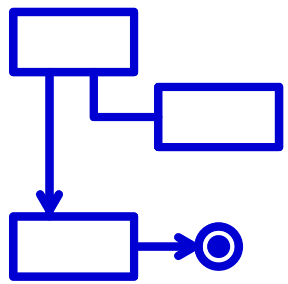
Requirements Engineering I

Chapter 7

Model-Based Requirements Specification



Chapter roadmap

7.1 Models in RE
The what and the why

7.2 Modeling structure and data
The static stuff

7.3 Modeling function and flow
Activities, flows, and processes

7.4 Modeling state and behavior
System dynamics

7.5 Modeling context and boundary
Context & context–system interaction

7.6 Modeling goals
Reflecting about rationale

7.7 UML
The most widely used modeling language

7.8 Lightweight, flexible modeling
The lightweight alternative

Motivation

Why do we model requirements?

- Gain an overview of a set of requirements
- Understand relationships and inter-connections between requirements
- Focus on some aspect of a system, abstracting from the rest

Primarily for functional requirements

Quality requirements and constraints are mostly specified in natural language

7.1 Models in RE

DEFINITION. Model – an abstract representation of an existing part of reality or a part of reality to be created.

The notion of reality includes any conceivable set of elements, phenomena or concepts, including other models.

With respect to a model, the modeled part of reality is called the original.

[Glinz 2024]

- Requirements models are problem-oriented models of the system to be built
- Architecture and design information is omitted

Requirements models can be used for

- Specifying requirements (as a means of replacing textually represented requirements)
- Paraphrasing textually represented requirements to improve understanding of complex structures and dependencies
- Testing textually represented requirements to uncover omissions, ambiguities and inconsistencies
- Decomposing a complex reality into comprehensible parts

Which aspects can be modeled?

- Structure and Data
  - Structural properties of a system, particularly of the static data
  - Structure of a system's domain
- Function and Flow

  Sequence of actions and control / data flow for
  - producing a required result
  - describing a (business) process
- State and Behavior

  Behavior of a system or a domain component
  - State-dependent reactions to events
  - Dynamics of component interaction

Which aspects can be modeled? – 2

- Context and boundary
  - Structural embedding of system in its environment
  - Interaction between system and actors in the context
- Goals

  Understanding the goals for a system
  - Goal decomposition
  - Goal-agent networks

7.2 Modeling structure and data

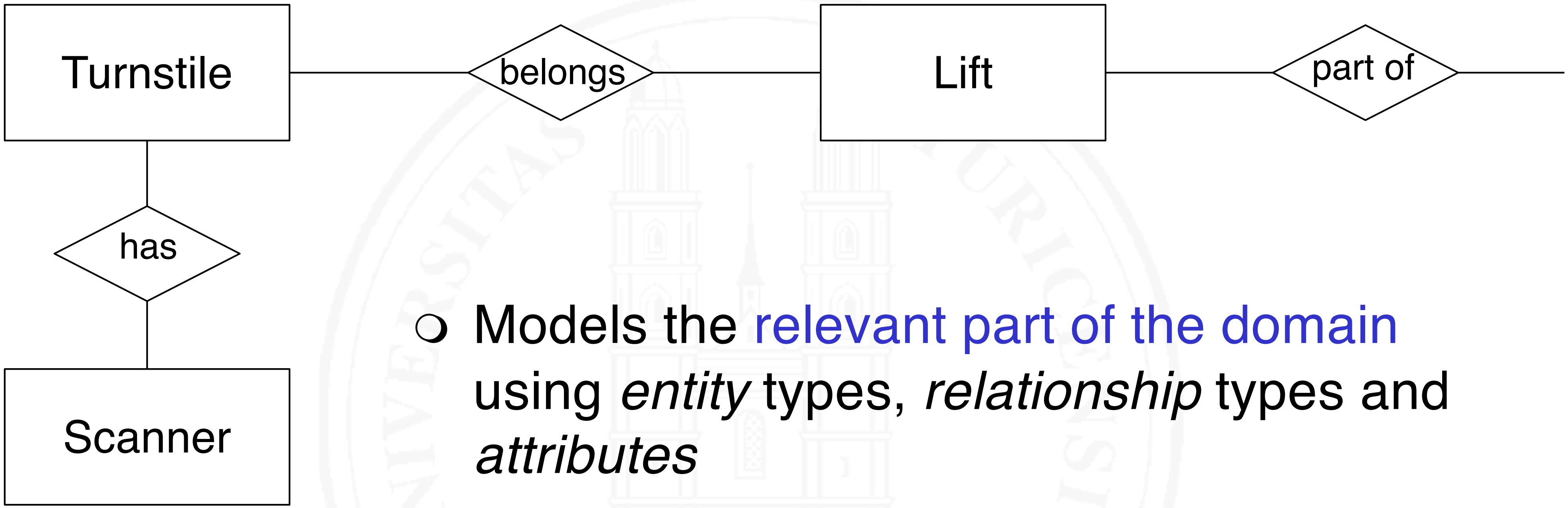
- Entity-relationship models
- Class and object models
- Component models

What to model

- Static system models: Information that a system needs to know and store persistently
- Static domain models: The (business) objects and their relationships in a domain of interest

Data modeling (entity-relationship models)

[Chen 1976]



- ❍ Models the relevant part of the domain using *entity* types, *relationship* types and *attributes*
- \+ Rather easy to model
- \+ Straightforward mapping to relational database systems
- – Ignores functionality and behavior
- – No means for system decomposition

Object and class modeling

[Booch 1986, Booch 1994, Glinz et al. 2002]

Idea

- Identify those entities in the domain that the system has to store and process
- Map this information to objects/classes, attributes, relationships and operations
- Represent requirements in a static structural model
- Modeling individual objects does not work: too specific or unknown at time of specification
  - → *Classify* objects of the same kind to classes: Class models
  - → or select an abstract *representative*: Object models

Terminology

Object – an individual entity which has an identity and does not depend on another entity.

Examples: Turnstile no. 00231, The Plauna chairlift

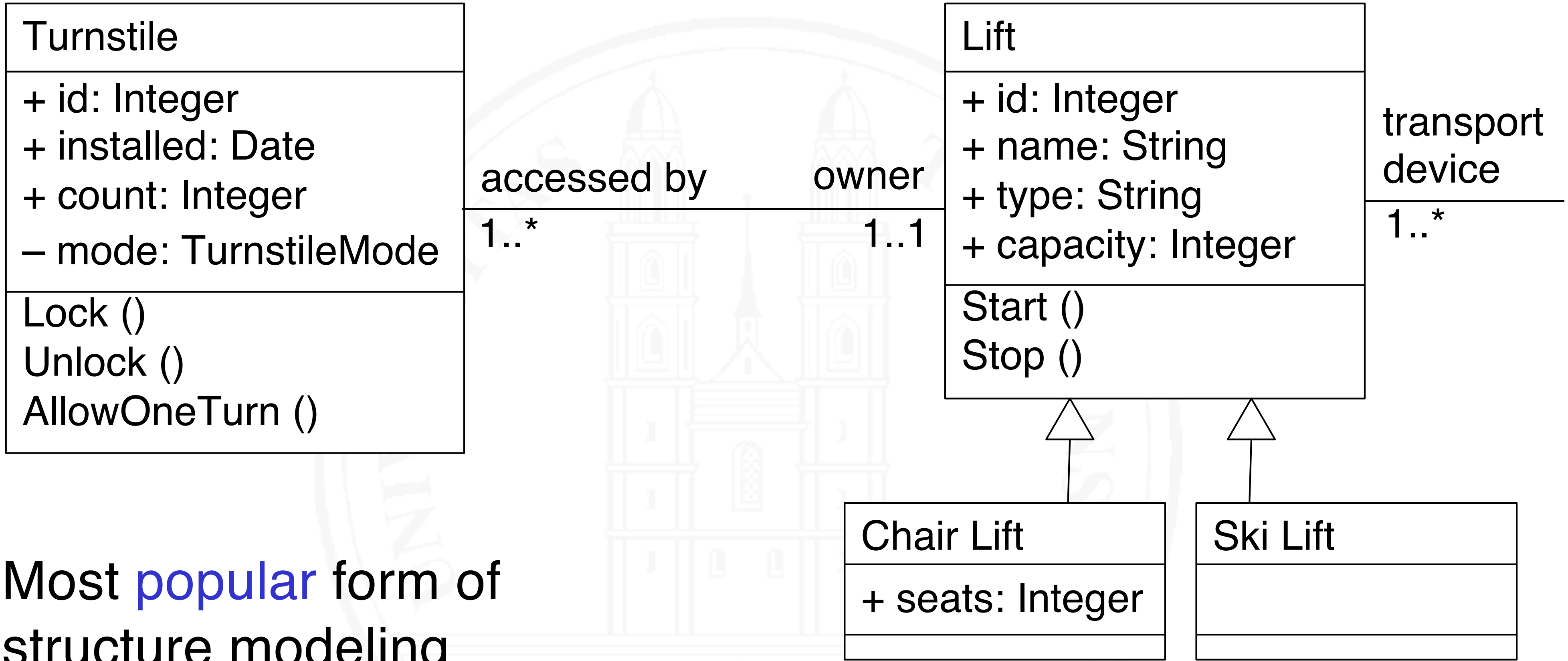
Class – Represents a set of objects of the same kind by describing the structure of the objects, the ways they can be manipulated and how they behave.

Examples: Turnstile, Lift

Abstract Object – an abstract representation of an individual object or of a set of objects having the same type

Example: A Turnstile

Class models / diagrams



Most popular form of structure modeling

Typically using UML class diagrams

Class diagram: a diagrammatic representation of a class model

Mini-Exercise: Class modeling

Assume that in our chairlift access control case study there is a requirement to keep an event log for every lift. Extend the class model from the previous slide such that the following information can be logged: A time stamp, the event name and event type (mode change, warning or failure), and (optionally) an action taken.

