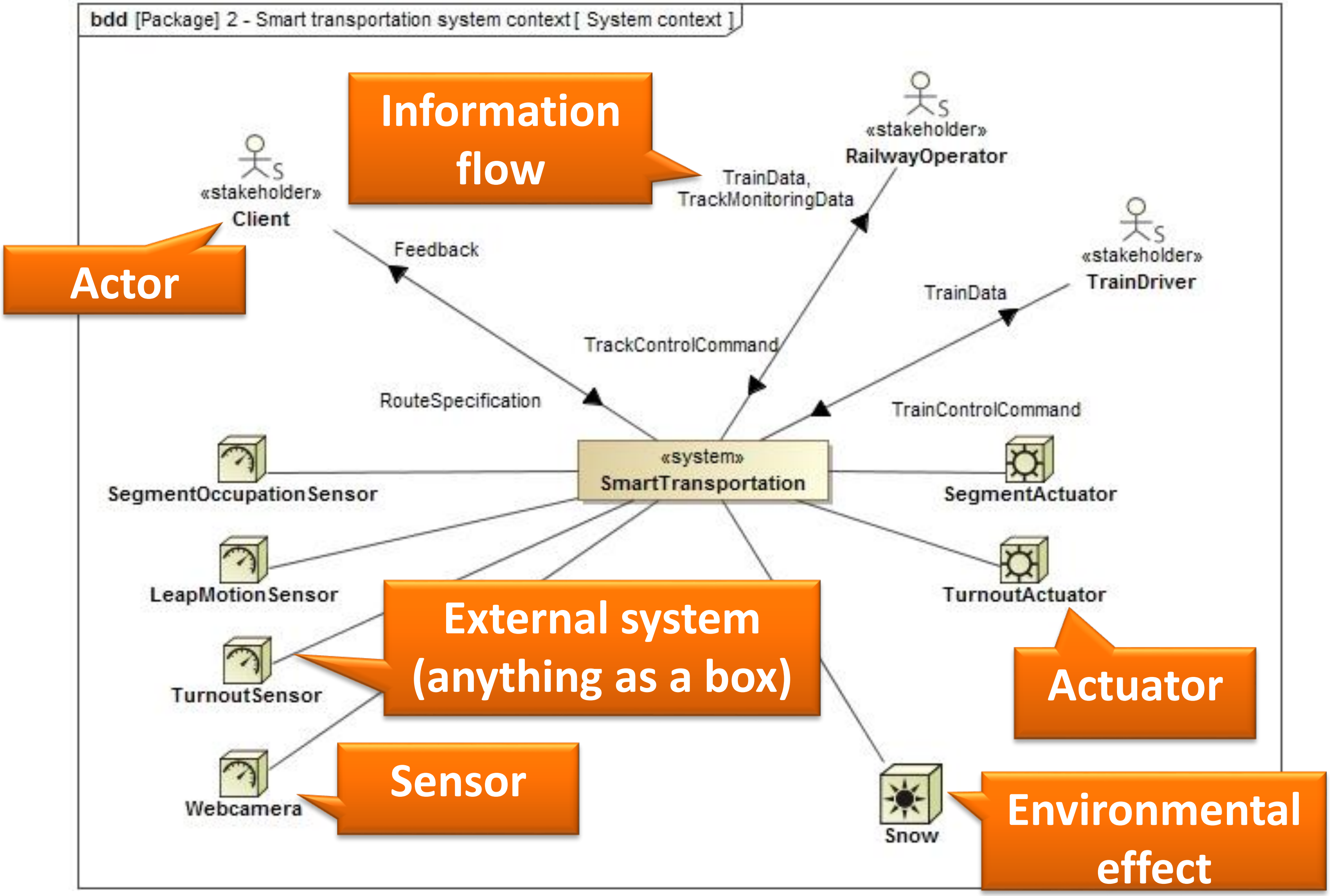
System Context

Who will use the system?



