TU Clausthal

Institute for Software and Systems Engineering

Requirement Engineering

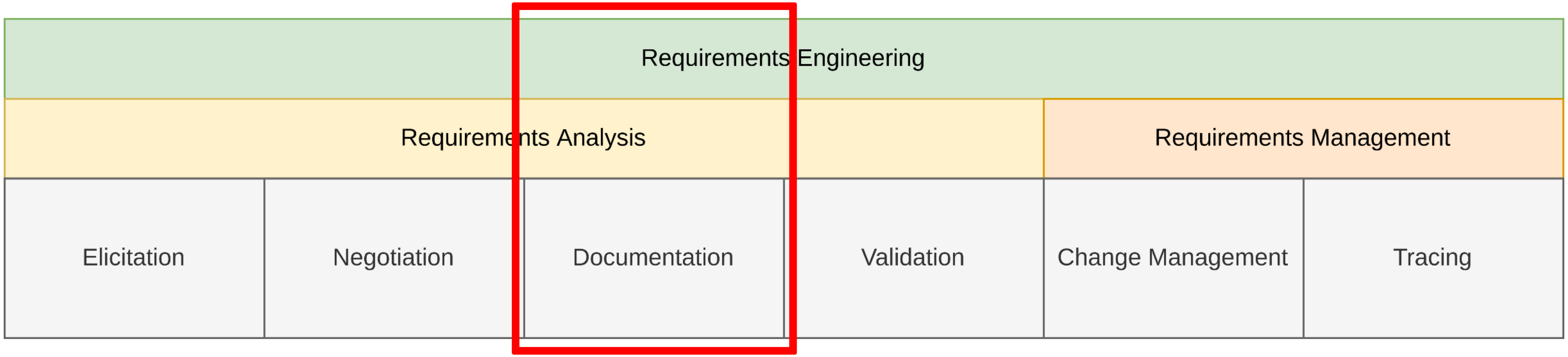
Lecture 4: Requirements Documentation

Part 3

Prof. Dr. Benjamin Leiding
Anant Sujatanagarjuna

General Requirements Engineering Process

Overview



Lecture 4: Requirements Documentation

Content

1. Model-based Requirements Documentation Techniques
2. Formal Specification Techniques

MODEL-BASED REQUIREMENTS DOCUMENTATION TECHNIQUES

Lecture 4: Requirements Documentation

Content

Model-based Requirements Documentation Techniques

Models in General – Requirements Model vs. Design Model

- Models are frequently used for system design
  - “Design Models”
  - E.g., architectural models

- Considerable difference between requirements models and design models
  - Requirements models depict aspects of the underlying problem
  - Design models document solutions chosen during system development

Model-based Requirements Documentation Techniques

Models in General – The Term “Model”

- According to Merriam-Webster:
  - *Structural design*
  - *A usually miniature representation of something*
  - *A system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs*

- We use the following definition in this lecture:
  - *A model is an abstract representation of an existing reality or a reality to be created.*

Model-based Requirements Documentation Techniques

Models in General – Properties of Models

- Mapping of reality
  - Aspects of the observed reality are mapped onto model elements
  - Descriptive model creation → Model documents the existing reality
  - Prescriptive model creation → Model prototypes fictious reality
  - Models can be both descriptive and prescriptive at the same time
    - Describes a stakeholder
    - Prescribes a use case of a system

Model-based Requirements Documentation Techniques

Models in General – Properties of Models

- Reduction of Reality
  - Models do not capture the complete reality
  - Instead, the models reduce the captured reality
  - Only particular aspects of the system are modeled
  - Subject matter is summarized during compression
- Pragmatic Property
  - Models serve a special purpose
  - Models are within a special context
    - **NOT general purpose!**
  - Purpose affects the construction of models and the reduction of the reality
  - Ideally contains only information pertaining to its purpose

Model-based Requirements Documentation Techniques

Models in General – Properties of Models

- Defined through **syntax** and **semantics**

- Syntax
  - Defines the modeling elements to be used
  - Specifies their valid combinations
- Semantics
  - Defines the meaning of the individual model elements
  - Foundation for the interpretation of the models

- Can be **formal**, **informal**, and **semiformal**
  - Depends on the magnitude of formal definitions

Model-based Requirements Documentation Techniques

Models in General – Advantages of Models

- Humans handle graphically depicted information better
  - Perceived faster
  - Memorized faster
  - Also true for requirements models

- Strictly defined focus
  - Everything not part of the focus of the model is removed → Removal of noise

- Harmonized level of abstraction
  - Modeling elements dictate the level of abstraction

Model-based Requirements Documentation Techniques

Models in General – Suppression of Details

- Complexity is reduced by abstraction
- Three main mechanisms
  - Selection
    - Selects a particular aspect to be depicted by the model
    - Other aspects are ignored completely, i.e., not part of the model
  - Aggregation
    - Combines aspects into aggregated aspects
    - Condenses information
  - Classification/generalization
    - Identifies common features
    - Suppresses differences between the common features
    - Commonalities are represented as generalized information

Model-based Requirements Documentation Techniques

Models in General – UML

- Object Management Group (OMG) standard
  - Current version UML 2.5.1
- Graphical notation for the analysis, design, and documentation of object-oriented systems
- UML is **not**
  - a development process
  - specialized for a certain topic
  - complete & formal
    - Cannot be complied without additional information
    - → Semiformal
  - Capable of semantics
    - UML only provides a syntax
    - Semantics depend on the reader of the document

Lecture 4: Requirements Documentation

Content

Model-based Requirements Documentation Techniques

Goal Models – Goals in General

- Goals are the stakeholders description of system properties
  - What they want from the system
- Effort for goal considerations usually minimal
- Positive impact of goal modeling is high
  - Especially concerning the comprehensiveness and quality

Model-based Requirements Documentation Techniques

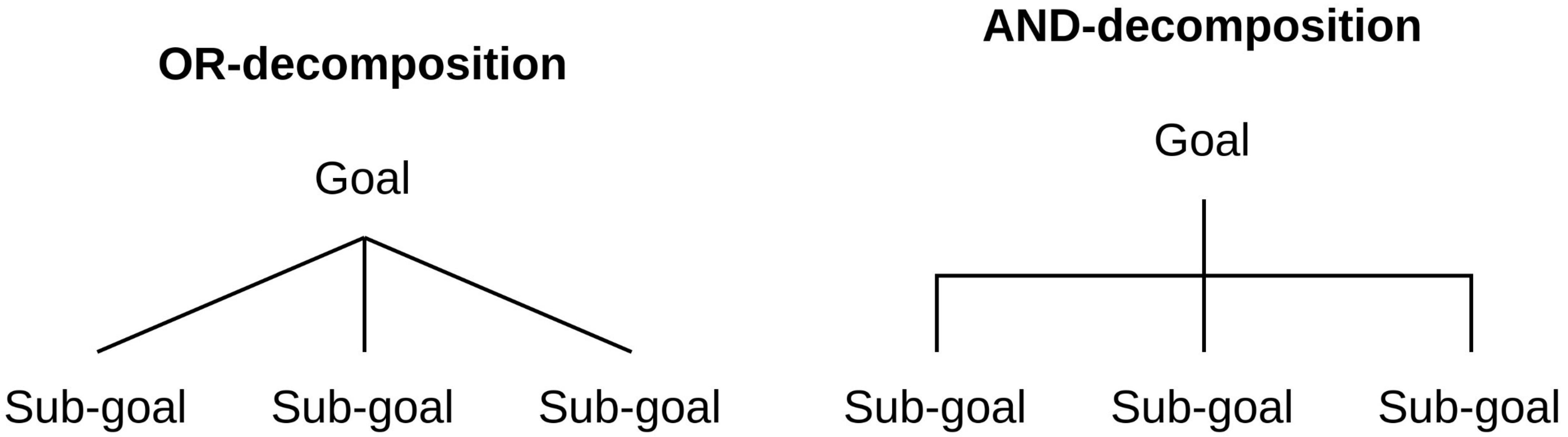
Goal Models – AND / OR Trees

- Documents hierarchical decompositions of goals into sub-goals

- Two types of decompositions
  - AND → All sub-goals must be fulfilled
  - OR → At least one sub-goal must be fulfilled

Model-based Requirements Documentation Techniques

Goal Models – AND / OR Trees



Lecture 4: Requirements Documentation

Content

Agent-oriented Modelling

Motivation: Why AOM?

- Most (if not all) processes in software systems are elicited by *Agents* playing a certain *Role* in the system, to achieve some *Goal.*
- AOM is a tool for *“modelling systems with multiple agents, both human and manmade, interacting with a diverse collection of hardware and software in a complex environment”*
- AOM models are clear and easily understandable for stakeholders → <u>useful for Requirements Engineering</u>

L. Sterling, K. Taveter (2009) – The Art of Agent-Oriented Modeling

Agent-oriented Modelling

Concepts and Definitions

- Goal: A situation description that refers to the intended state of the environment. Goals can:
  - be *functional* or *non-functional (quality).*
  - have sub-goals.
- "Goals are expressed by using nouns, verbs, and (optionally) adjectives. The nouns tend to be more of a state, and the verbs more into the activities that are needed to achieve a goal."
- e.g., if a *message* needs to be *transmitted securely*, the functional goal '*Transmit Message*' can be associated with the quality goal '*Securely*'

L. Sterling, K. Taveter (2009) – The Art of Agent-Oriented Modeling

Agent-oriented Modelling

Goal vs. Requirement

Goal	Requirement
Single desired result	Statement of need
One goal may consist of several requirements	One requirement may be related to many goals

- No one to one mapping between goals and requirements is possible

Agent-oriented Modelling

Concepts and Definitions

- How does one identify functional and non-functional goals?
  - Functional goals usually describe **what** a system must accomplish = Identification depends heavily on the system.
  - Non-functional goals describe **how** the system must accomplish those goals, in terms of standards and quality = Identification can depend on functional goals.
  - **However**, there are many commonalities: Reliability, Availability, Security, ...

https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119202660.app6

Agent-oriented Modelling

Concepts and Definitions

- Role: Some capacity or position that fascilitates the system to achieve it’s goals. Roles express functions, expectations, and obligations of the agents enacting them.
  - eg. Network Administrator, Firewall
- Agent: An entity that can act in the environment, perceive events, and reason.
  - Can be human or software

L. Sterling, K. Taveter (2009) – The Art of Agent-Oriented Modeling

Agent-oriented Modelling

Concepts and Definitions

- Activity**:** Some action performed by an agent playing a role in pursuance of a system goal.
- Environment: An abstraction that provides the surrounding conditions for agents to exist and that mediates both the interaction among agents and the access to resources.

