
Development of Component- and Service-Oriented Software Architectures with UML and UML-RT — Introduction —

Ingolf H. Krueger
ikrueger@ucsd.edu

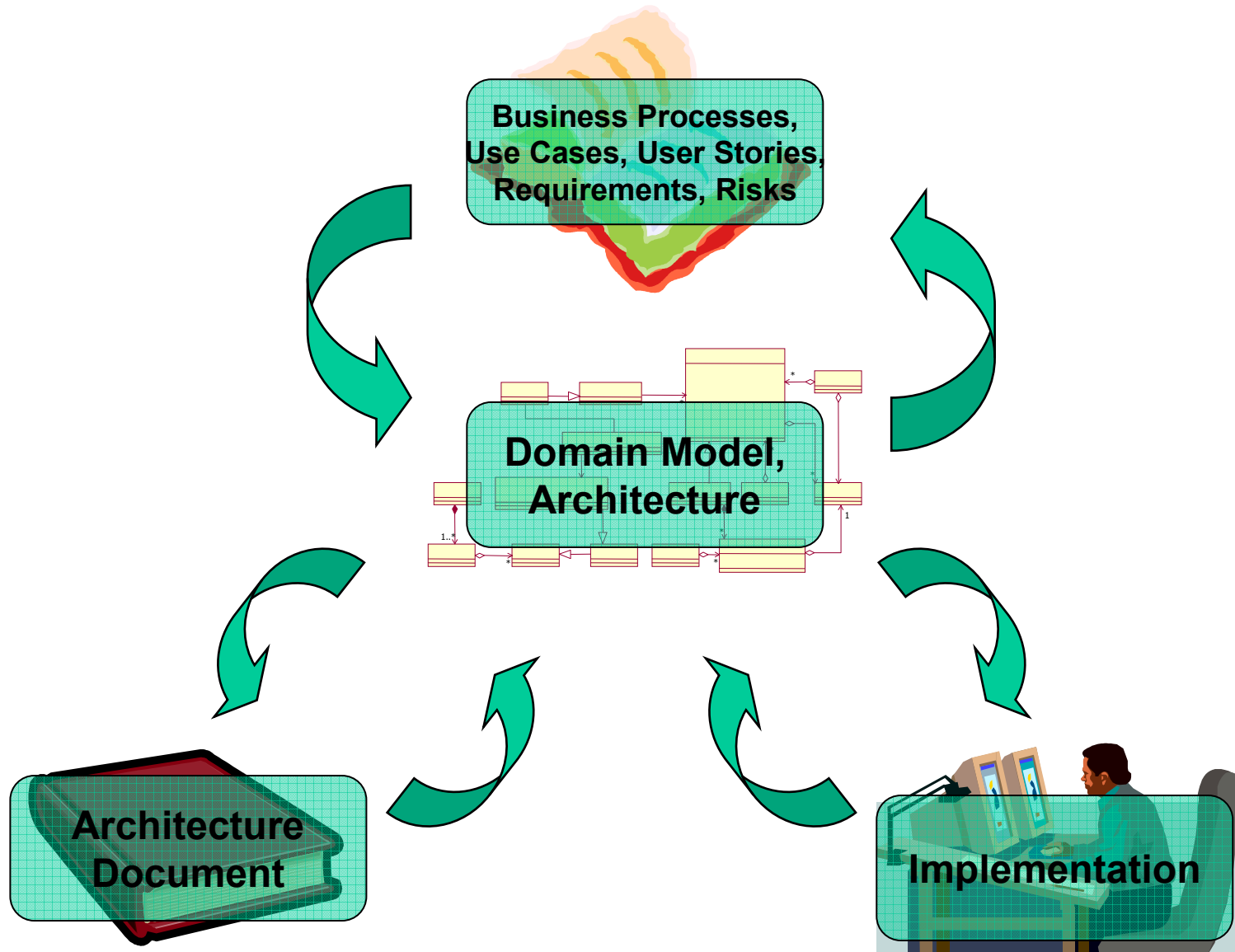
Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0404, USA