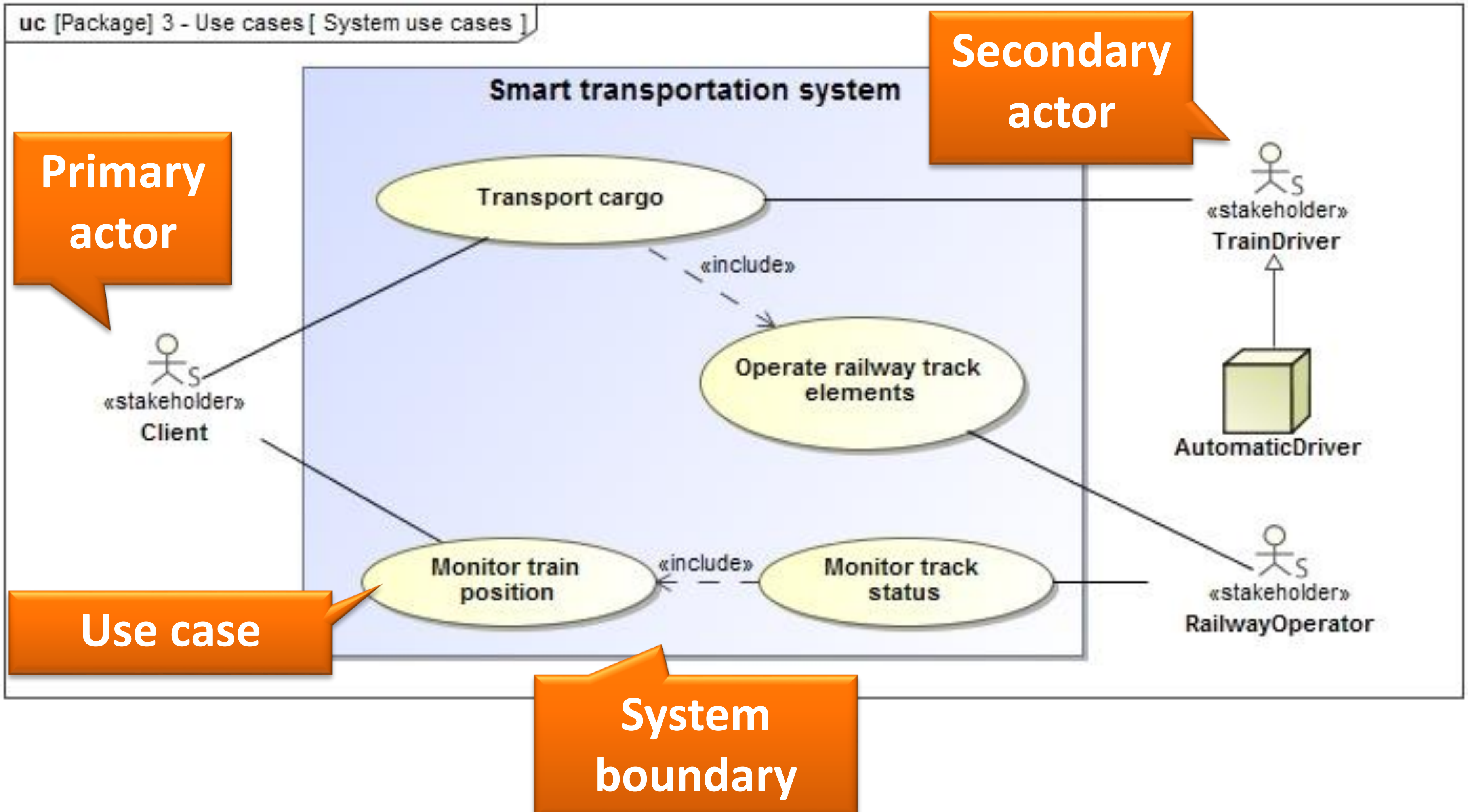
- Context diagram
  - System
  - Its boundaries
  - External entities
  - Incoming / outgoing
    - Information (data) flow
    - Control flow
- What form?
  - Whiteboard drawing
  - SysML UC diagram (context diagram)

SysML notation: Actors and External systems



Use cases

Who will use the system ***and for what****?*



Definition of Use Cases

- **Use case (használati eset)** captures a main functionality of the system corresponding to a functional requirement
- UCs describe
  - the typical interactions
  - between the *users* of a *system* and
  - **the system itself,**
  - by providing a narrative of how a system is used
- A set of scenarios tied together by a common user goal
- Language template: **Verb + Noun (Unique)!**
  - Example: Drive train, Switch turnout