Class models are sometimes inadequate

- Class models don't work when different objects of the same class need to be distinguished
- Class models can't be decomposed properly: different objects of the same class may belong to different subsystems
- Subclassing is a workaround, but no proper solution

In such situations, we need object models

Object models: a motivating example

Example: Treating incidents in an emergency command and control system

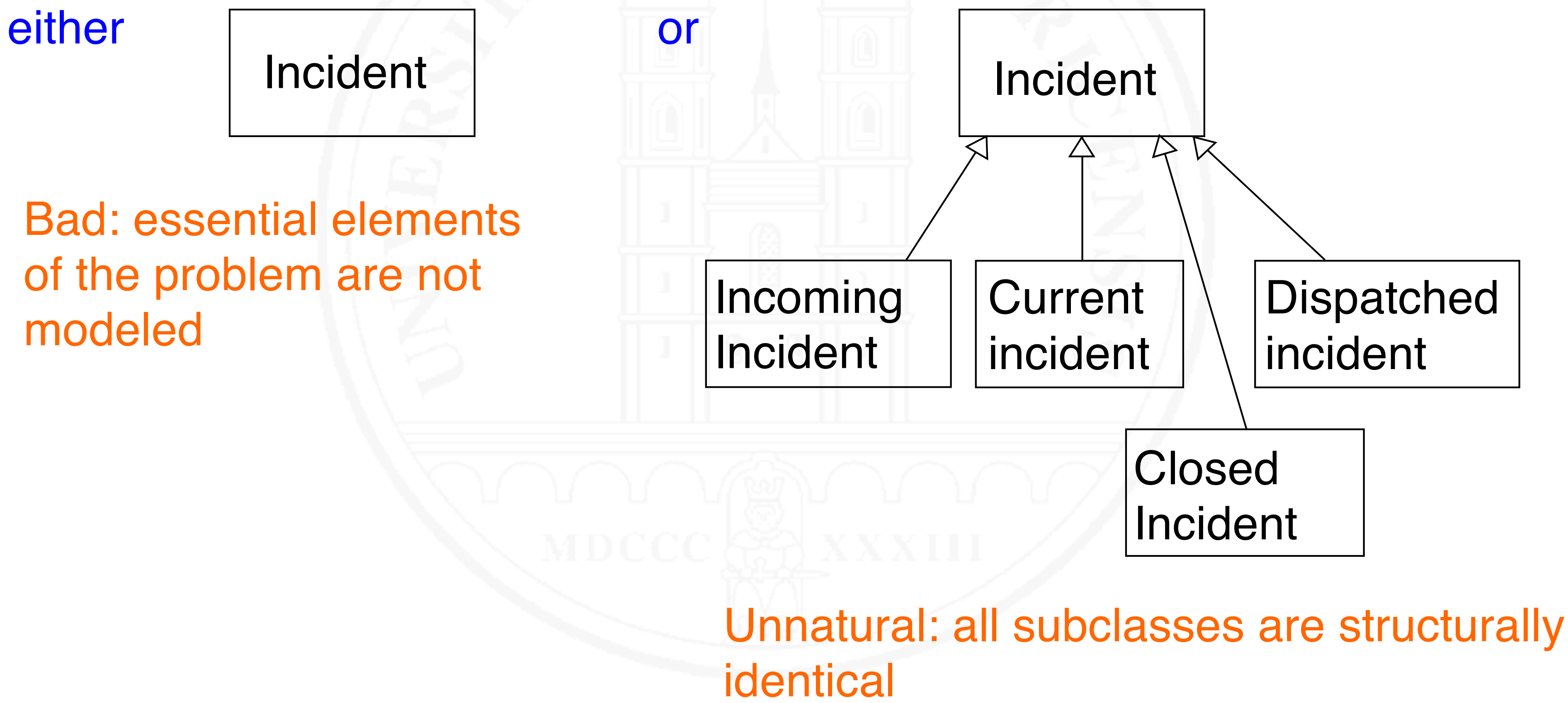
Emergency command and control systems manage incoming emergency calls and support human dispatchers in reacting to incidents (e.g., by sending police, fire fighters or ambulances) and monitoring action progress.

When specifying such a system, we need to model

- Incoming incidents awaiting treatment
- The incident currently managed by the dispatcher
- Incidents currently under treatment
- Closed incidents

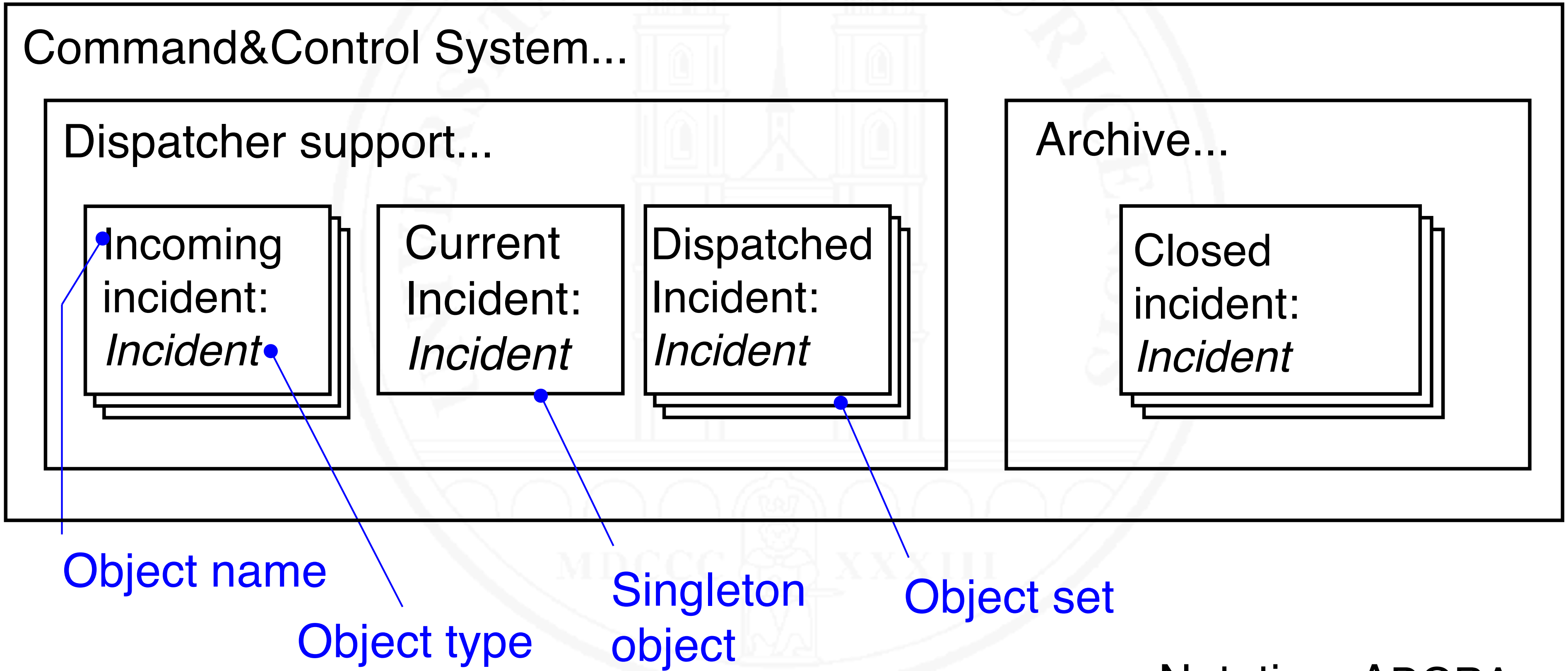
Class models are inadequate here

In a class model, incidents would have to be modeled as follows:



Object models work here

Modeling is based on a hierarchy of abstract objects



Notation: ADORA

ADORA

[Glinz et al. 2002]

- ADORA is a language and tool for object-oriented specification of software-intensive systems
- Basic concepts
  - Modeling with abstract objects
  - Hierarchic decomposition of models
  - Integration of object, behavior and interaction modeling
  - Model visualization in context with generated views
  - Adaptable degree of formality
- Developed in the RERG research group at UZH

Modeling with abstract objects in UML

- Not possible in the original UML (version 1.x)
- Introduced 2004 as an option in UML 2
- Abstract objects are modeled as components in UML
- The component diagram is the corresponding diagram
- Lifelines in UML 2 sequence diagrams are also frequently modeled as abstract objects
- In UML 2, class diagrams still dominate

What can be modeled in class/object models?

- Objects as *classes* or *abstract objects*
- Local properties as *attributes*
- Relationships / non-local properties as *associations*
- Services offered by objects as *operations* on objects or classes (called *features* in UML)
- Object behavior
  - Must be modeled in separate *state machines* in UML
  - Is modeled as an *integral part* of an object hierarchy in ADORA
- System-context interfaces and functionality from a user's perspective *can't* be modeled *adequately*

Object-oriented modeling: pros and cons

+ Well-suited for describing the structure of a system
+ Supports locality of data and encapsulation of properties
+ Supports structure-preserving implementation
+ System decomposition can be modeled
– Ignores functionality and behavior from a user's perspective
– UML class models don't support decomposition
– UML: Behavior modeling weakly integrated

Mini-Exercise: Classes vs. abstract objects

Specify a distributed heating control system for an office building consisting of a central boiler control unit and a room control unit in every office and function room.

- The boiler control unit shall have a control panel consisting of a keyboard, a LCD display and on/off buttons.
- The room control unit shall have a control panel consisting of a LCD display and five buttons: on, off, plus, minus, and enter.

Model this problem using

a. A class model

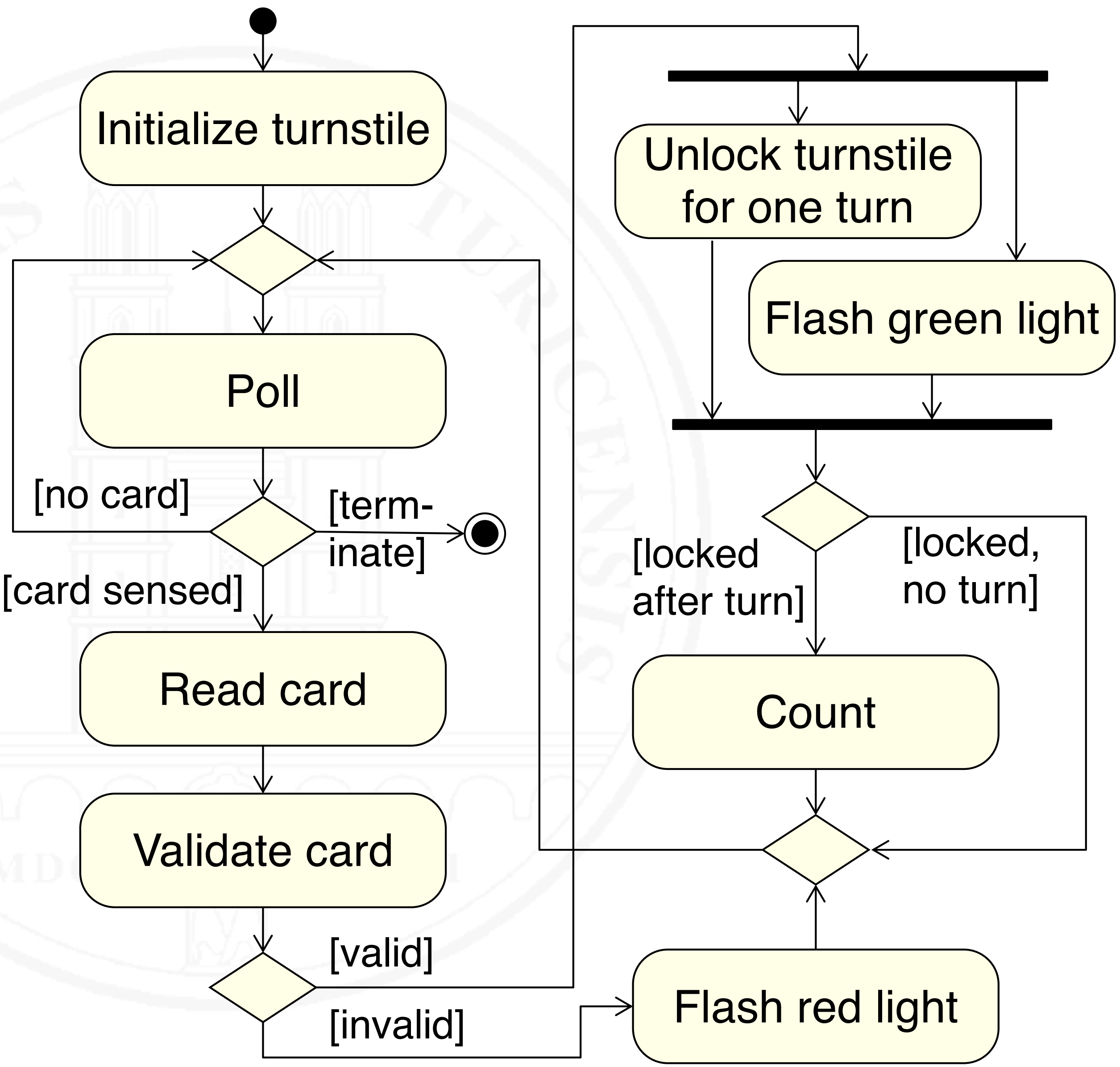
b. An abstract object model.