L. Sterling, K. Taveter (2009) – The Art of Agent-Oriented Modeling

Agent-oriented Modelling

Models

- Models that we will take a look at:

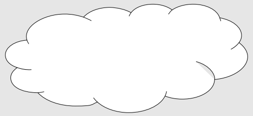
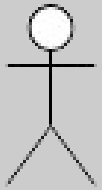
- Goal Models
- Behavioural Interface Models

Agent-oriented Modelling

Goal Models

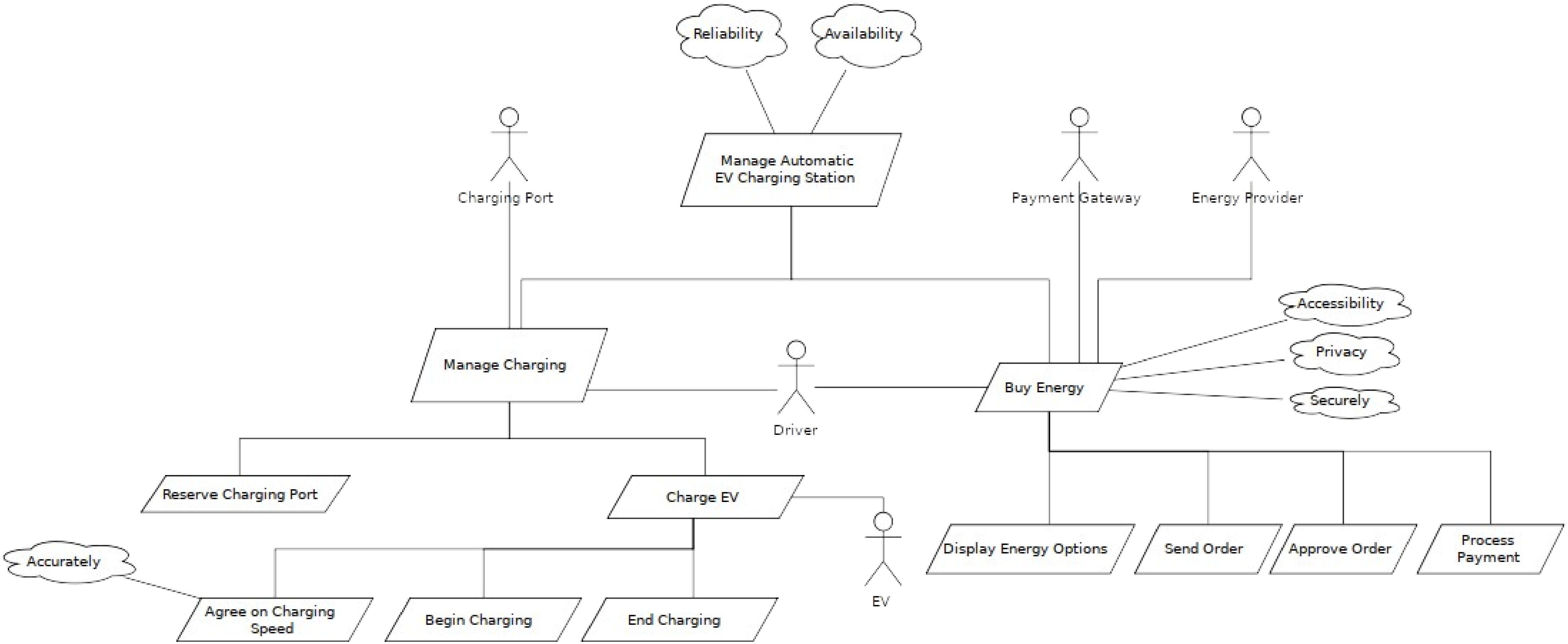
Goal models hierarchically express the relationships between goals (functional and non-functional) and the roles played by various agents in pursuit of those goals.

Sterling and Taveter’s AOM Goal models omit AND/OR decomposition for simplicity.

Symbol	Meaning
	Goal
	Quality Goal
	Role
	Reltionship between goals
	Relationship between goals and quality goals

Agent-oriented Modelling

Goal Model Example: Automated EV Charging Station



Agent-oriented Modelling

Behavioural Interface Models (BIM)

- Behavioral Interface Models model the behaviour of agents playing their roles
  - Models *Behavioural Units (= Activities)*
  - Represented as a table ↓

Activity	**Trigger(s)**	**Precondition(s)**	**Postcondition(s)**
Activity Name	*Event(s) that trigger(s) the activity*	*Conditions for Activity to proceed*	*Conditions for Activity to be considered complete*
…	…	…	…

Agent-oriented Modelling

BIM Example: Automated EV Charging Station (Manage Charging)

Activity	Trigger(s)	Precondition(s)	Postcondition(s)
Reserve Charging Port(CP)	Driver wants to charge EV	Driver has bought energy, CP is free, EV is ready to charge	Driver has reserved CP
Charge EV	"	Driver has reserved CP	Driver has charged EV
Agree on charging speed	"	Driver has reserved CP, Max CP speed ≥ Min EV speed, Max EV speed ≥ Min CP speed	Charging speed is agreed upon
Begin Charging	"	Charging speed is agreed upon	EV has begun charging
End Charging	None	EV has competed charging	Driver has charged EV, CP is free

Lecture 4: Requirements Documentation

Content

Model-based Requirements Documentation Techniques

Use Cases – Overview

- Method to document functionalities
  - Planned
  - Of existing system
- Relatively simple models
- Two concepts
  - Use case diagrams
  - Use case specification
- Both should be used in conjunction

Model-based Requirements Documentation Techniques

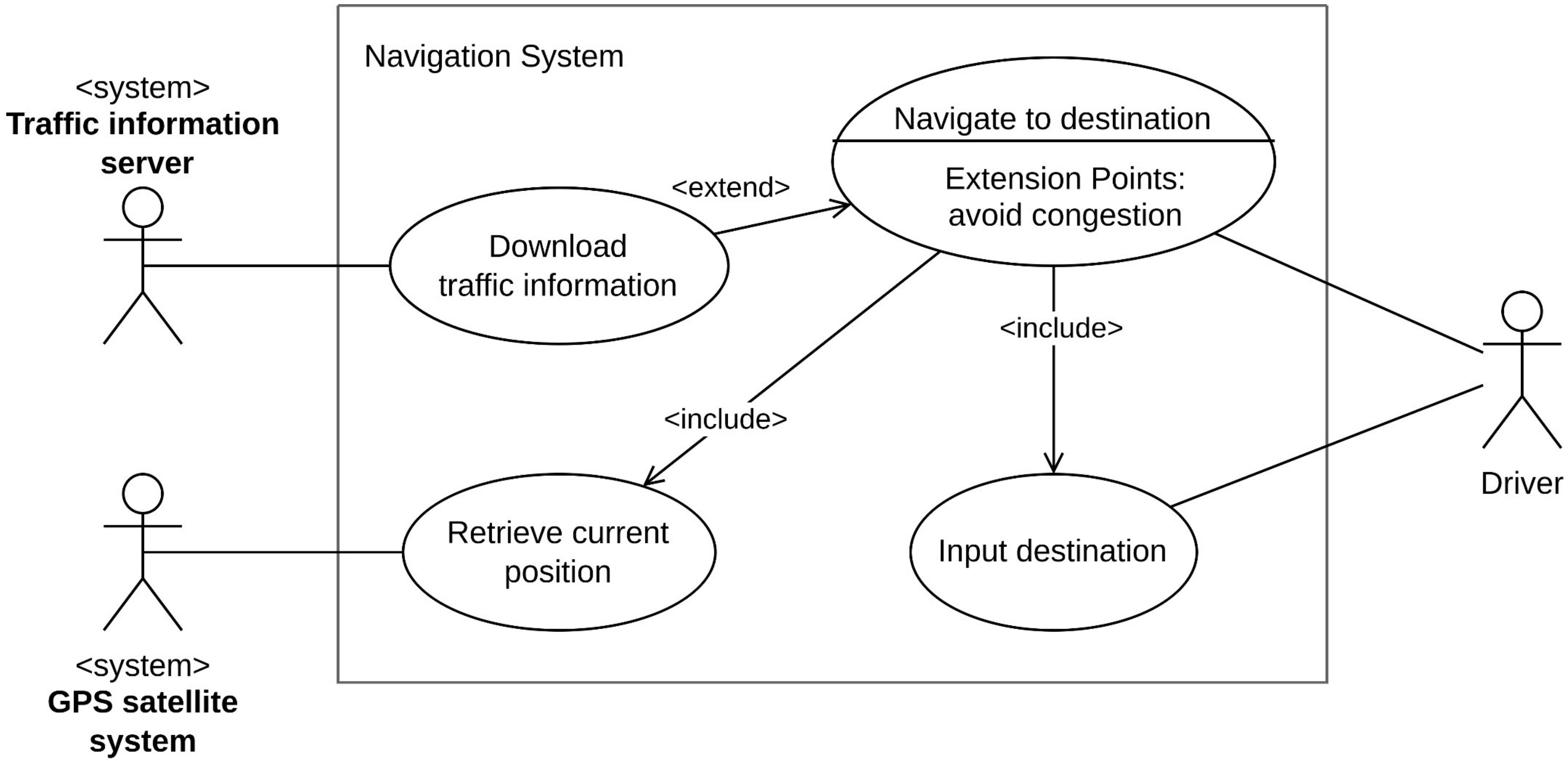
Use Cases – UML Use Case Diagrams

- Models to schematically depict:
  - Functions from a user’s point of view
  - Interrelations of functions of a system
  - Relations between functions and their environment