M. Fowler: UML Distilled.
3rd Edition. Addison-Wesley

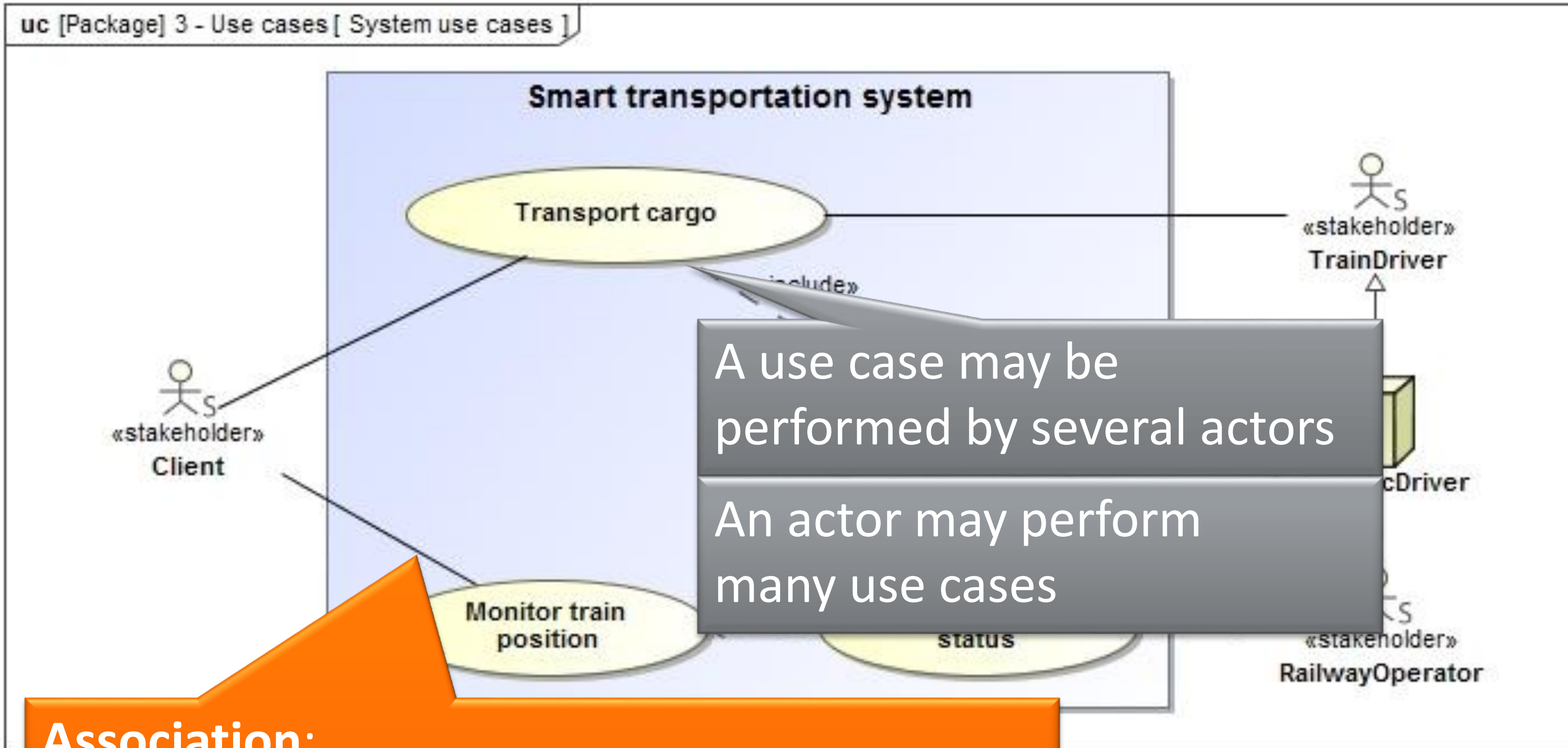
Use Case Descriptions

- Additional textual description to detail use cases
  - **Preconditions**: must hold for the use case to begin
  - **Postconditions**: must hold once the use case has completed
  - **Primary flow**: the most frequent scenario(s) of the use case (aka. main success scenario)
  - **Alternate flow**: less frequent (or not successful)
  - **Exception flow**: not in support of the goals of the primary flow
- Elaborated behavior in SysML (discussed later)
  - Activity diagrams: scenarios with complex control logic
  - Interaction diagrams: for message based scenarios

Definition of Actors

- **Actor** **(aktor, szereplő)** is a **role** that a user plays with respect to the system.
  - *Primary actor*: invokes the system to deliver a service
  - *Secondary actor*: the system communicates with them while carrying out the service
- An actor is outside the boundary of the system
- *Characteristics:*
  - One person may act as more than one actor
    - Example: The farmer may also act as a laborer who performs the spraying
  - Can be an external subsystem (and not a person)