7.3 Modeling function and flow

- Activity models
- Data flow / information flow models
- Process and work flow models
- Domain story models

Activity modeling: UML activity diagram

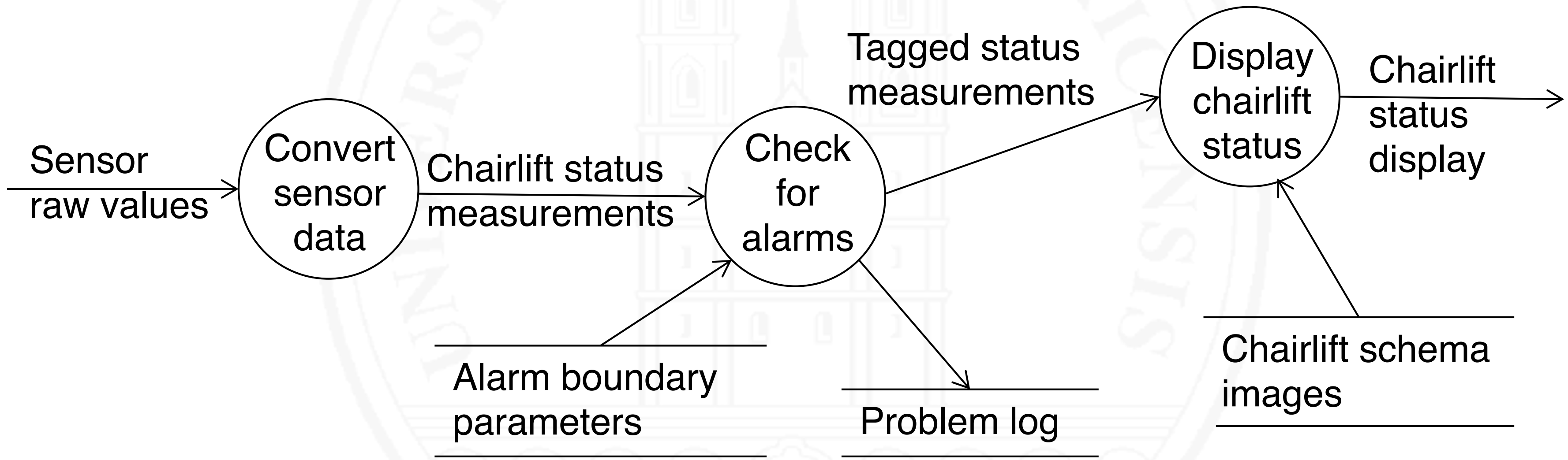
- Models process activities and control flow
- Can model data flow
- Model can be underpinned with execution semantics



Data and information flow

[DeMarco 1978]

- Models system functionality with data flow diagrams
- Once a dominating approach; rarely used today



+ Easy to understand
+ Supports system decomposition
– Treatment of data outdated: no types, no encapsulation

Process and workflow modeling

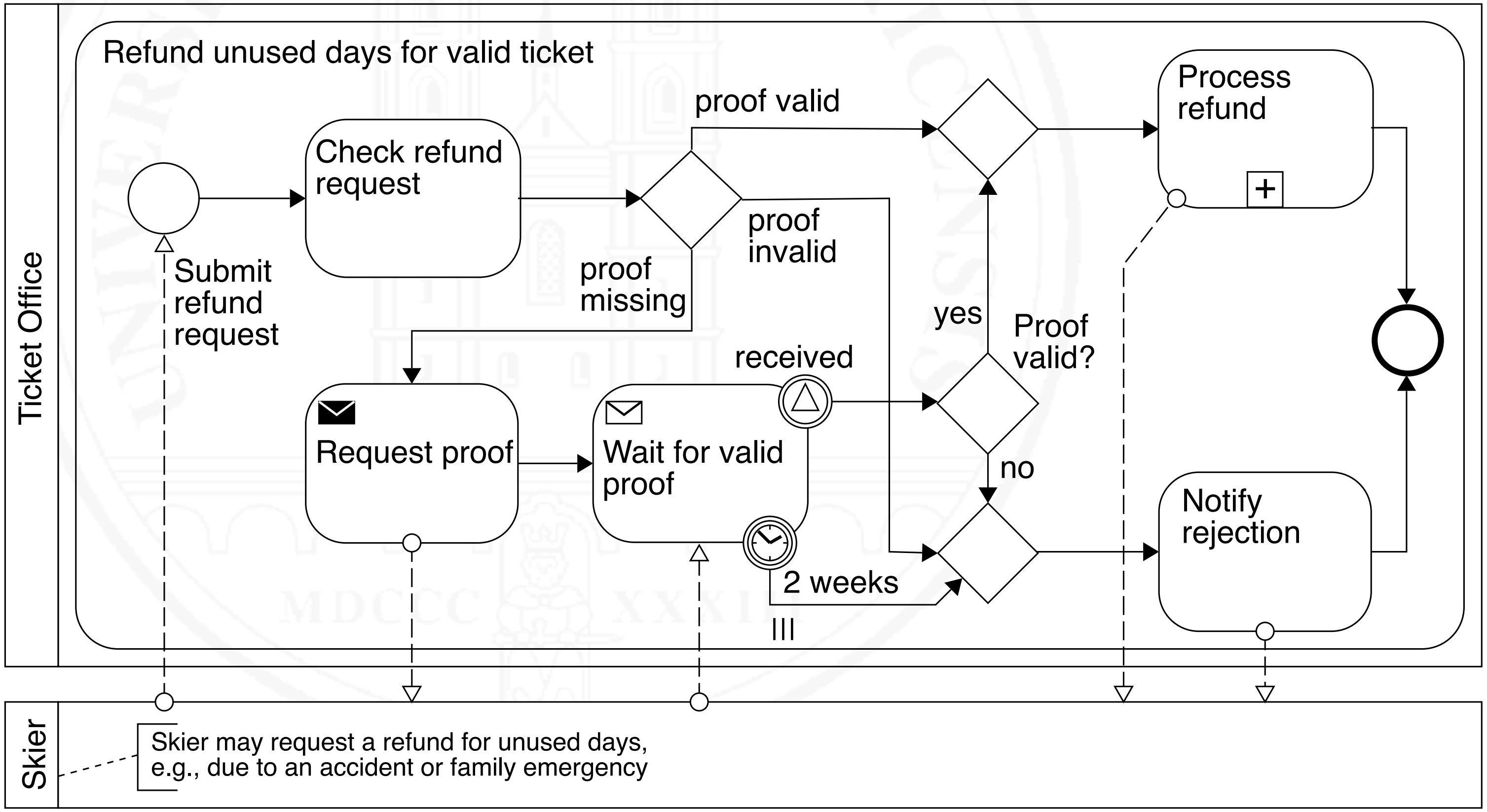
- Elements
  - Process steps / work steps
  - Events influencing the flow
  - Control flow
  - Maybe data / information access and responsibilities
- Typical languages
  - UML activity diagrams
  - BPMN
  - Event-driven process chains

Process modeling: BPMN

[Object Management Group 2013]

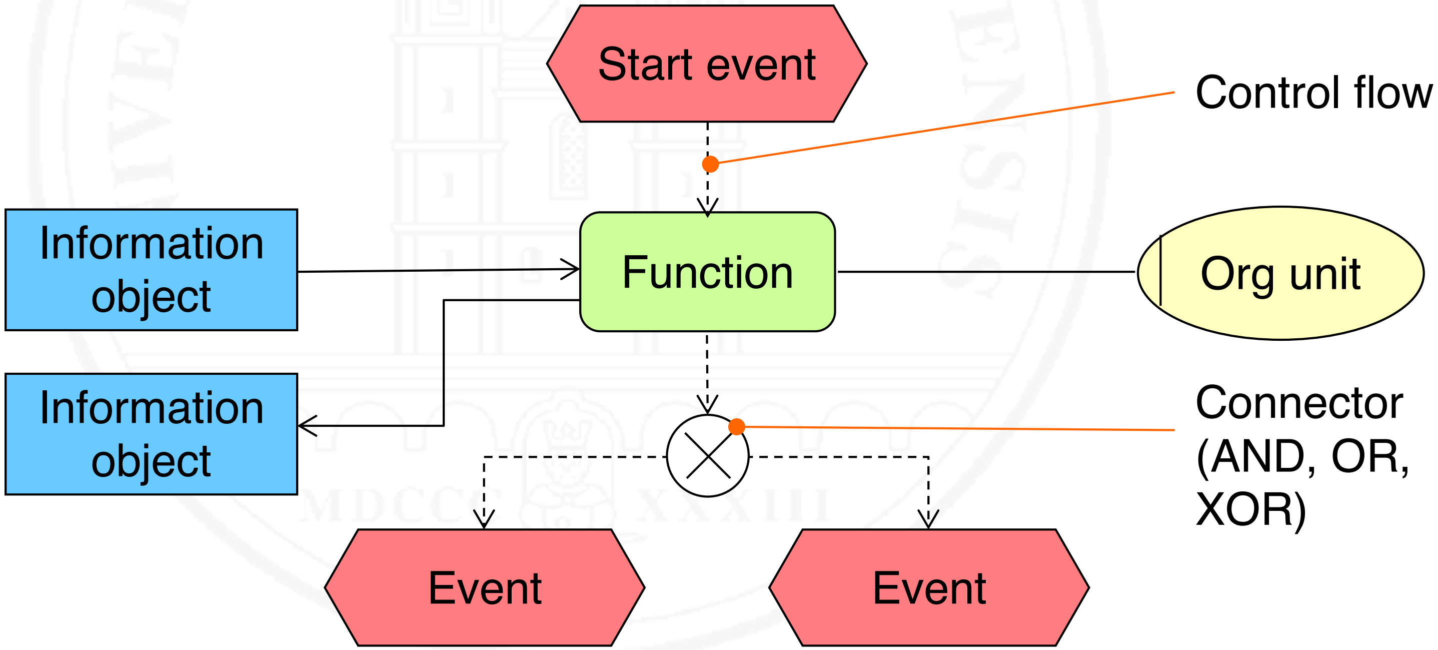
BPMN (Business Process Model and Notation)