- **We do not cover all concepts of use case diagrams in this lecture**
  - Additional information can be found in the literature

Model-based Requirements Documentation Techniques

Use Cases – UML Use Case Diagram (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

Model-based Requirements Documentation Techniques

Use Cases – Issues of UML Use Case Diagrams

- Diagrams do not contain details
  - Very high level
  - Very abstract

- Examples for open questions
  - How does the driver communicate with the Navigate to destination use case?
  - Is there an order in the inclusion of the use cases Retrieve current location and Input destination?

Model-based Requirements Documentation Techniques

Use Cases – Use Case Specifications

- Use case specifications provide details to the diagrams
- Specifications documented textually
- Not simple prose, but in form of templates (usually tabular)
- The template defines the concrete information contained in the use case specification

Model-based Requirements Documentation Techniques

Use Cases – Use Case Specification Template

- Template prescribes the following information
  - Attributes for unique identification of use cases
  - Management attributes
  - Attributes for the description of the use case
  - Specific use case attributes, e.g.,
    - the trigger event,
    - actors,
    - pre- and post-conditions,
    - the result of the use case,
    - the main scenario,
    - alternative and exception scenarios,
    - cross references,
    - quality requirements

Model-based Requirements Documentation Techniques

Use Cases – Use Case Specification Template (Example)

Section	Content
Designation	UC-12-37
Name	Navigate to destination
Authors	John Smith, Sandra Miller
Priority	Importance for system success : high Technological risk : high
Criticality	High
Source	C. Warner (domain expert for navigation systems
Person Responsible	J. Smith
Description	The driver of the vehicle types the name of the destination. The navigation system guides the drive to the desired destination.
Trigger event	The driver wishes to navigate to his destination
Actors	Driver, traffic information system, GPS satellite system

Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

Model-based Requirements Documentation Techniques

Use Cases – Use Case Specification Template (Example)

Section	Content
Pre-conditions	The navigation system is activated
Post-conditions	The driver has reached his destination
Result	Route guidance
Main scenario	1. The navigation system asks for the desired destination 2. The driver enters the desired destination 3. The navigation system pinpoints the destination in its maps 4. On the basis of the current position and the desired destination, the navigation system calculates a suitable route 5. The navigation system compiles a list of waypoints 6. The navigation system shows a map of the current position and shows the route to the next waypoint 7. When the last waypoint is reached, the navigation system shows “destination reached” on the screen

Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

Model-based Requirements Documentation Techniques

Use Cases – Use Case Specification Template (Example)

Section	Content
Alternative scenario	4a. Calculation of the route must honor traffic information and avoid traffic congestions. 4a1. The navigation system queries the server for updated traffic information. 4a2. The navigation system calculates a route that does not contain any traffic congestions.
Exception scenarios	Trigger event: The navigation system does not receive GPS signal from the GPS satellite system.
Qualities	→ QR.04 (reaction time upon user input) → QR.15 (operating comfort)

Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

Lecture 4: Requirements Documentation

Content

Model-based Requirements Documentation Techniques

Modelling Requirements in the Three Perspectives

- Different perspective → Different models

- **Data perspective**
  - Entity-relationship diagrams
  - UML class diagrams
- **Functional perspective**
  - Data flow diagrams
  - UML activity diagrams
- **Behavioral perspective**
  - Statecharts
  - UML state machine diagrams

Model-based Requirements Documentation Techniques

Data Perspective – Entity-relationship Diagrams

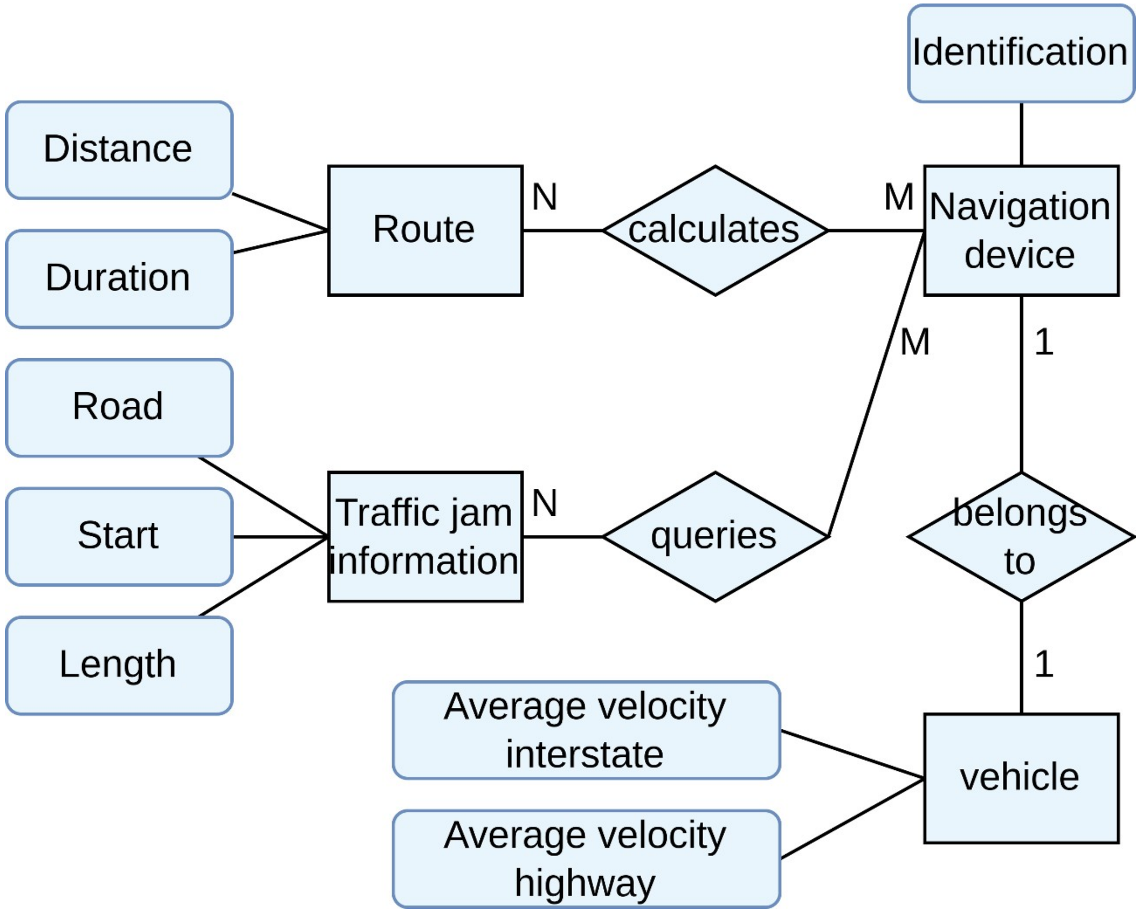
- Concept from the world of databases
- Used to model data (entities) and their relationships

- Extensions of entity-relationship diagrams developed over the years
  - Min/max notations for cardinalities
  - Inheritance mechanism
  - ...

- (Extensions out of scope in this lecture)

Model-based Requirements Documentation Techniques

Data Perspective – Entity-relationship Diagrams (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