Relations between Actors and Use cases

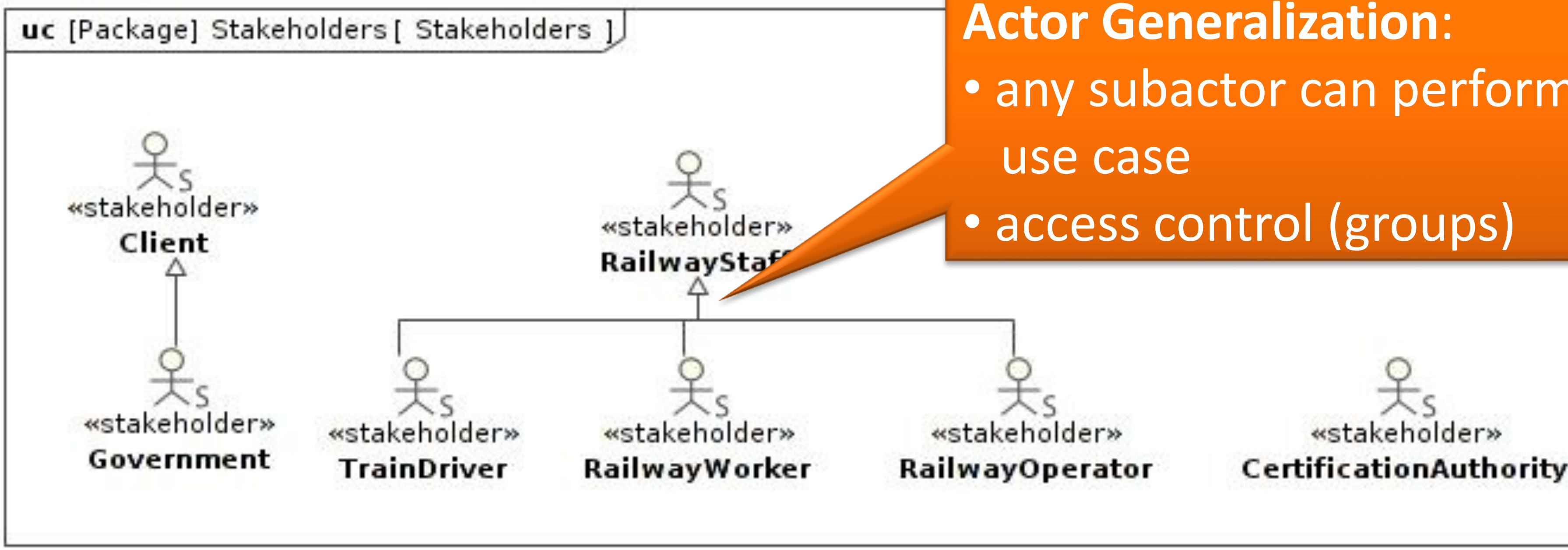


A use case may be performed by several actors

An actor may perform many use cases

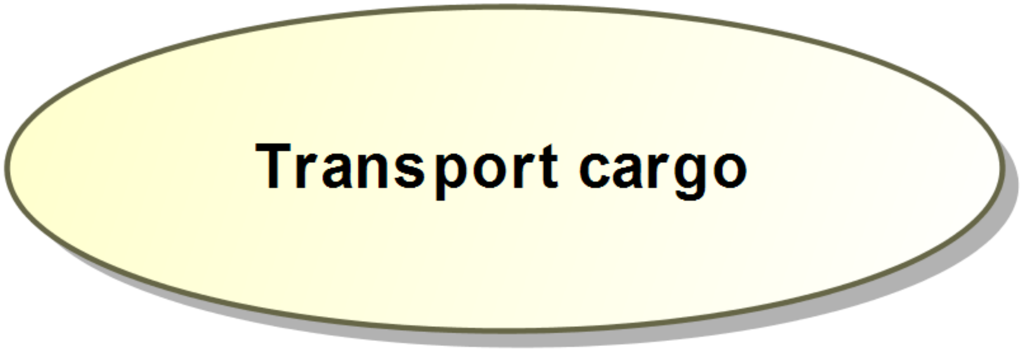
Association:
- primary actor initiates or
- secondary actor participates in interaction
- (rarely between 2 actors)