California Institute for Telecommunications
and Information Technologies
La Jolla, CA 92093-0405, USA



Recap: Architecture vs. Process

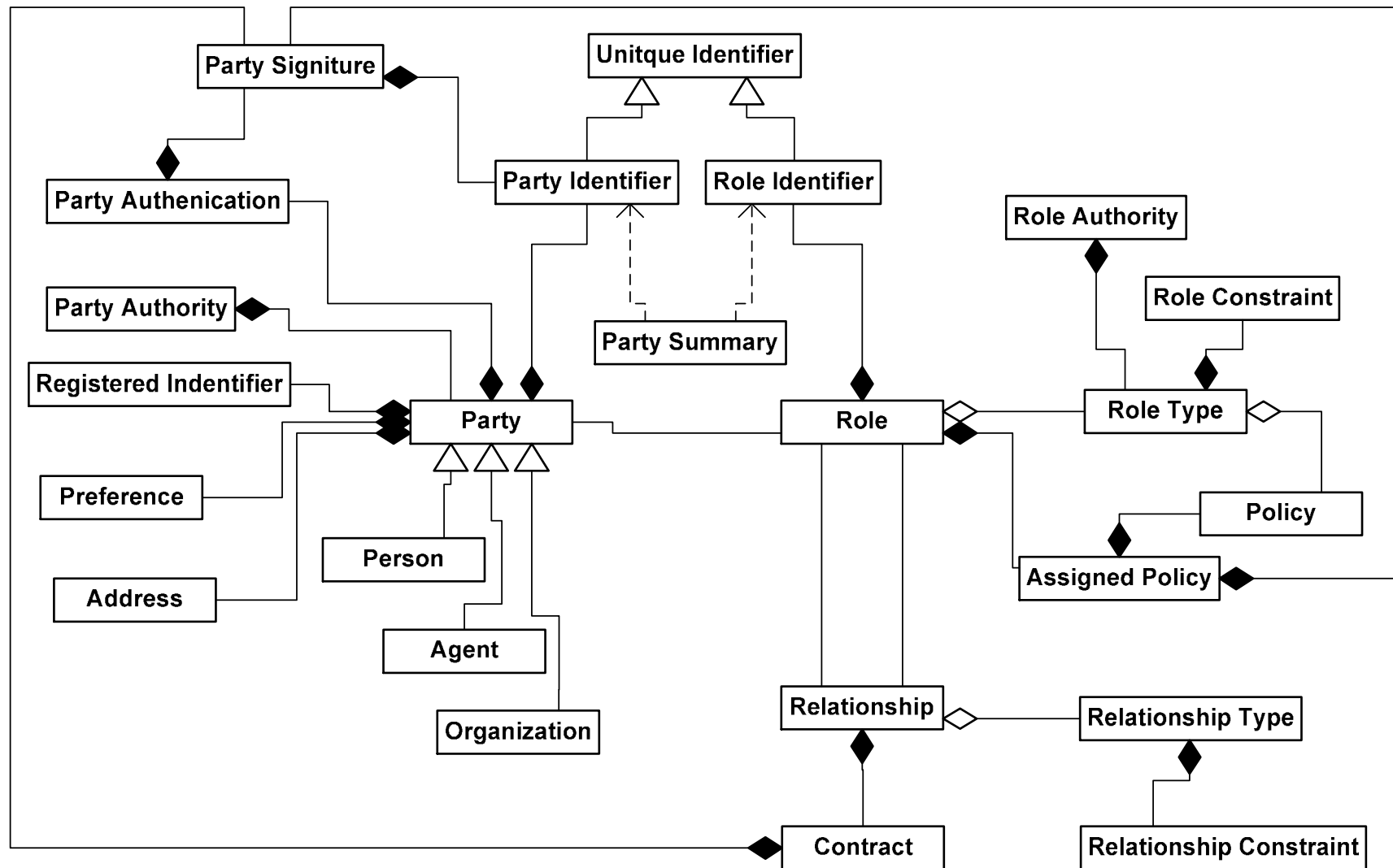


Tools for Architecture Development/Documentation

- Domain model is centerpiece of architecture development/documentation process
- What is a domain model?
- What can you do with it?
- How to find a domain model?
- How to document it?
- How to implement it?
- Tools:
 - Object-Oriented Design/Implementation
 - Component-Oriented Design/Implementation
 - Service-Oriented Design/Implementation
 - Refinement/Refactoring