Model-based Requirements Documentation Techniques

Data Perspective – UML Class Diagrams

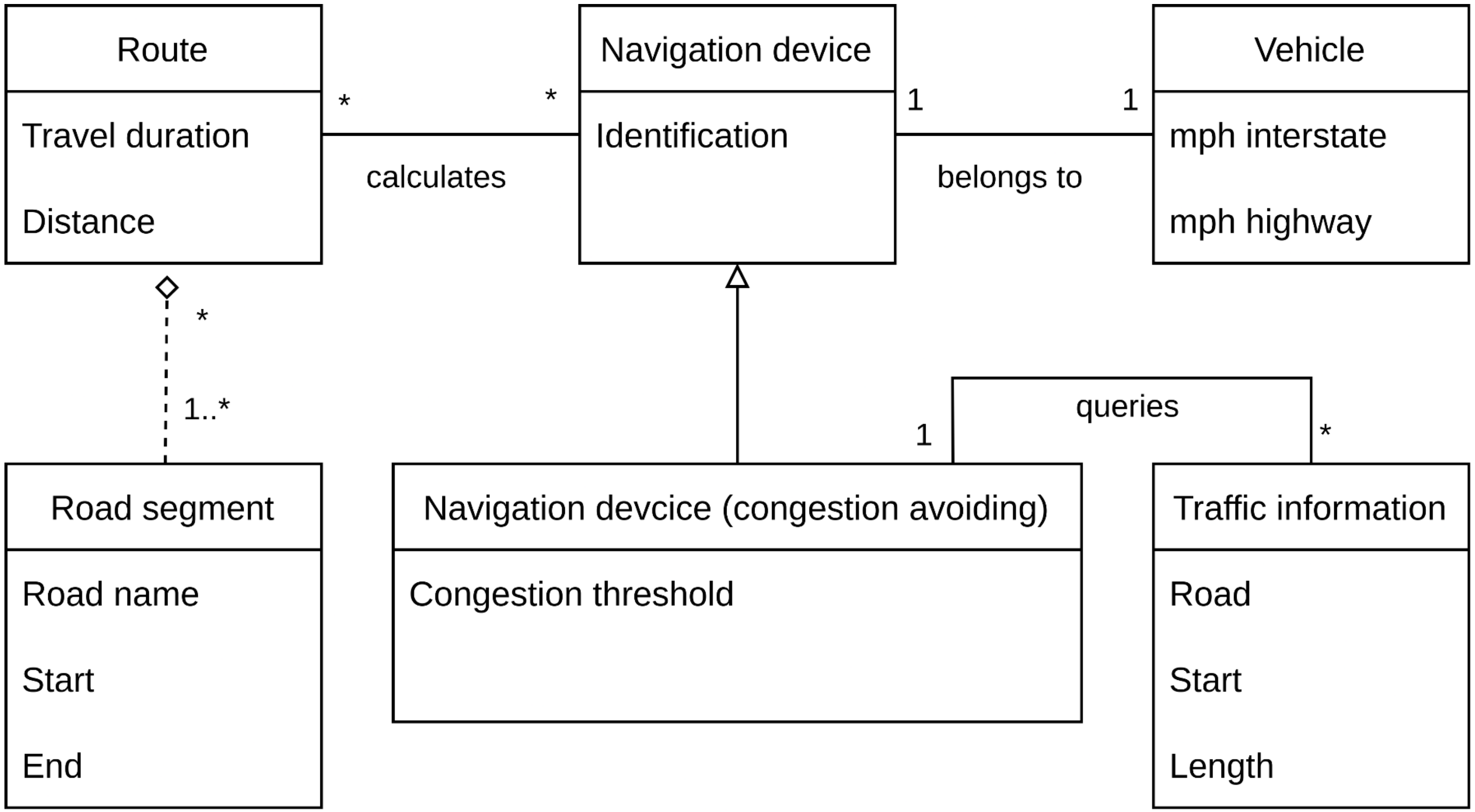
- Consists of classes and their associations

- In principle, similar to entity-relationship diagrams
  - Classes ~ entity types
  - Associations ~ relation types

- Class diagrams more powerful than entity-relationship diagrams

Model-based Requirements Documentation Techniques

Data Perspective – UML Class Diagrams (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

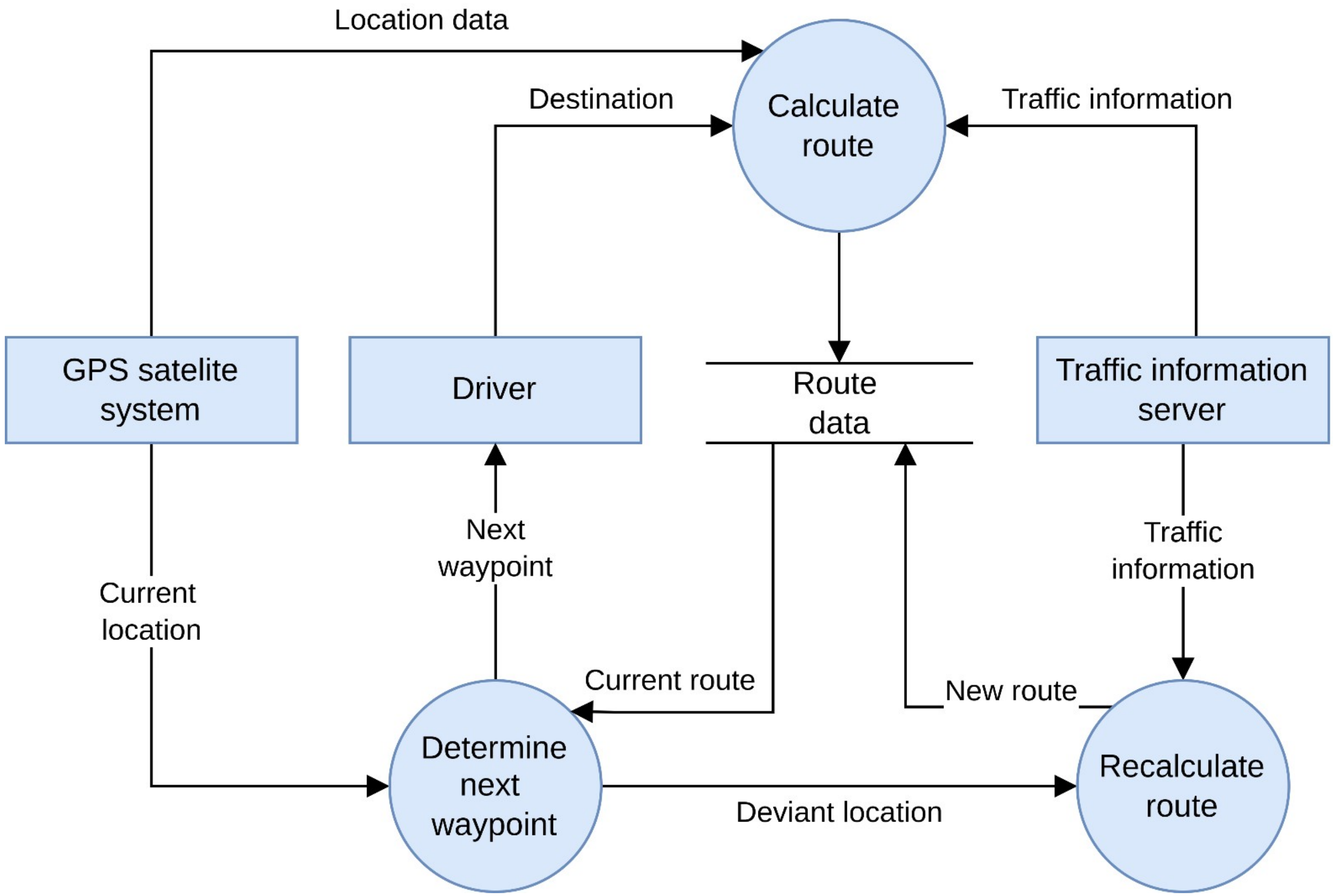
Model-based Requirements Documentation Techniques

Functional Perspective – Data Flow Diagrams

- Model the flow of the data through the system
  - Input/Output data
  - Recipients of the data
- Can be applied on different levels of abstraction
  - Requirements on different levels of abstraction possible

Model-based Requirements Documentation Techniques

Functional Perspective – Data Flow Diagrams (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

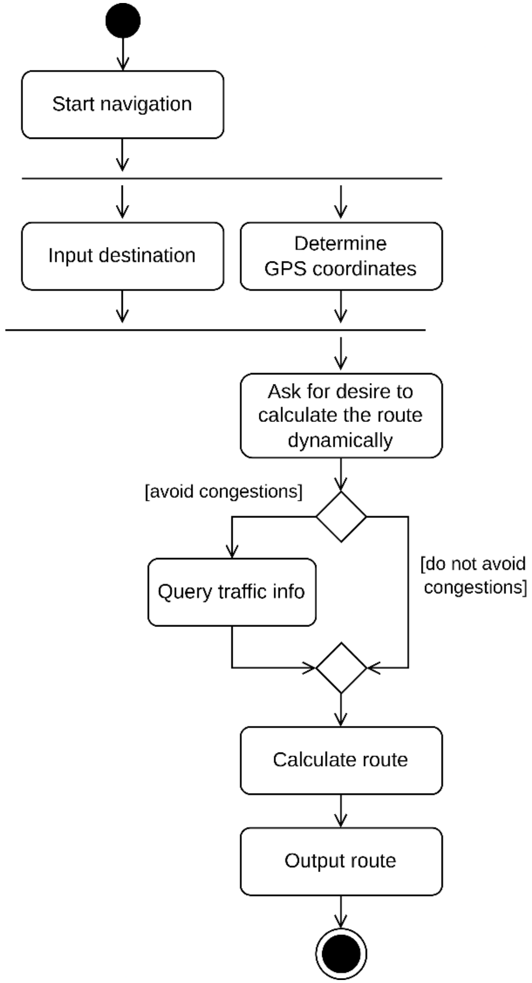
Model-based Requirements Documentation Techniques

Functional Perspective – UML Activity Diagrams

- Method to model action sequences
- Depict the control flow between activities and actions
- Can include the data flow (optional!)