- Rich language for describing business processes

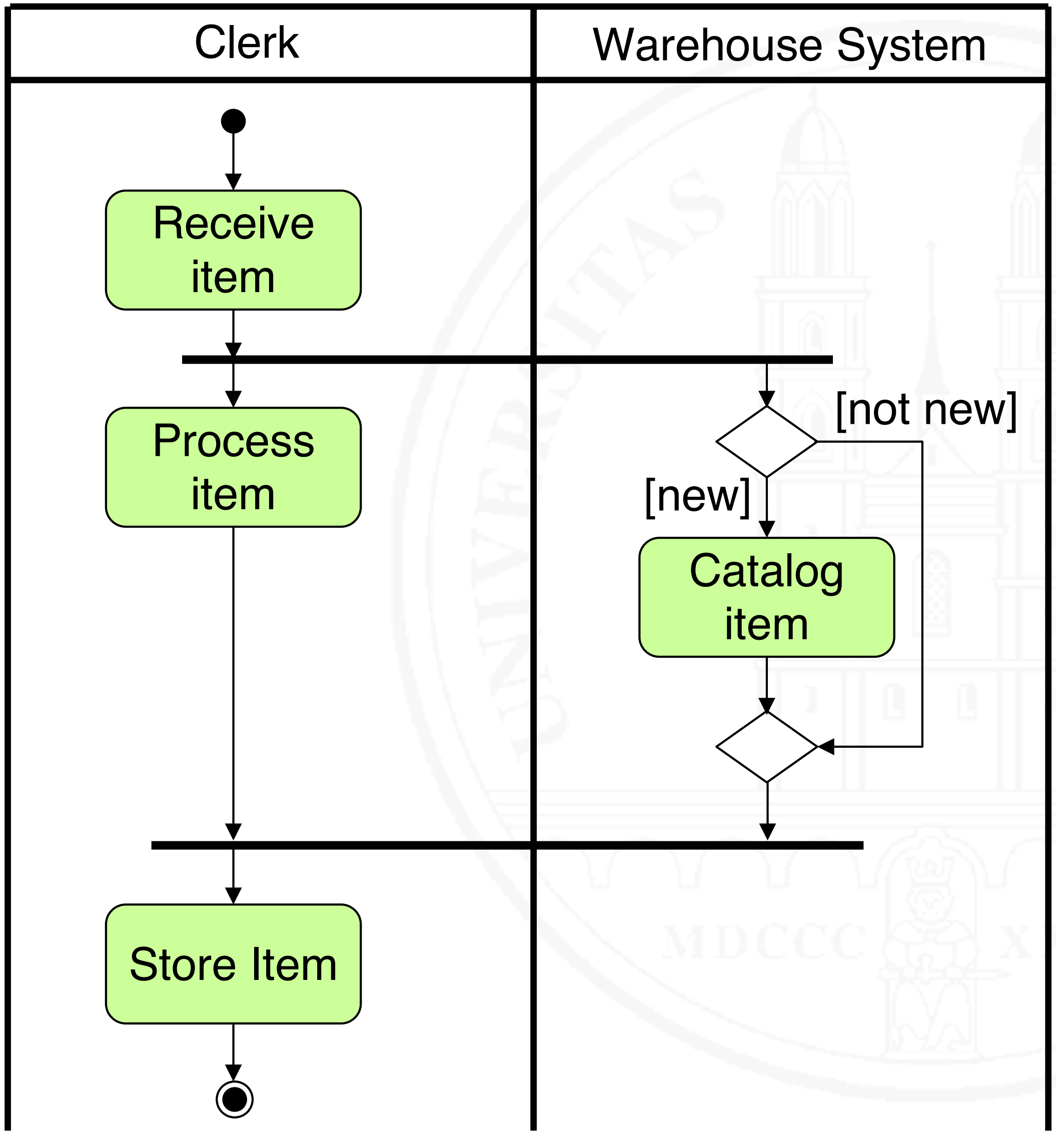


Process modeling: EPC

- Event-driven process chains (In German: ereignisgesteuerte Prozessketten, EPK)
- Adopted by SAP for modeling processes supported by SAP's ERP software



Process modeling: UML Activity Diagram



Initial node

Action

Fork node (fork into parallel flows)

Decision node (with decision conditions)

Merge node

Join node (join&synchronize flows)

Final node

Activity partitions (“swimlanes”) model who is responsible for what

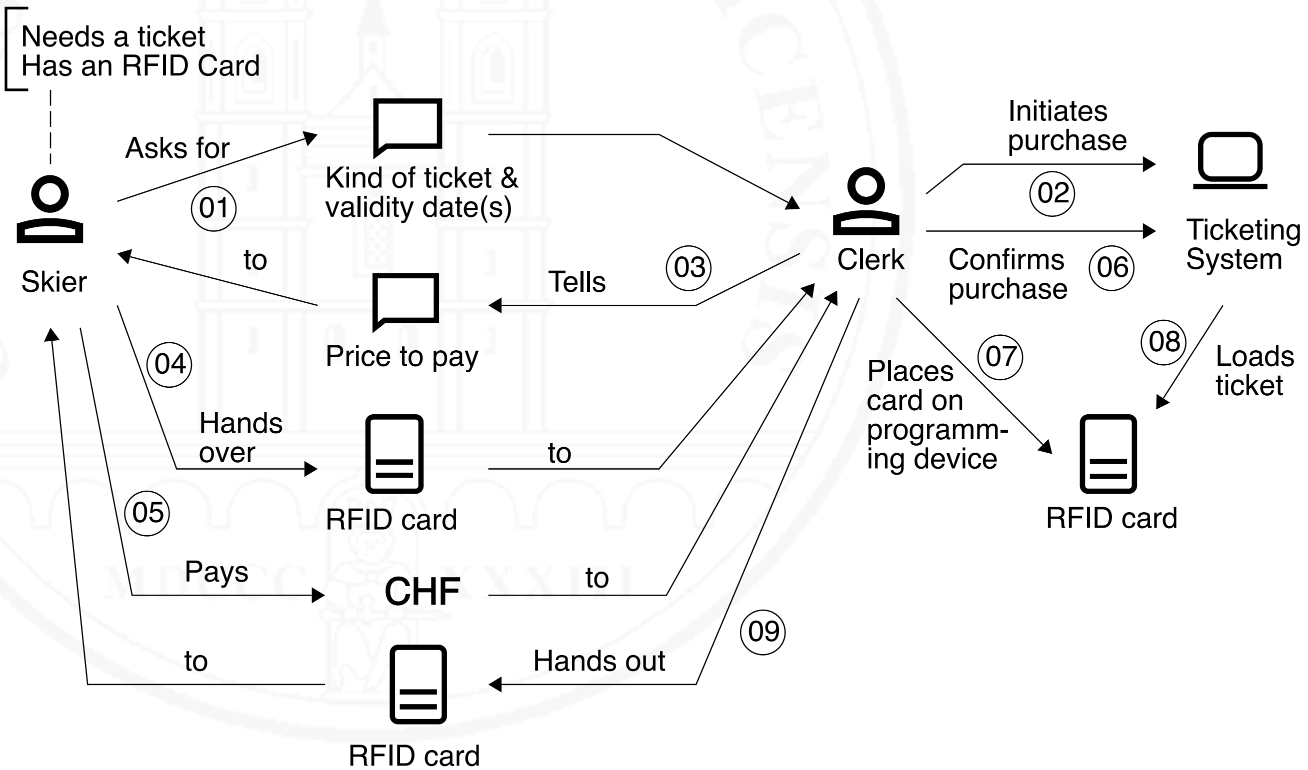
Mini-Exercise: Process modeling with UML

Model the ticket issuing process in the chairlift access control case study with skier, ticket clerk and ticketing system as actors/organizational units.

Domain story models

[Hofer&Schwendtner 2021]

- Visual stories about what stakeholders want to achieve
- Includes information about processes, system, people and organizations



The circled numbers indicate the order of steps in the story

7.4 Modeling state and behavior

Goal: describe dynamic system behavior

- How the system reacts to a sequence of external events
- How independent system components coordinate their work

Means:

- Finite state machines (FSMs) – not discussed here
- Statecharts / State machines
  - Easier to use than FSMs (although theoretically equivalent)
  - State machines are the UML variant of statecharts
- Sequence diagrams (primarily for behavioral scenarios)
- Petri nets – not discussed in this course

Statecharts

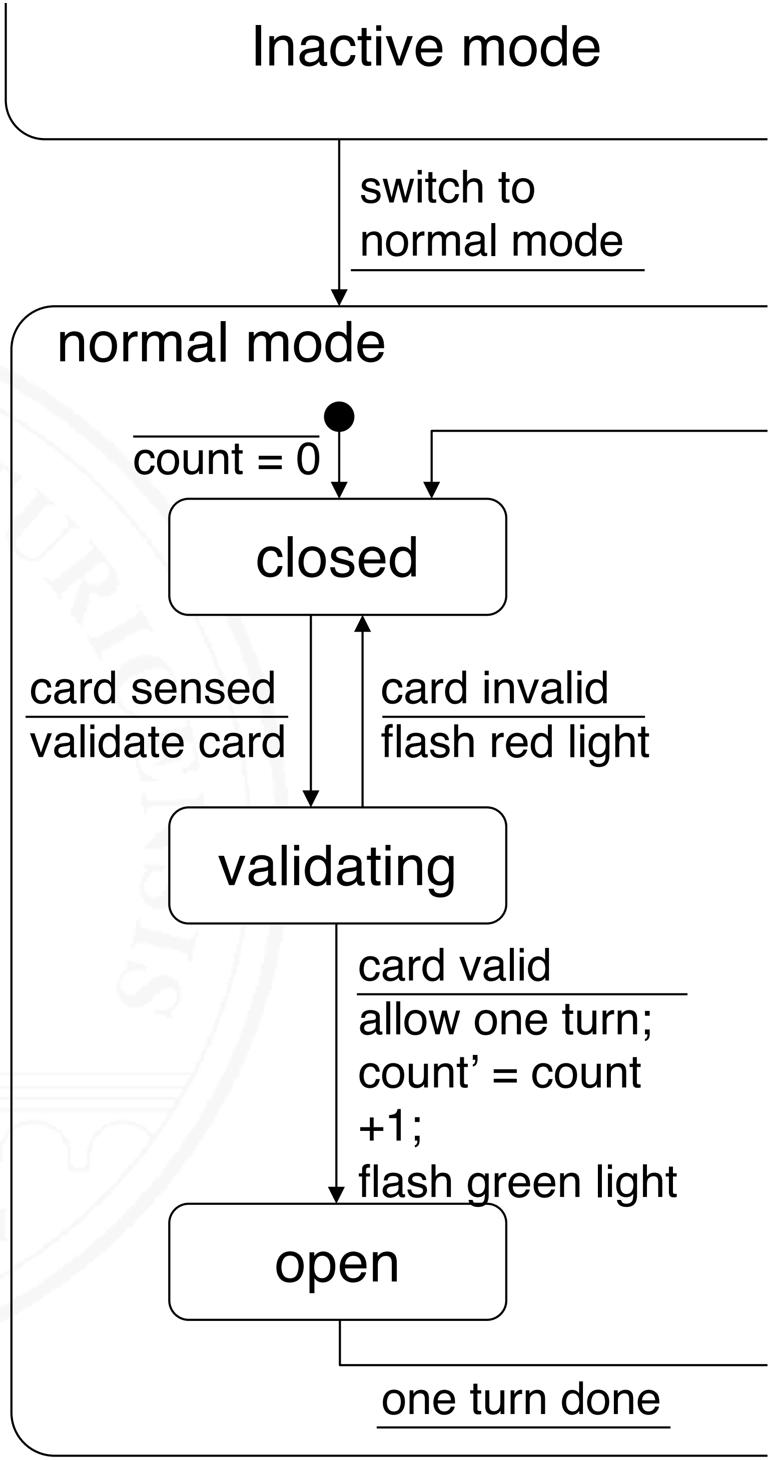
[Harel 1988]