Relations between Two Actors



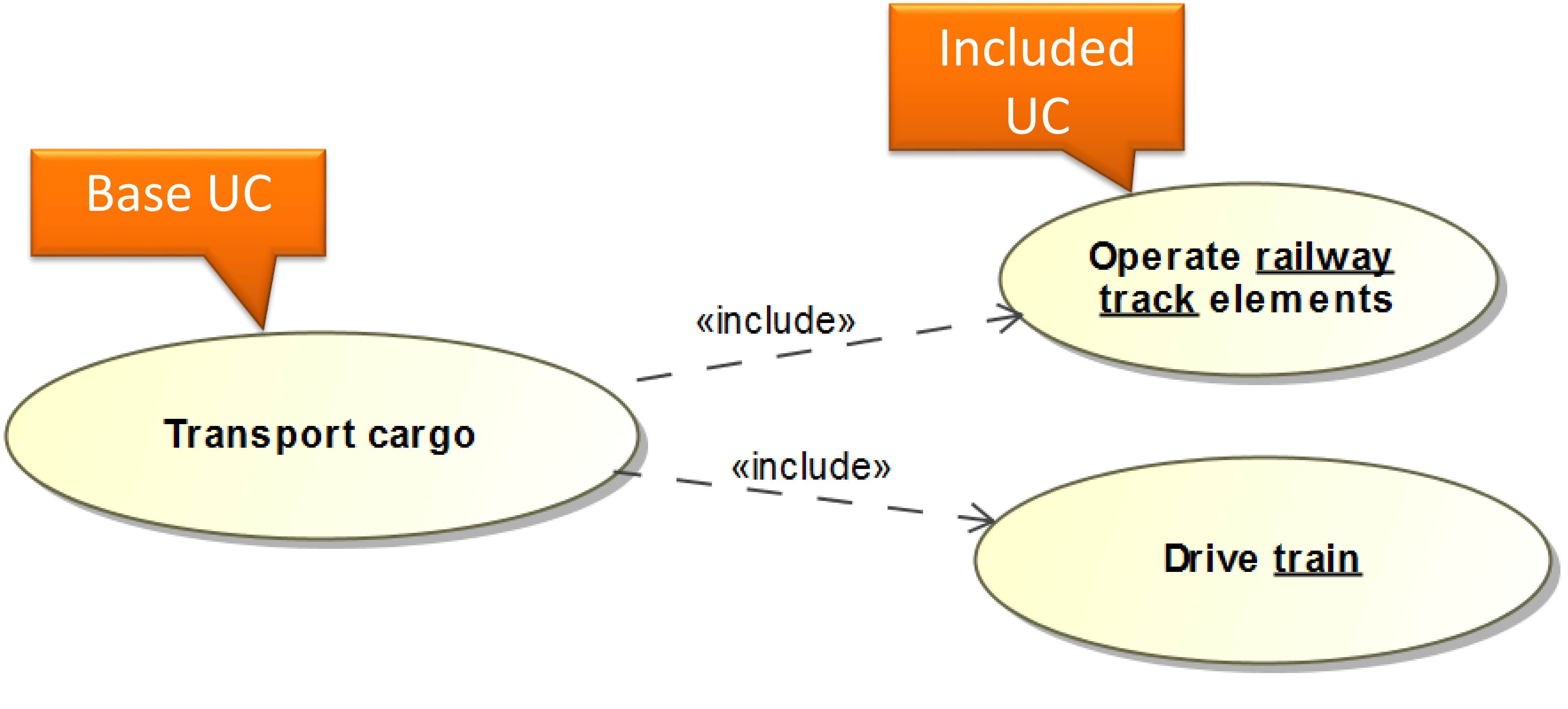
Actor Generalization:
- any subactor can perform use case
- access control (groups)

How to handle complex functionality?



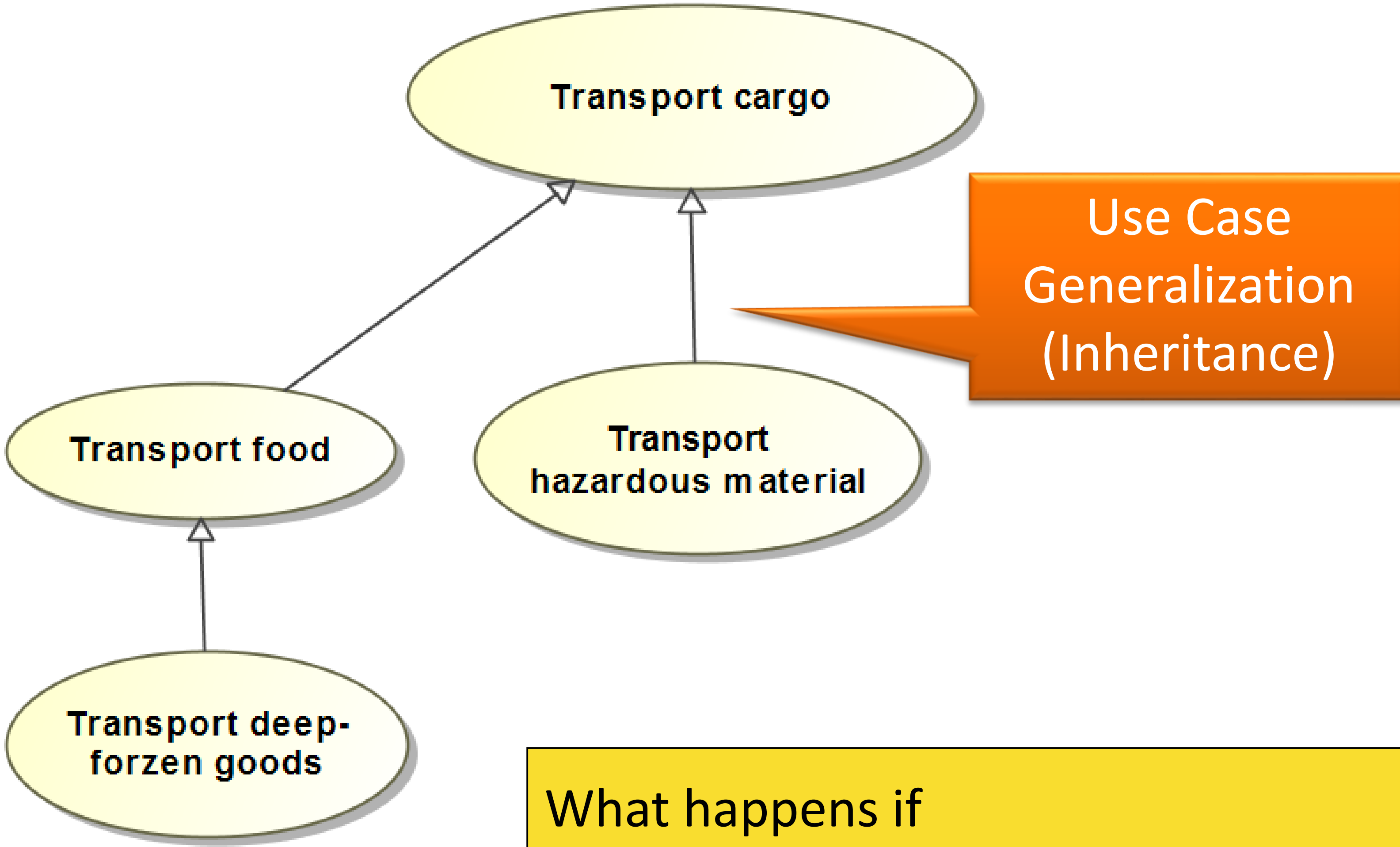
Transport cargo =
- Operate turnouts
- Drive train