Model-based Requirements Documentation Techniques

Functional Perspective – UML Activity Diagrams (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

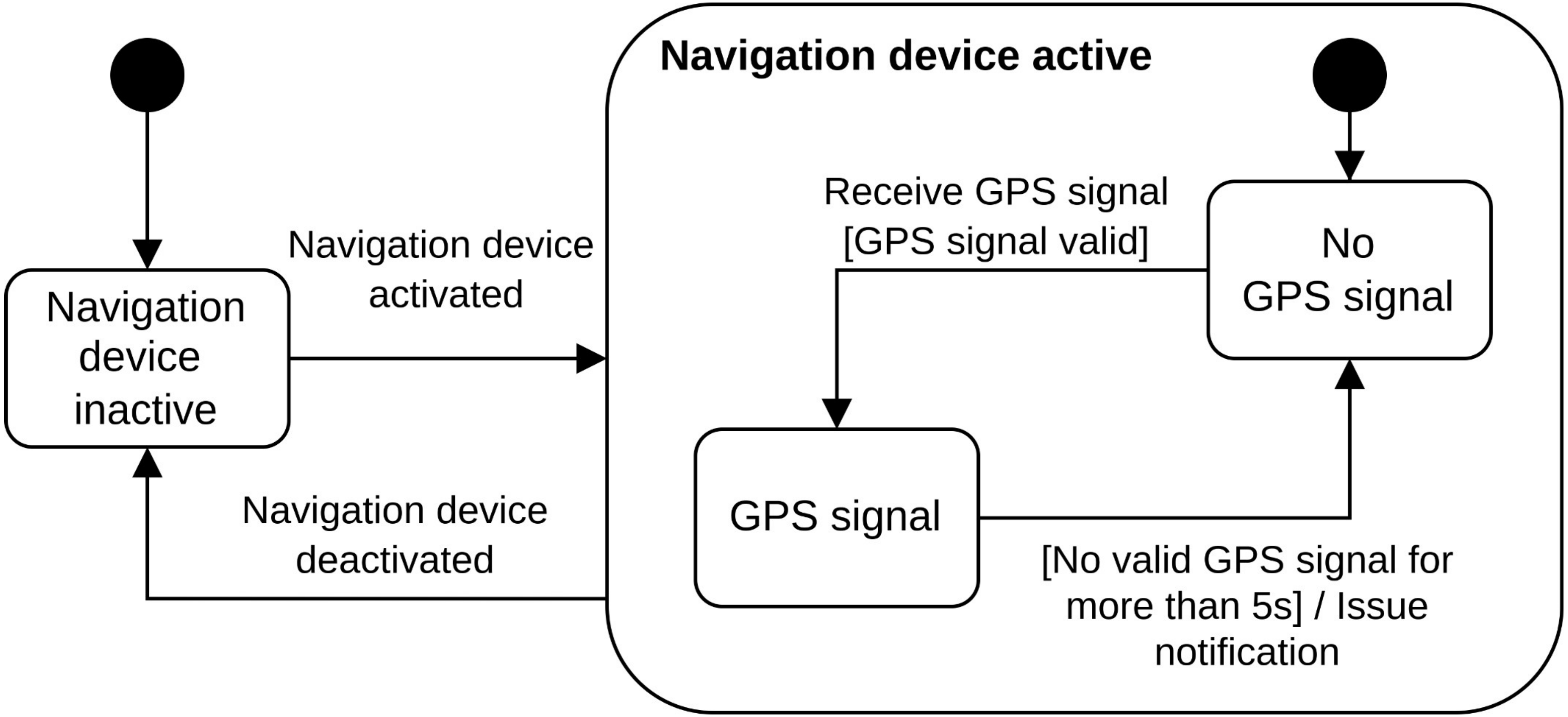
Model-based Requirements Documentation Techniques

Behavioural Perspective – Statecharts

- Extension of finite automata
- Support hierarchization of states
- Allow concurrent behavior

Model-based Requirements Documentation Techniques

Behavioural Perspective – Statechart (Example)



Based on K. Pohl, C. Rupp (2011) – Requirements Engineering Fundamentals: A study guide for Requirements Engineering Foundation Level

FORMAL REQUIREMENTS SPECIFICATION

Lecture 4: Requirements Documentation

Content

1. Model-based Requirements Documentation Techniques
2. Formal Specification Techniques
    1. **Coloured Petri Nets (CPNs)**
    2. AOM → CPN Mapping
    3. Four Variable Model
    4. NRL / SCR

Formal Specification Techniques

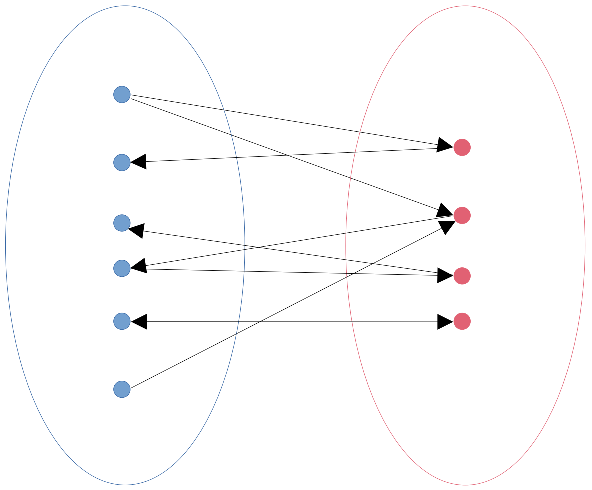
Coloured Petri Nets – Motivation: Why CPNs?

- They can model concurrency and communication in complex systems very well.
- They combine concepts from Petri Nets and programming languages (CPN ML).
- They are Formal (syntatically and mathematically defined)
- They are *executable*.
- They can be derived from other models, such as AOM.
- **Allows for easy manual or automatic system verification and evaluation.**

Formal Specification Techniques

Coloured Petri Nets – What are CPNs?

- Directed bipartite graphs:
- Bipartite graphs divide the vertices of the graph into:
- two disjoint and independent sets.
- Edges in the graph ONLY connect vertices from one set to the other.



Formal Specification Techniques

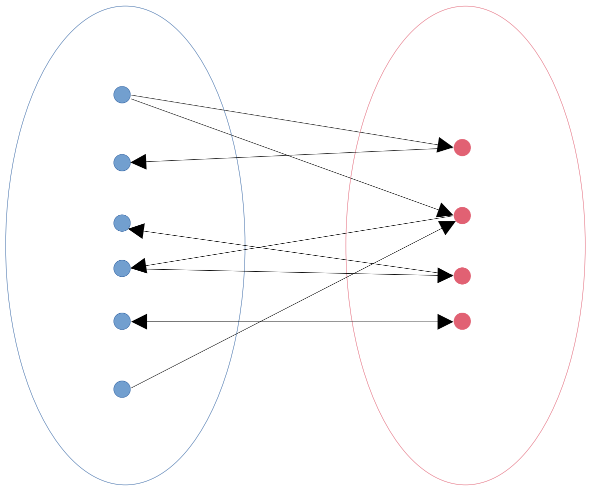
Coloured Petri Nets – What are CPNs?

Directed bipartite graphs:

- CPNs divide the vertices of the graph into:

Places Transitions

- *Arcs* in the graph **ONLY** connect Places with Transitions.

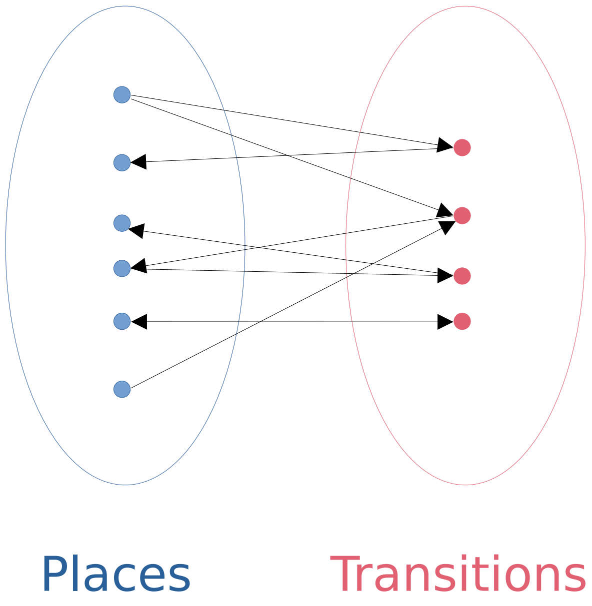


Places Transitions

Formal Specification Techniques

Coloured Petri Nets – What are CPNs?

- CPNs simulate the *change of state* in the system with the exchange of Tokens.
- Tokens reside in, and flow between Places though transitions.
- Tokens never *stay* in Transitions.



Formal Specification Techniques

Coloured Petri Nets – Core Concepts : ColorSets, Tokens and Variables

- Places can hold tokens of only one *type*(= ColorSet).
- Tokens are *instances* of ColorSets.
- Variables are used to distinguish tokens while in transit.

```
var a: INT;
var a2:
INT;
```

Formal Specification Techniques

Coloured Petri Nets – Core Concepts : Markings, Arc Inscriptions and Guards

- Markings specify the tokens held by a Place.
- Arc Inscriptions are expressions that can *modify* the tokens when the transition occurs.
- Guards are a comma-separated list of conditions that must be satisfied for a Transition.

```
var a: INT;
var a2:
INT;
```

Formal Specification Techniques

Coloured Petri Nets – Core Concepts : Flow of Tokens

- A Transition is enabled if and only if:
  - All it's incoming arcs have at least one token.
  - All guard conditions are satisfied.
  - No place that is connected with an inhibitor arc has any tokens.

```
var a: INT;
var a2:
INT;
```

Formal Specification Techniques

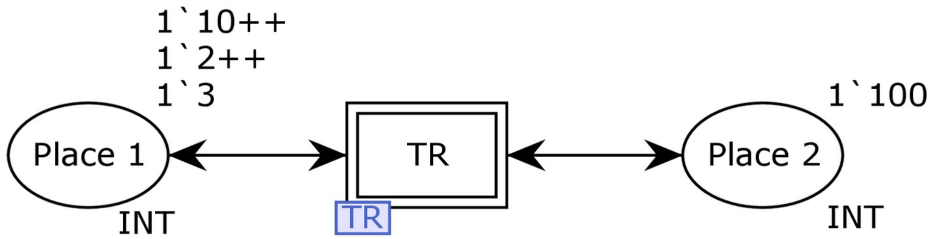
Coloured Petri Nets – Core Concepts : Flow of Tokens

- When a Transition is fired:
  - Any one suitable token is consumed from each input place, according to outgoing arc inscriptions.
  - The tokens are transferred to the output places according to outgoing arc inscriptions.
- **IMP:** Only one transition can be fired at a time.

```
var a: INT;
var a2:
INT;
```

Formal Specification Techniques

Coloured Petri Nets – Core Concepts : Hierarchical CPNs




```
var a: INT;
var a2:
INT;
```

Formal Specification Techniques

Coloured Petri Nets – Evaluation using state-space simulations

Basic Idea

- Compute all the reachable states and the state changes of the CPN model
- Represent these as a directed graph where nodes represent states and arcs represent occurring events
- Pros
- From a constructed state–space, it is possible to verify various aspects of the behaviour of the system:
  - Absence of deadlocks
  - Possibility of always being able to reach a given state
  - Guaranteed delivery of a given service
- Cons
  - State-space graph size (and computation time) increases exponentially! → requires independent computation in some cases.