- Models the *dynamic* behavior:
  - How the system reacts to external events in a given state
  - Reaction depends on actual state
  - States may be hierarchically nested and/or orthogonal (parallel)
- In UML: state machine diagrams

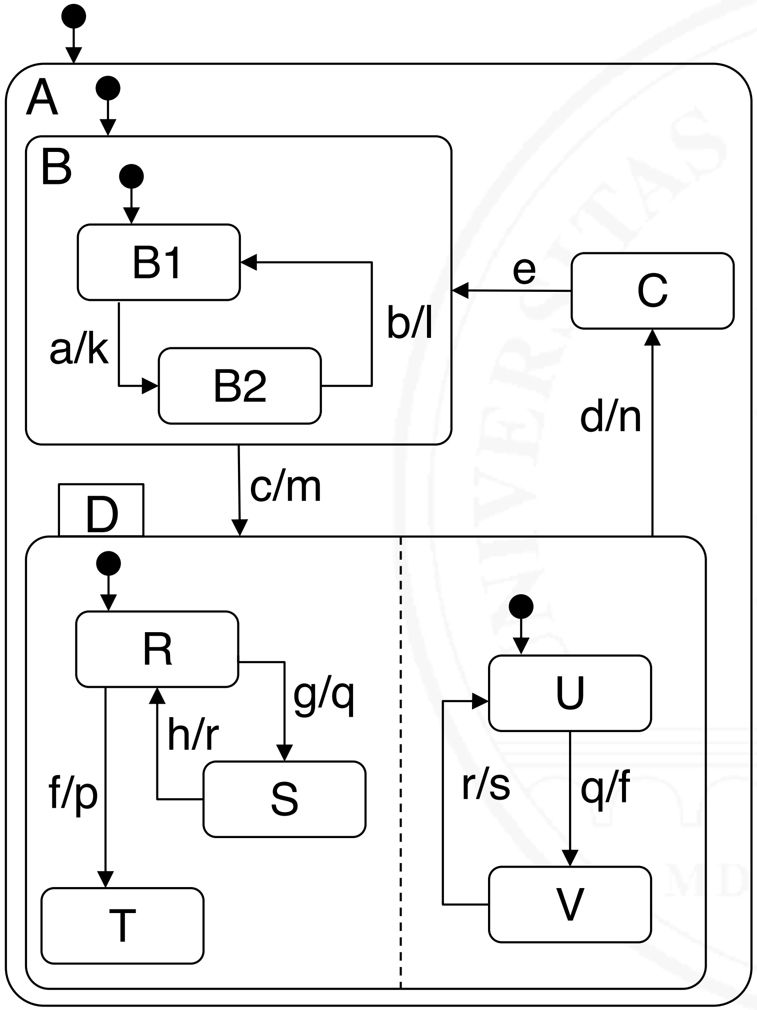
\+ Global view of system behavior

\+ Precise, but still readable

– Weak for modeling functionality and data

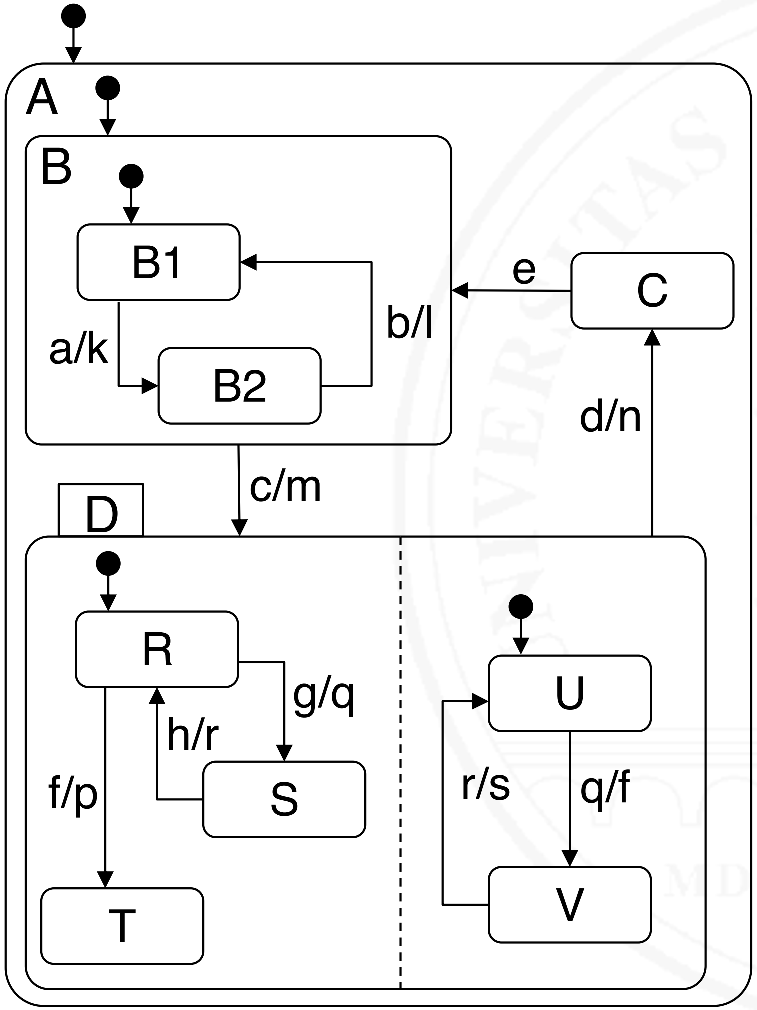


Interpretation of Statecharts



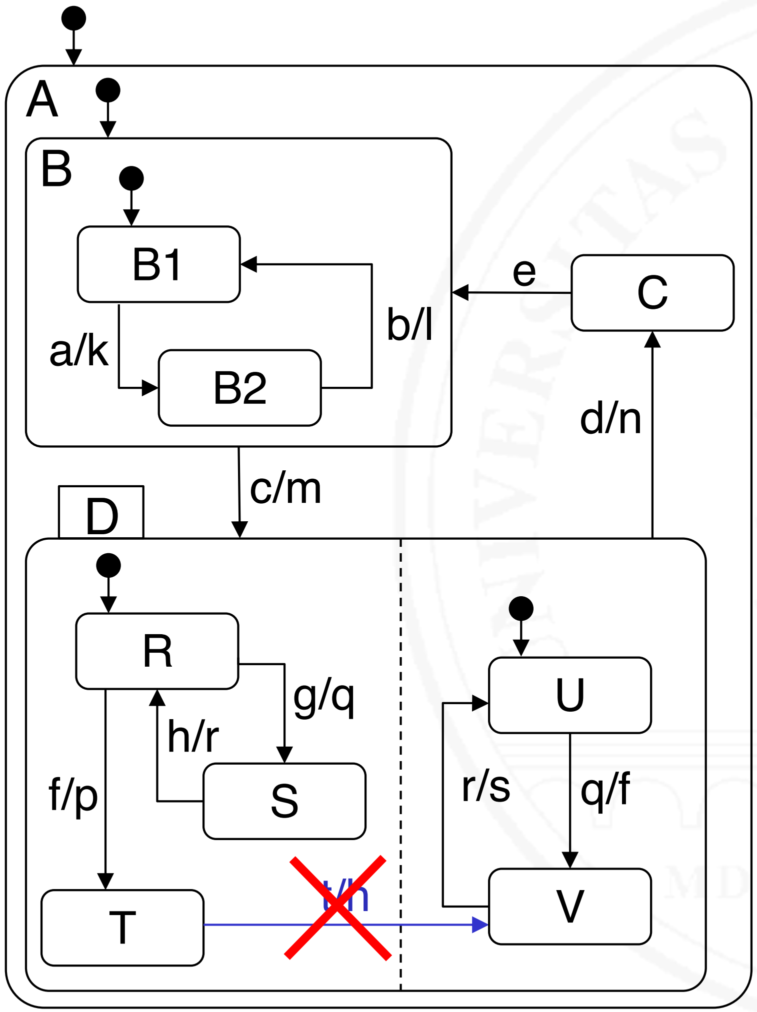
- Statecharts may have *composite states* with substates and *parallel regions*, for example:
  - B is a composite state, consisting of substates B1 and B2
  - D is a composite state with two parallel regions
- *Events* trigger *state transitions* and can trigger *actions* or *new events*, for example:
  The occurrence of event c triggers the transition from state B to D, provided the system currently is in state B. The transition triggers m, which may be an action or an event.

Interpretation of Statecharts – 2



- The system is always in exactly one combination of states and nested substates, for example:
  – Statechart A initially is in state B and its substate B1
  – After the occurrence of event c, A is in state D and substates (R, U)
  – After the occurrence of event f, A still is in state D, but now in substates (T, U)
- Events are ignored when there is no transition for it in the current state: e.g., in state B2, event f is ignored

Interpretation of Statecharts – 3

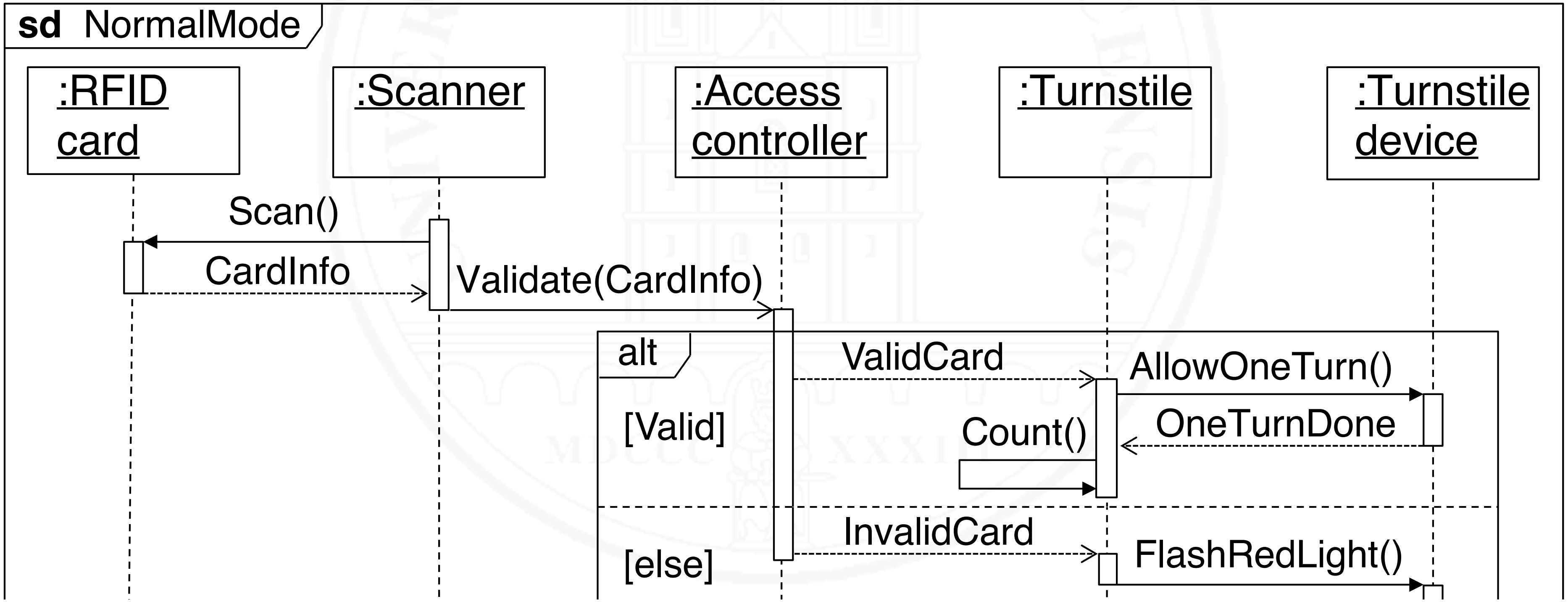


- State transitions into a composite state also enter its substates
- Leaving a state implies leaving all its substates
- Regions can influence each other via events, for example:
  If the system is in R and U, the event g triggers a transition from R to S, producing q. Event q in turn triggers a transition from U to V.
- Transitions between regions are forbidden