Refinement with include relation



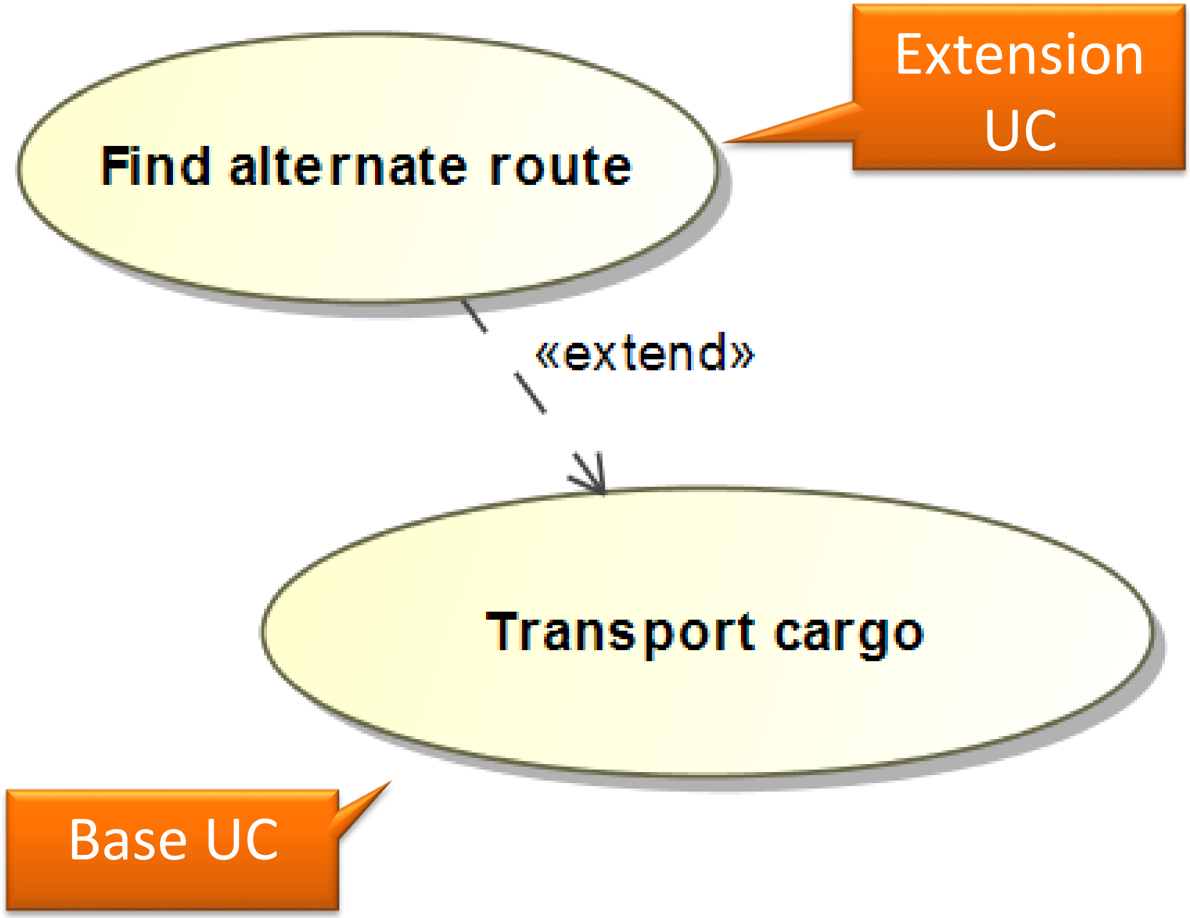
The included UC breaks down the complex core functionality into more elementary steps