K. Jensen et al. (2007) – Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (9): 213-254. DOI:10.1007/s10009-007-0038-x

Formal Specification Techniques

Coloured Petri Nets – CPN Tools

“A tool for editing, simulating, and analysing Coloured Petri Nets”

cpntools.org

Lecture 4: Requirements Documentation

Content

AOM → CPN Mapping

Motivation and Methodology

- CPN models are powerful, *BUT* can be arbitrarily complex. Where to start?
- It is important to ensure inter-model consistancy.
- Mapping AOM → CPN models leverages the advantages of both, and also creates a feedback loop using the simulation and evaluation capabilities of CPNs.

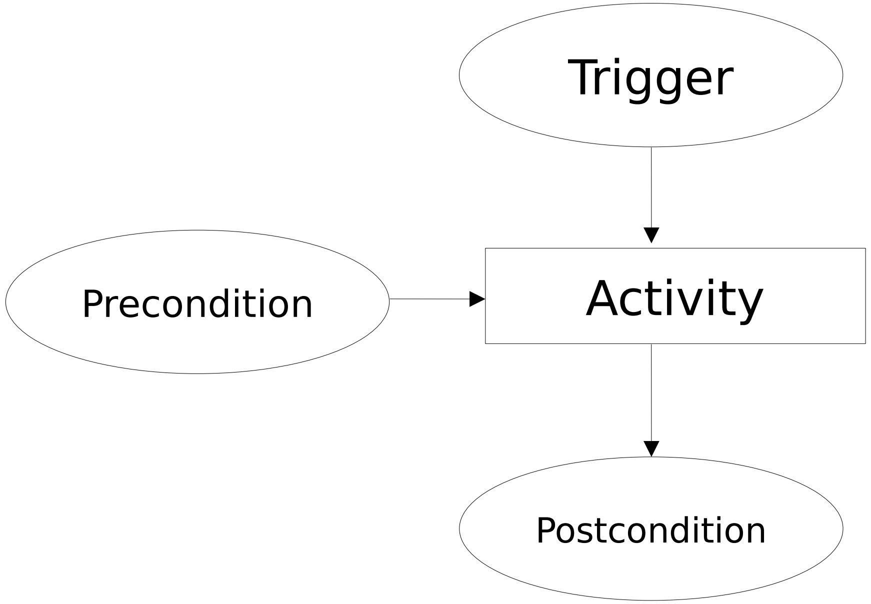
- *Heuristics for Designing and Evaluating Socio-Technical Agent-Oriented Behaviour Models with Coloured Petri Nets (2014).* Msury Mahunnah, Alex Norta, Lixin Ma, Kuldar Taveter

M. Mahunnah et al. (2014) – Heuristics for Designing and Evaluating Socio-technical Agent-Oriented Behaviour Models with Coloured Petri Nets. IEEE 38th International Computer Software and Applications Conference Workshops (2014): 438-443. https://doi.org/10.1109/COMPSACW.2014.74

AOM → CPN Mapping

Mapping Methodology

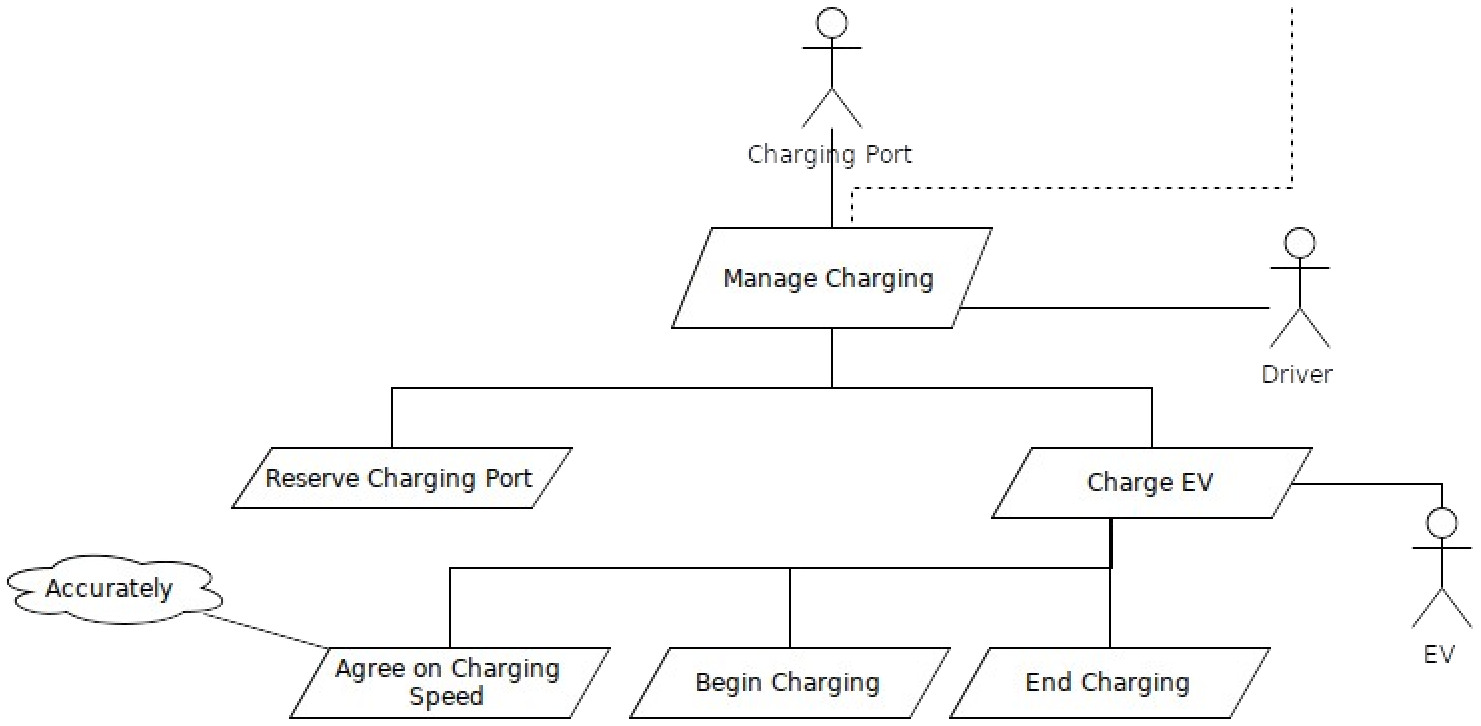
Notation	Name
	Connecting arc
	Sub-goal or activity
	Trigger/ precondition
	Postcondition
	Goal
[<condition(s)>]	Precondition(s)



Adapted from M. Mahunnah et al. (2014) – Heuristics for Designing and Evaluating Socio-technical Agent-Oriented Behaviour Models with Coloured Petri Nets. IEEE 38th International Computer Software and Applications Conference Workshops (2014): 438-443. https://doi.org/10.1109/COMPSACW.2014.74

AOM → CPN Mapping

Example: Automated EV Charging Station (Manage Charging)