Sequence diagrams / MSCs

Object Management Group (2017)

- Models ...
  - ... lifelines of system components or objects
  - ... messages that the components exchange



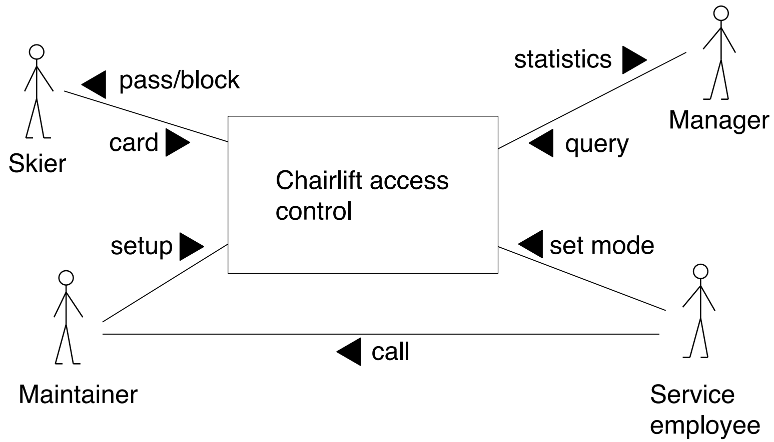
- Notation/terminology:
  - UML: Sequence diagram
  - Otherwise: Message sequence chart (MSC)

+ Visualizes component collaboration on a timeline

– In practice confined to the description of required scenarios

– Design-oriented, can detract from modeling requirements

7.5 Modeling context and boundary



Structural embedding

- Context diagrams, modeling
  - The system
  - The actors in the system's context
  - Information interfaces between actors and system
  - Information interfaces among actors

→ Chapter 2.4

Dynamic interaction between system and context

- Scenarios
- Use cases

Dynamic interaction: modeling the users' view

Describing the functionality of a system from a user's perspective: How can a user interact with the system?

Two key terms:

- Use case
- Scenario

[Carroll 1995,
Glinz 1995,
Glinz 2000a,
Jacobson et al. 1992,
Sutcliffe 1998,
Weidenhaupt et al. 1998]

Use case

DEFINITION. Use case – A set of possible interactions between external actors and a system that provide a benefit for the actor(s) involved.

Use cases specify a system from a user's (or other external actor's) perspective: every use case describes some functionality that the system must provide for the actors involved in the use case.

- Use case diagrams provide an overview
- Use case descriptions provide the details

[Jacobson et al. 1992, Glinz 2024]

Scenario

DEFINITION. Scenario – 1. In general: A description of a potential sequence of events that lead to a desired (or unwanted) result.
2. In RE: An ordered sequence of interactions between partners, in particular between a system and external actors. May be a concrete sequence (instance scenario) or a set of potential sequences (type scenario, use case).

[Carroll 1995,
Sutcliffe 1998,
Glinz 1995]

Use case / scenario descriptions

Various representation options

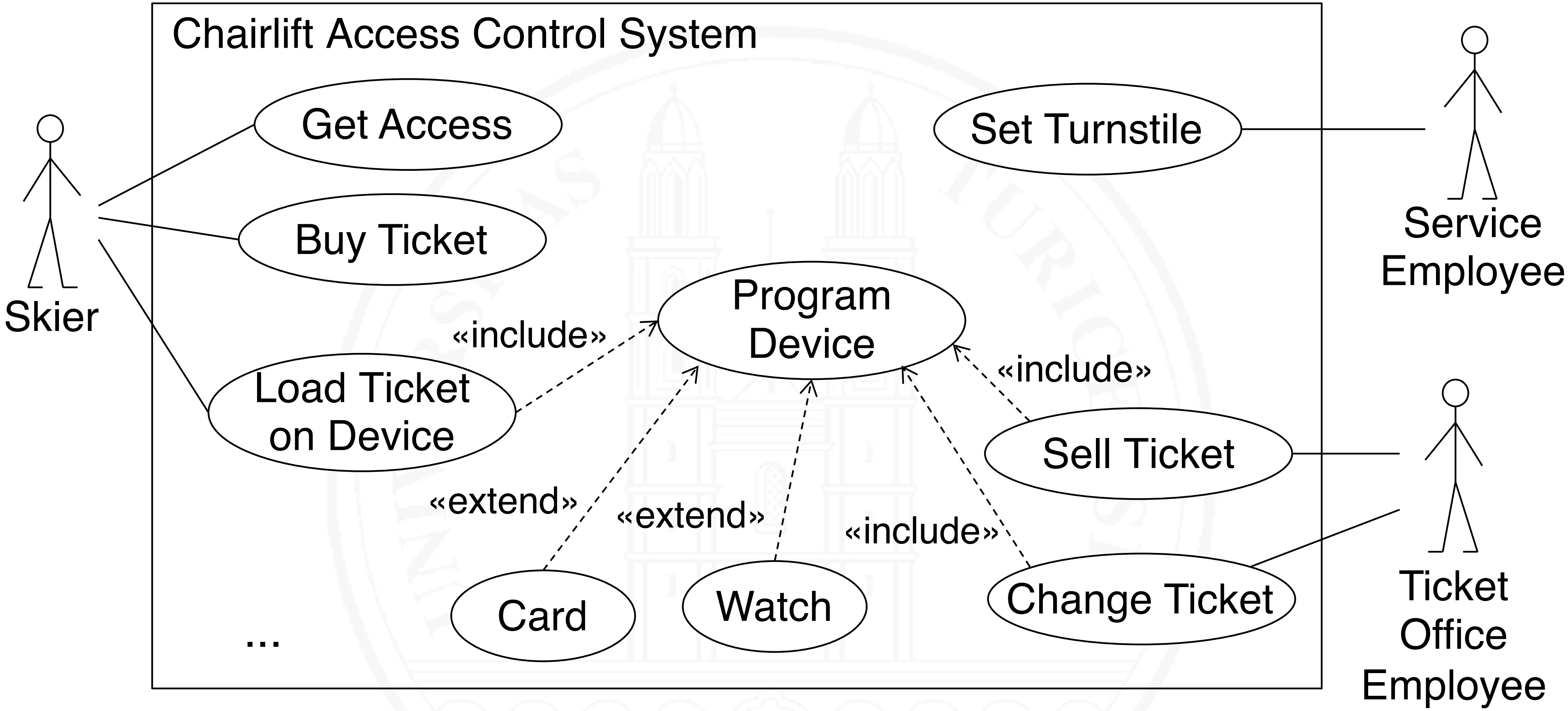
- Free text in natural language
- Structured text in natural language
- Statecharts / UML state machines
- UML activity diagrams
- Sequence diagrams / MSCs

Structured text is most frequently used in practice

A use case description with structured text

USE CASE SetTurnstiles
Actor: Service Employee
Precondition: none
Normal flow:
1 Service Employee chooses turnstile setup.
System displays controllable turnstiles: locked in red, normal in green, open in yellow.
2 Service Employee selects turnstiles s/he wants to modify.
System highlights selected turnstiles.
3 Service Employee selects Locked, Normal, or Open.
System changes the mode of the selected turnstiles to the selected one, displays all turnstiles in the color of the current mode.
...
Alternative flows:
3a Mode change fails: System flashes the failed turnstile in the color of its current mode.
...

UML Use case diagram



+ Provides abstract overview from actors' perspectives
– Ignores functions and data required to provide interaction
– Can't properly model hierarchies and dependencies

Dependencies between scenarios / use cases

- UML can only model inclusion, extension and generalization
- However, we need to model
  - Control flow dependencies (sequence, alternative, iteration)
  - Hierarchical decomposition
- Largely ignored in UML (Glinz 2000b)
- Options
  - Pre- and postconditions
  - Statecharts
  - Extended Jackson diagrams (in ADORA, Glinz et al. 2002)
  - Specific dependency charts (Ryser and Glinz 2001)

Dependencies with pre- and postconditions

Scenario AuthenticateUser
  Precondition: none
  Steps: ...
  Postcondition: User is authenticated

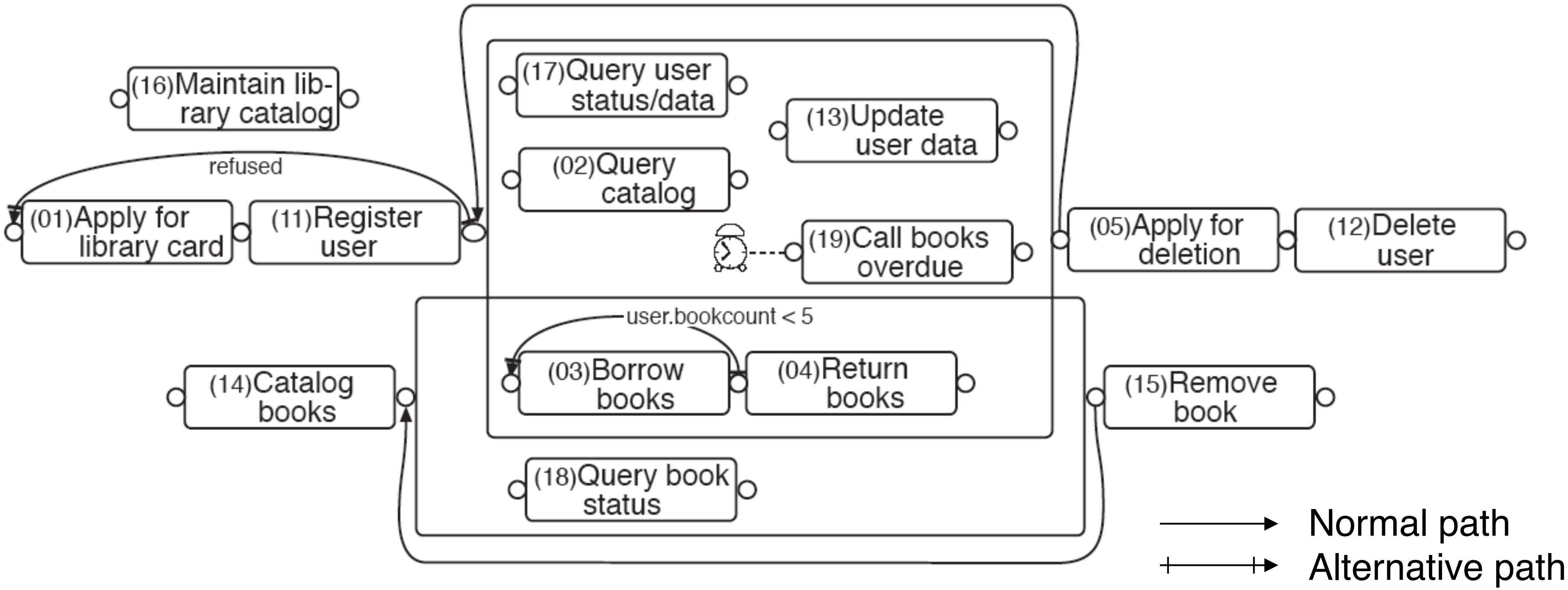
Scenario BorrowBooks
  Precondition: User is authenticated
  Steps: ...
  ...