Generalization of UCs



What happens if
- the selected route of transportation is blocked?

Extend relationship



The extension UC Extends core functionality by handling unusual (*exceptional*) situation

Overview of UC Relations

Association

- Actor – use case (rarely: actor – actor)
- an actor initiates (or participates in) the use of the system

Generalization

- actor – actor OR use case – use case
- a UC (or actor) is more general than another UC or actor

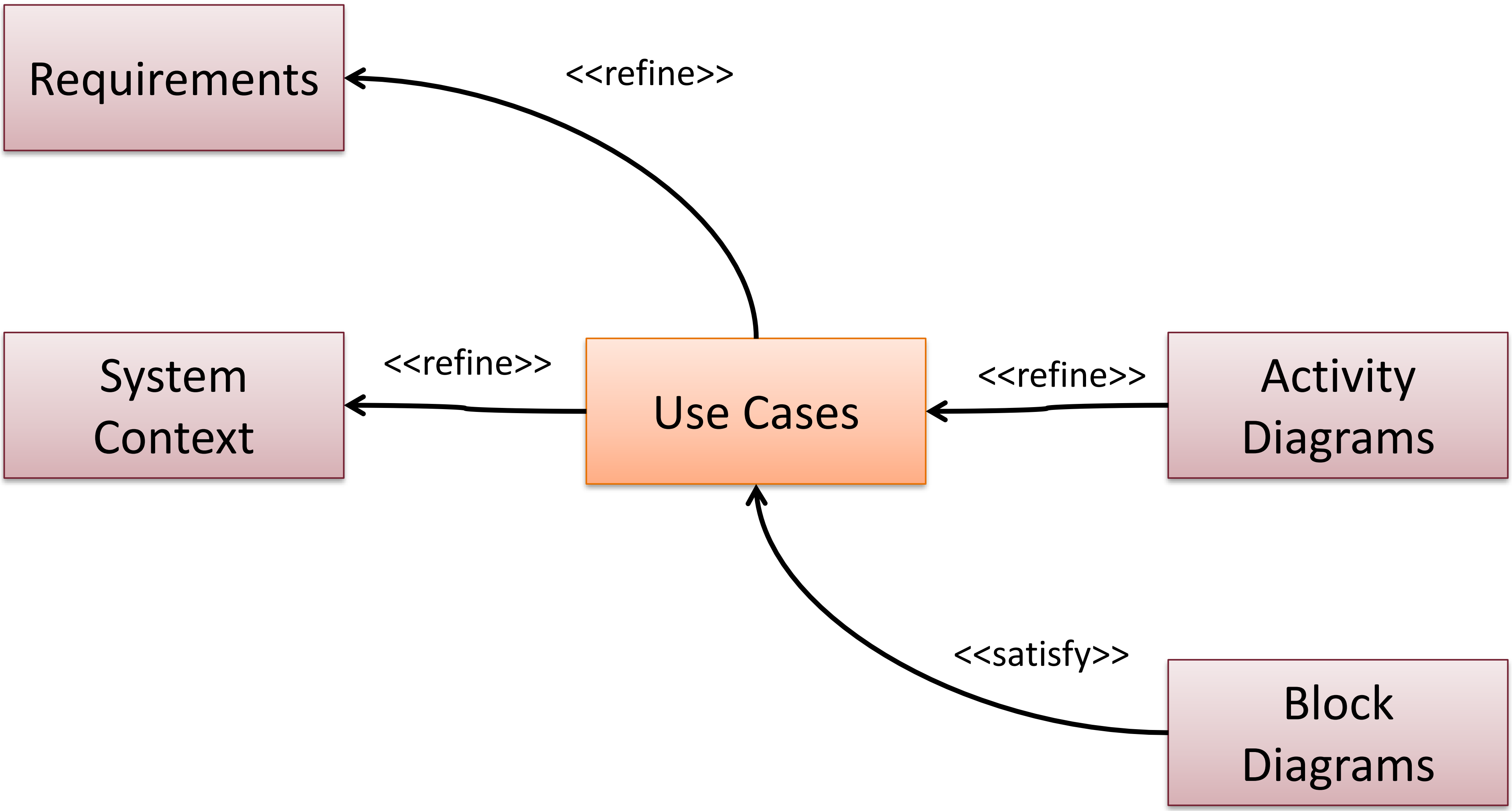
Includes

- use case – use case
- a complex step is divided into elementary steps
- a functionality is used in multiple UCs