AOM → CPN Mapping

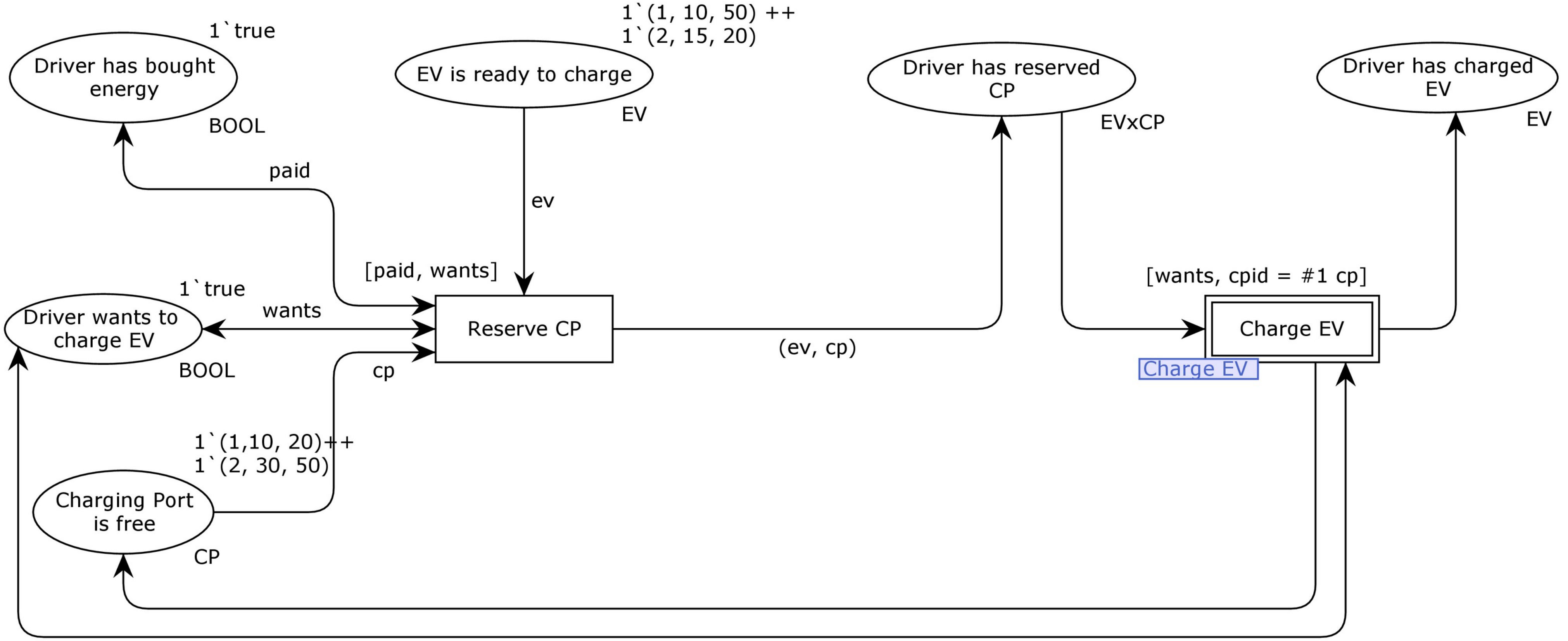
Example: Automated EV Charging Station (Manage Charging)

Activity	Trigger(s)	Precondition(s)	Postcondition(s)
Reserve Charging Port(CP)	Driver wants to charge EV	Driver has bought energy, CP is free, EV is ready to charge	Driver has reserved CP
Charge EV	〃	Driver has reserved CP	Driver has charged EV
Agree on charging speed	〃	Driver has reserved CP, Max CP speed ≥ Min EV speed, Max EV speed ≥ Min CP speed	Charging speed is agreed upon
Begin Charging	〃	Charging speed is agreed upon	EV has begun charging
End Charging	None	EV has competed charging	Driver has charged EV, CP is free

AOM → CPN Mapping

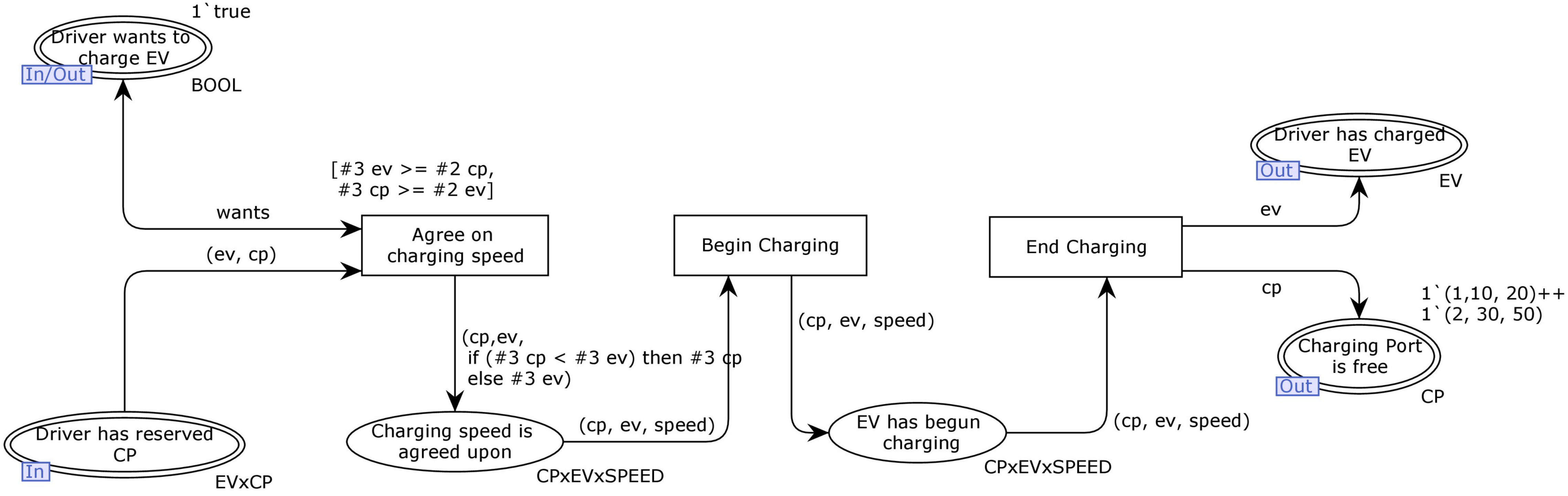
Example: Automated EV Charging Station (Manage Charging)

Here we assume some connection to the Buy Energy Sub-Goal that gives a true/false response indicating whether a driver has paid or not.



AOM → CPN Mapping

Example: Automated EV Charging Station (Charge EV)



AOM → CPN Mapping

State-space Simulations

Basic Idea

- Compute all the reachable states and the state changes of the CPN model
- Represent these as a directed graph where nodes represent states and arcs represent occurring events

Pros

- From a constructed state–space, it is possible to verify various aspects of the behaviour of the system:
  - Absence of deadlocks
  - Possibility of always being able to reach a given state
  - Guaranteed delivery of a given service

Cons

  - State-space graph size (and computation time) increases exponentially! → requires independent computation in some cases.

K. Jensen et al. (2007) – Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (9): 213-254. DOI:10.1007/s10009-007-0038-x

Lecture 4: Requirements Documentation

Content

Formal Specification Techniques

Four Variable Model – Basic Elements

- Four Variables
  - Input variables → Physical variables measured by input devices
  - Output variables → Physical variables controlled by output devices
  - Monitored environmental variables
  - Controlled environmental variables

- Relations
  - NAT, REQ, IN/OUT, SOF, SOFREQ
- Are used as basis for requirements documentation

Formal Specification Techniques

Four Variable Model – Documents

- System Requirements Document
  - Models the complete system as a black-box
  - Includes a description of the environment
  - Identifies a set of quantities and associates each one with a mathematical variable
  - Describes constraints of the environment like physical laws
  - Describes constraints related to the new system

Formal Specification Techniques

Four Variable Model – Documents

- System Design Document
  - Describes relevant properties of peripheral devices
  - Identifies input- and output registers, modeled as mathematical variables
  - Describes relation between input registers and associated environmental quantities
  - Describes relation between output registers and associated environmental quantities

Formal Specification Techniques

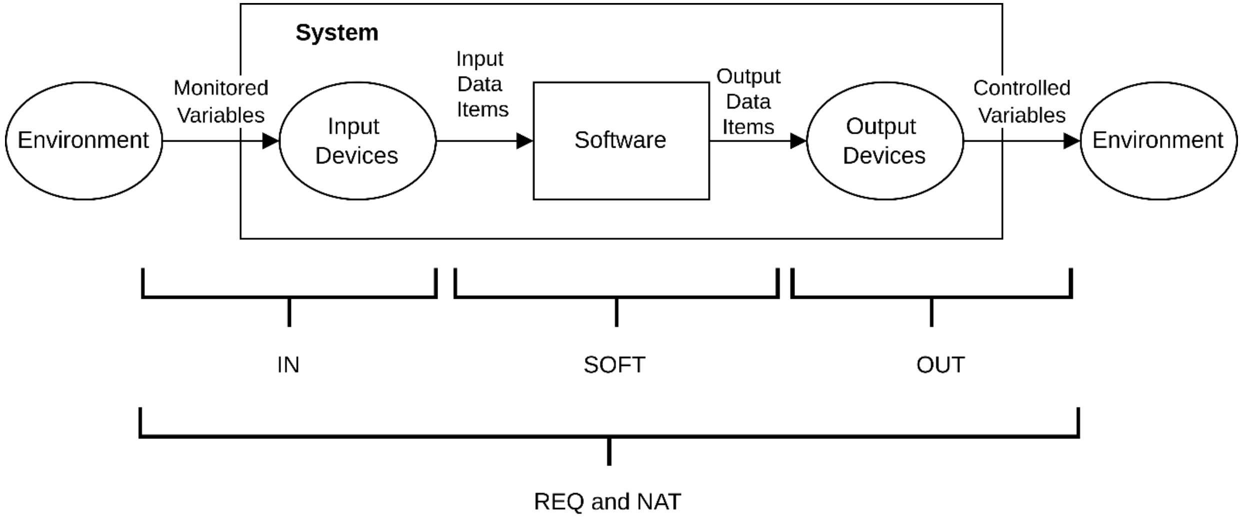
Four Variable Model – Documents

- Software Requirements Document
  - Combination of system requirements document and system design document
- Software Behavior Specification
  - Records additional design decisions
  - Provides a description of the actual software behavior

Formal Specification Techniques

Four Variable Model – Relations

- *NAT*: expresses constraints due to restrictions imposed by nature
- *REQ*: expresses the requirements of the system
- *IN, OUT*: input and output relations
- *SOF*: behavior of a particular software implementation
- *SOFREQ*: software requirements relation, all acceptable software behavior