Scenario ReturnBooks
  Precondition: User is authenticated
  Steps: ...
  ...

- Simple dependencies of kind «B follows A» can be modeled
- Relationships buried in use case descriptions, no overview
- No hierarchical decomposition
- Modeling of complex relationships very complicated

Dependency charts

- Specific notation for modeling of scenario dependencies (Ryser und Glinz 2001)
- Research result; not used in today's practice



Mini-Exercise: Writing a use case

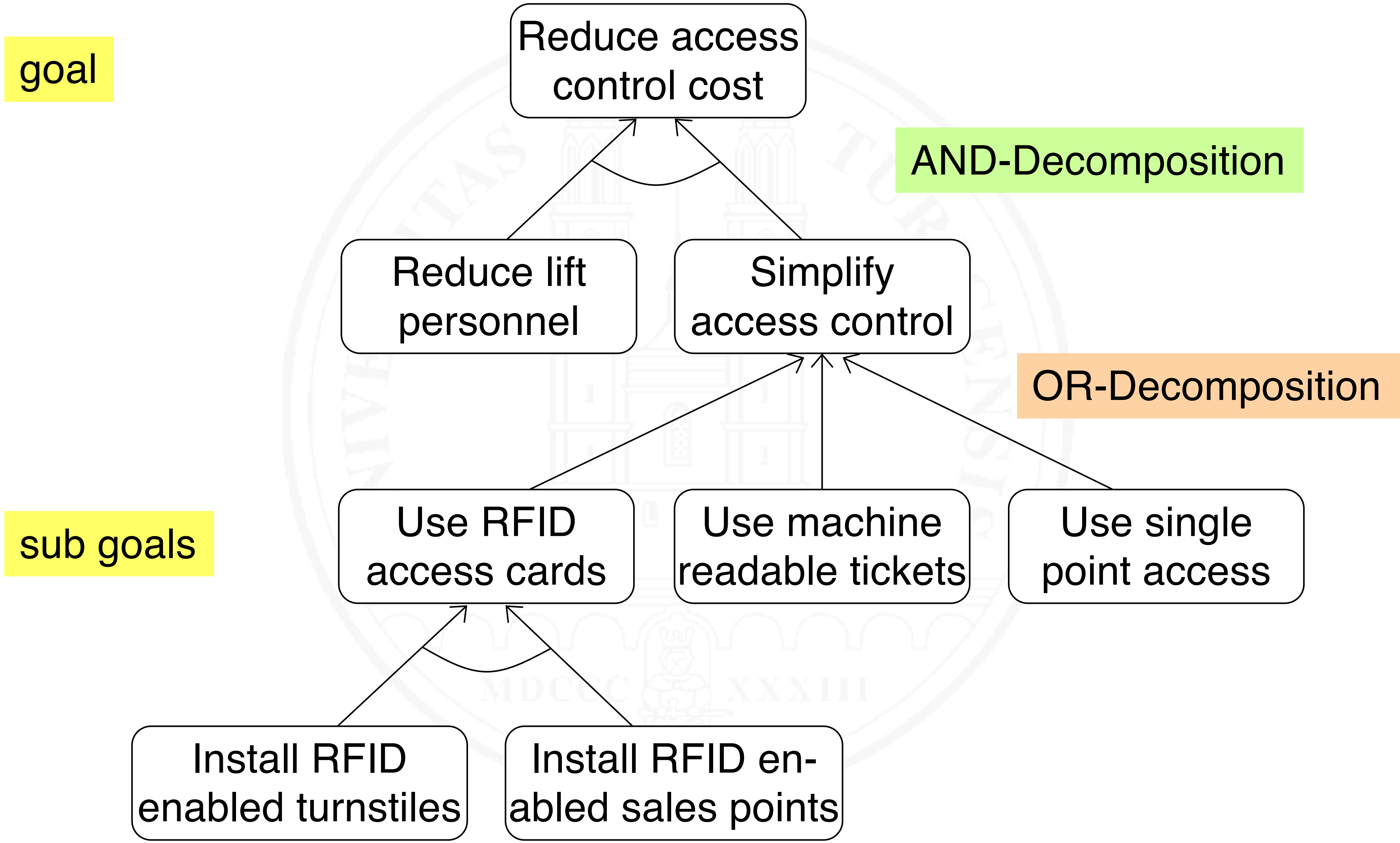
For the Chairlift access control system, write the use case "Get Access", describing how a skier gets access to a chairlift using his or her RFID ticket.

7.6 Modeling goals

- Knowing the goals of an organization (or for a product) is essential when specifying a system to be used in that organization (or product)
- Goals can be decomposed into sub goals
- Goal decomposition can be modeled with AND/OR trees
- Considering multiple goals results in a directed goal graph

[van Lamsweerde 2001, 2004
Mylopoulos 2006
Yu 1997]

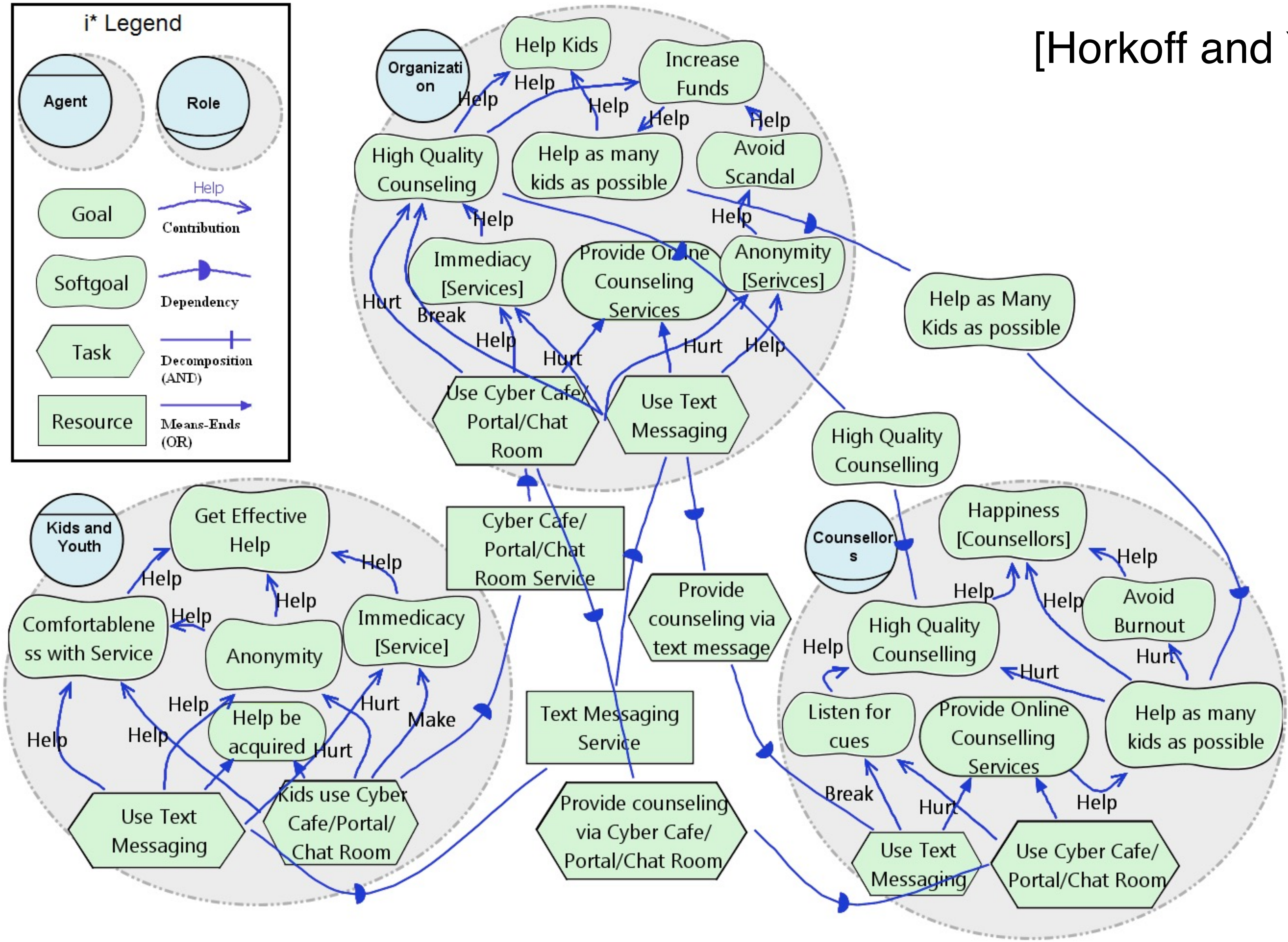
AND/OR trees for goal modeling



Goal-agent networks

- Explicitly models agents (stakeholders), their goals, tasks that achieve goals, resources, and dependencies between these items
- Many approaches in the RE literature
- i* is the most popular approach
- Rather infrequently used in practice

A real world i* example: Youth counseling



[Horkoff and Yu 2010]