Extend

- use case – use case
- a UC may be extended by another UC
- typically solutions for exceptional situations

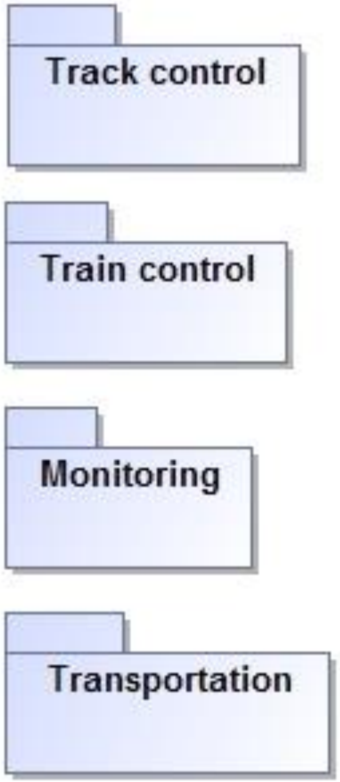
Traceability of Use Cases in SysML Models



Good practices of UC analysis

Good practice: Grouping

- Grouping UCs
  - Identify functional building blocks
  - Group them into packages
  - NOTE: related by functionality, NOT by role



- Grouping actors:
  - Dedicated (top-level) „Actors” package OR
  - Keep actors in a package within the subsystem they exclusively belong to

Good practice: Naming and arrangement

- Actors
  - Name actors according to their roles and avoid using job titles
  - Divide complex roles into multiple actors
  - Start the diagram by placing the most important actor in the top left corner
- Use Cases
  - Use domain specific verbs for UCs
  - Avoid technical descriptions – UCs are frequently for non-technical reader
- Relationships
  - Avoid crossing or curved lines when drawing relations
  - Use <<extend>> and <<include>> relations „lightly"
  - Place them into the appropriate functional block

Main guideline:
UC diagrams
should be *SIMPLE*

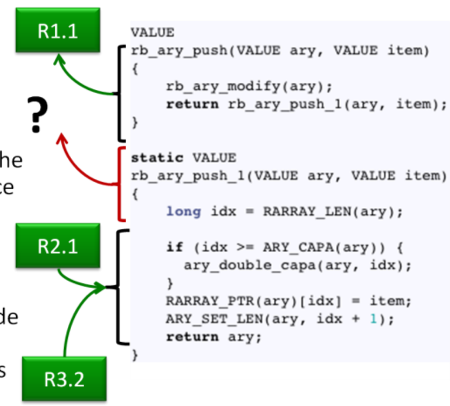
Summary

Definition of a Requirement

- Definitions
  - A condition or capability a system must conform to (IBM Rational)
  - A statement of the functions required of the system (Mentor Graphics)
- Each requirements needs to be
  - **Identifiable + Unique**: unique IDs
  - **Consistent**: no contradiction
  - **Unambiguous**: one interpretation
  - **Verifiable**: e.g. testable to decide if met
- Captured with special statements and vocabulary

The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility
- **Backward traceability**:
  - From any lines of source code to one ore more corresponding requirements
  - No extra functionality



Definition of Use Cases

- **Use case (használati eset)** captures a main functionality of the system corresponding to a functional requirements
- UCs describe
  - the typical interactions
  - between the *users* of a *system* and
  - **the system itself,**
  - by providing a narrative of how a system is used
- A set of scenarios tied together by a common user goal
- Language template: **Verb + Noun (Unique)!**
  - Example: Drive train, Switch turnout

M. Fowler: UML Distilled. 3rd Edition. Addison-Wesley

Definition of Actors

- **Actor (aktor, szereplő)** is a **role** that a user plays with respect to the system.
  - *Primary actor*: invokes the system to deliver a service
  - *Secondary actor*: the system communicates with them while carrying out the service
- An actor is outside the boundary of the system
- *Characteristics:*
  - One person may act as more than one actor
    - Example: The farmer may also act as a laborer who performs the spraying
  - Can be an external subsystem (and not a person)