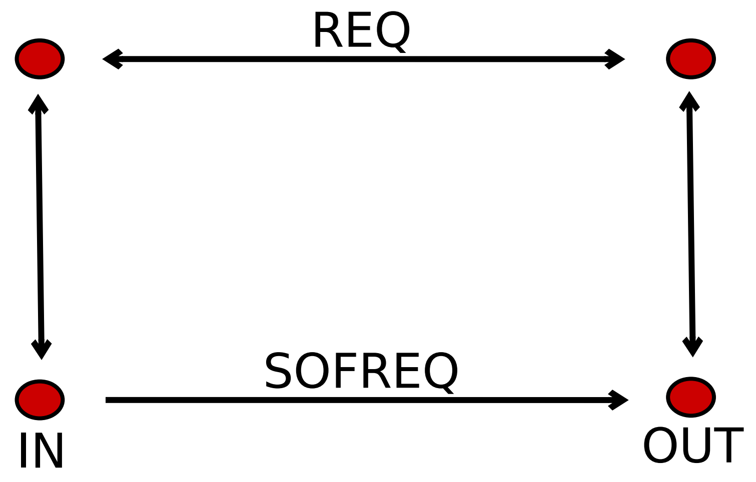


Based on C.L. Heitmeyer, R.D. Jeffords, B.G. Labaw (2013) – Automated Consistency Checking Requirements Specifications

Formal Specification Techniques

Four Variable Model – Basic Relations

- Basic Relations
- SOFREQ ⊆ SOF
- IN, OUT, REQ $\xrightarrow[\text{w.r.t. NAT}]{}$ SOFREQ
- OUT ○ SOFREQ ○ IN = REQ



Formal Specification Techniques

Four Variable Model – Variables

- Monitored variables
  - Mathematical function whose domain consists of real numbers
  - Environmental function $m^t : \mathbb{R} \rightarrow Value$
  - Value at time s: $m^t(s)$
  - Vector of monitored variables: $m^t = (m^t_1, m^t_2, \ldots, m^t_p)$

- Controlled variables
  - Mathematical function whose domain consists of real numbers
  - Environmental function $c^t : \mathbb{R} \rightarrow Value$
  - Vector of monitored variables: $c^t = (c^t_1, c^t_2, \ldots, c^t_q)$

Formal Specification Techniques

Four Variable Model – Relations

- REQ
  - Expresses the requirements of the system
  - *domain(REQ)*: set of vectors containing exactly the instances of m^t allowed by the environmental constraints
  - *range(REQ)*: set of vectors containing only those instances of c^t considered permissible
  - $(m^t, c^t) \in REQ$ if environmental constraints allow the controlled variables to take the values given by c^t, if the values of the monitored variables are given by m^t
  - *REQ* may **tolerate 'small' errors** in the values of controlled variables

Formal Specification Techniques

Four Variable Model – Relations

- IN
  - Describes the behavior of the input devices
  - Is a relation due to imprecision in the measurement

- OUT
  - Describes the behavior of output devices
  - Is a relation due to device imperfections

Formal Specification Techniques

Four Variable Model – Relations

- SOF
  - Describes behavior of a particular software implementation
  - *domain(SOF)*: set of vectors containing all possible instances of i^t
  - *range(SOF)*: set of vectors containing all possible instances of o^t
  - $(i^t, o^t) \in SOF$ iff the software could produce values described by o^t
  - **Verification condition:** *SOF* implements a subset of *REQ*
    - $SOF \subseteq IN^{-1} \circ REQ \circ OUT^{-1}$
    - (where $\circ$ denotes the composition of binary relations)
    - Often: $SOF = IN^{-1} \circ REQ \circ OUT^{-1}$

Formal Specification Techniques

Four Variable Model – Relations

- SOFREQ
  - Characterizes all acceptable software behavior
  - SOFREQ corresponds on the software level to REQ on the system level
  - It consists of all tuples (i^t, o^t) satisfying for all m^t, c^t :
    - $(IN(m^t, i^t) \wedge OUT(o^t, c^t) \wedge NAT(m^t, c^t)) \rightarrow REQ(m^t, c^t)$

Formal Specification Techniques

Four Variable Model – Feasibility and Acceptability

- Feasibility of REQ w.r.t. NAT
  - Requirements should specify behavior for all cases that can arise
  - $domain(NAT) \subseteq domain(REQ)$
  - $domain(NAT \cap REQ) = domain(NAT) \cap domain(REQ) = domain(NAT)$

- Acceptability
  - Describes the behavior that the software must exhibit to be acceptable for use and for the requirements to be satisfied
  - $NAT \cap (IN \circ SOF \circ OUT) \subseteq REQ$

Lecture 4: Requirements Documentation

Content

Formal Specification Techniques

NRL / SCR – Motivation

- Tabular notations → precise and compact notation for requirements
- Applications → avionics systems, controlling nuclear power plants, telephone networks
- Can be used for automatic analysis
- History
  - 1978: flight program of the A-7 aircraft
  - Real-time, embedded system
  - Organizations: Bell Laboratories, Ontario Hydro, Naval Research Laboratory, Lockheed
  - Applications: submarine communication system, shutdown system for the Darlington nuclear power plant, flight program for Lockheed’s C130J aircraft.

Formal Specification Techniques

NRL / SCR – SCR Formal Model

- Behavior
  - Non-deterministic system environment
  - Deterministic system behavior
- 4VM → Monitored and controlled variables, NAT, REQ
- Model: System is represented as labeled transition system (LTS), responds to each monitored event
- Synchronous behavior: The system completely processes one set of inputs before processing the next state
- One input assumption: At most one monitored variable is allowed to change from one state to the next

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Formal Model

- Auxiliary variables: (specification of *REQ*)
  - Mode classes: values are called modes
  - Modes: equivalence class of system states
  - Terms: internal variables

- System: labeled transition system $(S, I, E^m, \rightarrow)$ consisting of
  - States S, initial states I
  - Labels E^m (set of monitored events)
  - Transition relation $\rightarrow$ realized as a function that maps a monitored event $e \in E^m$ and the current state $s \in S$ to the next state $s' \in S$

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Formal Model

- Condition → Predicate defined on a single system state
- Event → Predicate defined on two system states
- Occurrence → An event occurs if a condition changes
- *@T(c)*: *c* becomes *true*
- *@F(c)*: *c* becomes *false*
- Conditioned event → $@T(c_1)\ WHEN\ c_2$

Special form: $@T(c)\ WHEN\ d$ *iff* $\neg c \land c' \land d$

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Tables

- Mode transition table:
  - Associates a source mode and an event with a destination mode
  - Each table should describe a **total function**
  - Should exhibit disjointness and coverage properties
- Event table:
  - An event table defines how a term or controlled variable changes in response to input events
  - Defines a (partial) function from modes and events to variable values
- Condition table:
  - A condition table defines the value of a term or controlled variable under every possible condition
  - Defines a total function from modes and conditions to variable values

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Tables

Current Mode	Powered on	Too Cold	Temp OK	Too Hot	New Mode
Off	@T @T @T	- t -	t - -	- - t	**Inactive** **Heat** **AC**
Inactive	@F - -	- @T -	- - -	- - @	**Off** **Heat** **AC**
Heat	@F -	- -	- @T	- -	**Off** **Inactive**
AC	@F -	- -	- @T	- -	**Off** **Inactive**

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Tables

Event table

Modes		
NoFailure	@T(INMODE)	Never
ACFailure, HeatFailure	Never	@T(INMODE)
Warning light =	**Off**	**On**

Condition table

Modes	Events	
NoFailure	true	false
ACFailure	temp > temp0	temp <= temp0
HeatFailure	false	waterlevel = low
Warning light =	**Off**	**On**

Formal Specification Techniques / NRL / SCR

NRL / SCR – Tool Support: SCR Toolset

- Specification editor for creating the tabular specification
- Simulator for validation
- Dependency graph browser for understanding the relationship between different parts of the specification
- Consistency checker for analyzing syntax, type correctness, determinism, case coverage, ...
- Model checker for checking linear temporal properties of finite state systems
- Theorem prover for checking properties deductively, avoiding the state explosion problem, often user interaction necessary

Formal Specification Techniques / NRL / SCR

NRL / SCR – SCR Toolset: Construction of Requirements Specifications

- Specification
  - Specification editor
  - Function tables: define the value of dependent variables
  - Dictionaries: variable declarations, environmental assumptions, type definitions

- Analysis
  - Consistency checker, property checker, dependency graph browser
  - Well-formedness errors
  - Disjointness and coverage: no nondeterminism, no missing cases

Formal Specification Techniques / NRL / SCR
NRL / SCR – SCR Toolset: Construction of Requirements Specifications

- Validation
  - Check inconsistencies between the intended and the specified behavior
  - Simulator: the user can run scenarios (sequences of monitored events)
  - Invariant generator: the user can generate state invariants

- Application analysis
  - Check application properties
  - Model checker
  - Property checker
  - Theorem prover (TAME, PVS)

SUMMARY

Summary

- Conceptual models as a means for requirements documentation
  - Abstraction and good overview vs. learning a modeling language
  - Different models for different purposes → Model needs to fit the purpose
- UML provides models for almost anything
  - We only covered a small part → Other UML models can also be useful for requirements documentation
  - UML is not the only answer → Other models work fine, too.
- Formal specification techniques
  - Coloured Petri Nets (incl. mapping from AOM), four variable model, NRL/SCR
  - More exist and might be better suited for your project

Questions?