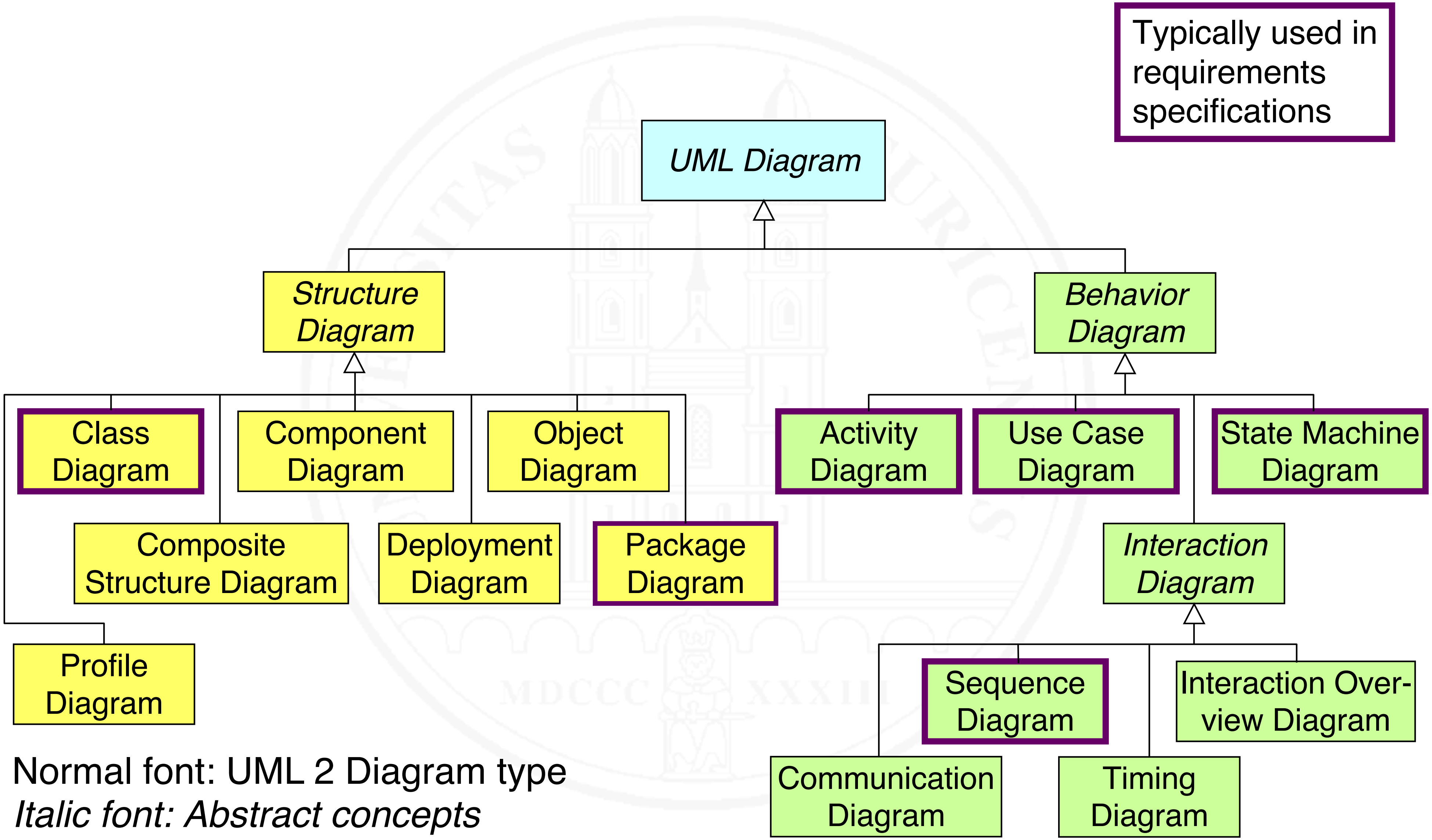
7.7 UML (Unified Modeling Language)

[Object Management Group 2017]

- UML is a collection of primarily graphic languages for expressing requirements models, design models, and deployment models from various perspectives
- A UML specification typically consists of a collection of loosely connected diagrams of various types
- Additional restrictions can be specified with the formal textual language OCL (Object Constraint Language)

[Object Management Group 2014]

UML – Overview of diagram types



Normal font: UML 2 Diagram type
Italic font: Abstract concepts

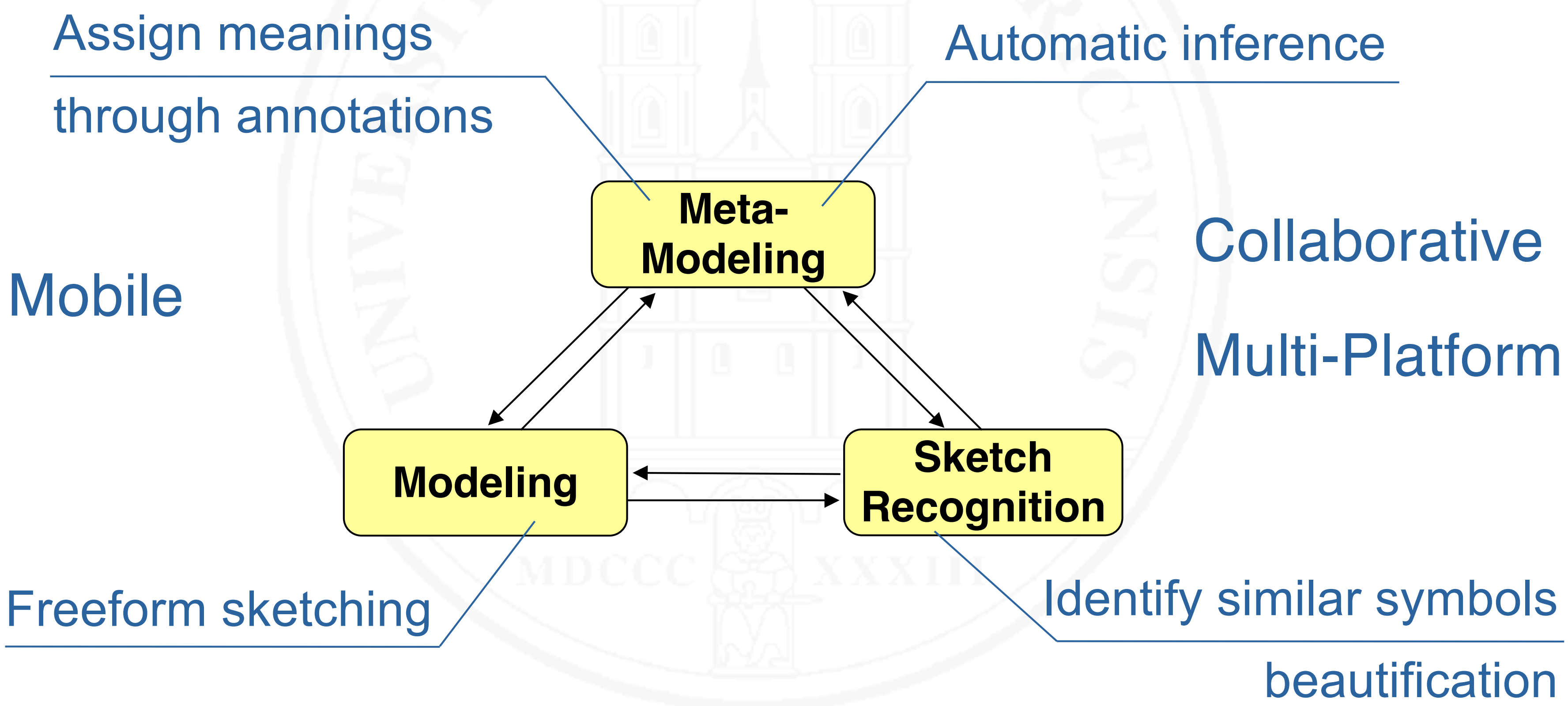
7.8 Lightweight, flexible modeling

- Modeling languages – Have a predetermined syntax
  - Limited expressibility and flexibility
  - ➔ Too restrictive for sketching ideas or initial requirements
- Free-form sketching – Is fully flexible
  - Resulting sketches do not carry any structure or meanings
  - ➔ Too vague when sketches serve as a basis for further RE tasks
- Need for a middle-ground approach
  - High flexibility; no fixed set of language constructs
  - Co-evolution of models and model syntax & meanings
  - ➔ FlexiSketch

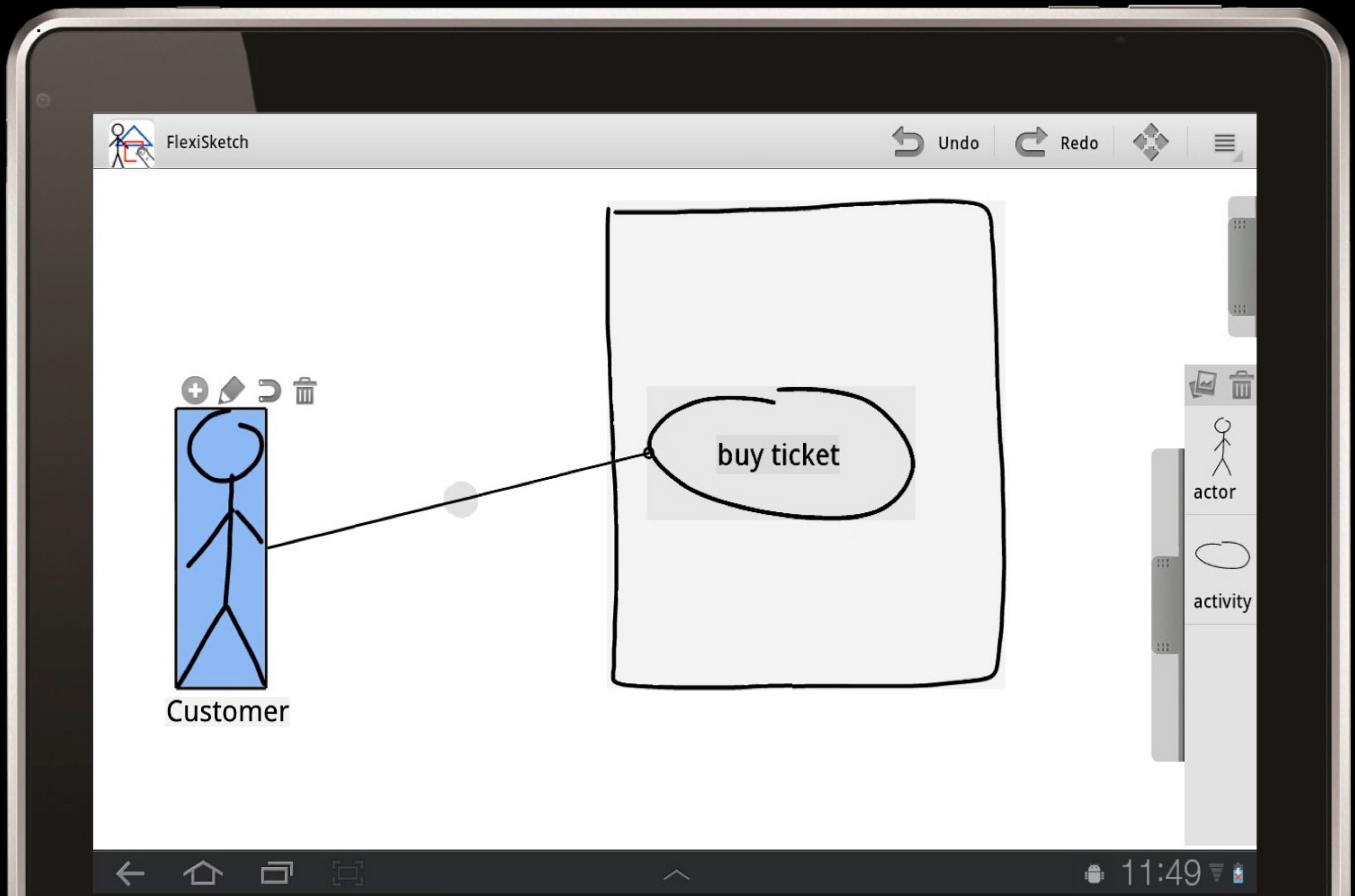
[Wüest, Seyff, Glinz 2019]
www.flexisketch.org

FlexiSketch – supporting flexible modeling

- Allow users to define their own notations & languages on the fly
- Co-evolve models and their metamodels



FlexiSketch Demo



Watch the FlexiSketch TEAM Demo Video at http://www.flexisketch.org

Dustin Wüest, Norbert Seyff, Martin Glinz (2015). FlexiSketch Team: Collaborative Sketching and Notation Creation on the Fly. *37th International Conference on Software Engineering (ICSE 2015). 685-688.*