Example: Party – Role – Relationship – Policy Model*

* courtesy of Matthew Arrott; adapted from J. Arlow, I. Neustadt: Enterprise Patterns and MDA : Building Better Software with Archetype Patterns and UML, Addison-Wesley, 2003



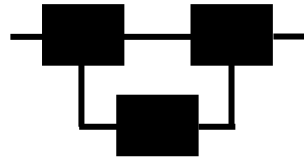
Overview

- Object-Orientation
 - Modeling concepts / Advantages / Shortcomings
- Component-Orientation
 - Modeling concepts / Advantages / Shortcomings
- UML – An Overview
 - Evolution / Modeling concepts / Notations

Basic Concepts of Object-Orientation

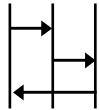
Objects as "Black boxes"

- encapsulation of data and operations
- interface definition



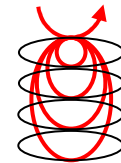
- sender/receiver-model
- loose coupling

Communication via Messages



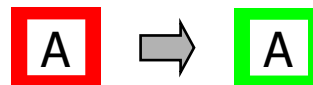
Object-Orientation

Seamlessness



Reuse

- classes
- inheritance
- composition



cross-phase

- terminology
- abstraction

OOA/OOD/OOI

OOA (Object-Oriented Analysis)

Application of object-oriented principles during requirements analysis

OOD (Object-Oriented Design)

Application of object-oriented principles during system-Design (make captured requirements more concrete)

OOI (Object-Oriented Implementation)

Implementation of a design using an OO-programming language (such as Java, C++, Smalltalk, ...)

Advantages of OO

- Support for abstraction
 - Application-/Problem-domain
 - Implementation
- Support for encapsulation
 - Locality of data and behavior
 - Interface definition
- Support for reuse
 - Classes
 - Inheritance
 - Composition

Shortcomings of OO

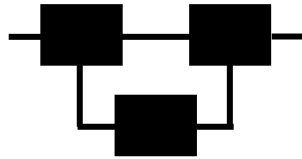
- Much leeway in application of modeling techniques
 - ⇒ High demand for methodological support
- No adequate concept for hierarchy
 - ⇒ Lack of structuring of OO models
- Too much emphasis on inheritance as principle for reuse
 - ⇒ Incomprehensible models for behavior
 - ⇒ Lack of modularity

Overview

- Object-Orientation
 - Modeling concepts / Advantages / Shortcomings
- Component-Orientation
 - Modeling concepts / Advantages / Shortcomings
- UML – An Overview
 - Evolution / Modeling concepts / Notations

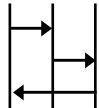
Basic Concepts of Component-Oriented

Components as "Black boxes"



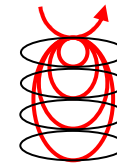
- encapsulation of data and operations
- interface definition
- concepts for hierarchy

Communication via Messages



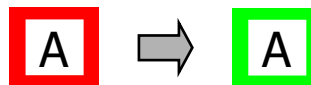
Component-Oriented

Seamlessness



Reuse

- classes
- composition
- (inheritance)



Advantages of Component-Orientation

- All advantages of OO
- In addition:
 - Support for hierarchy
 - Support for modularity
 - Support for concurrency
- Support for implementation via “middleware”
- Disciplined use of OO paves way towards component-orientation

Shortcomings of Component-Oriented

- Increased overhead for interface definitions
- Increased number of indirections in implementation
- Middleware (CORBA, .NET, DCOM, ...) required
- Still:
 - Lack of standard components
 - Lack of methodological support

Overview

- Object-Orientation
 - Modeling concepts / Advantages / Shortcomings
- Component-Orientation
 - Modeling concepts / Advantages / Shortcomings
- UML – An Overview
 - Evolution / Modeling concepts / Notations

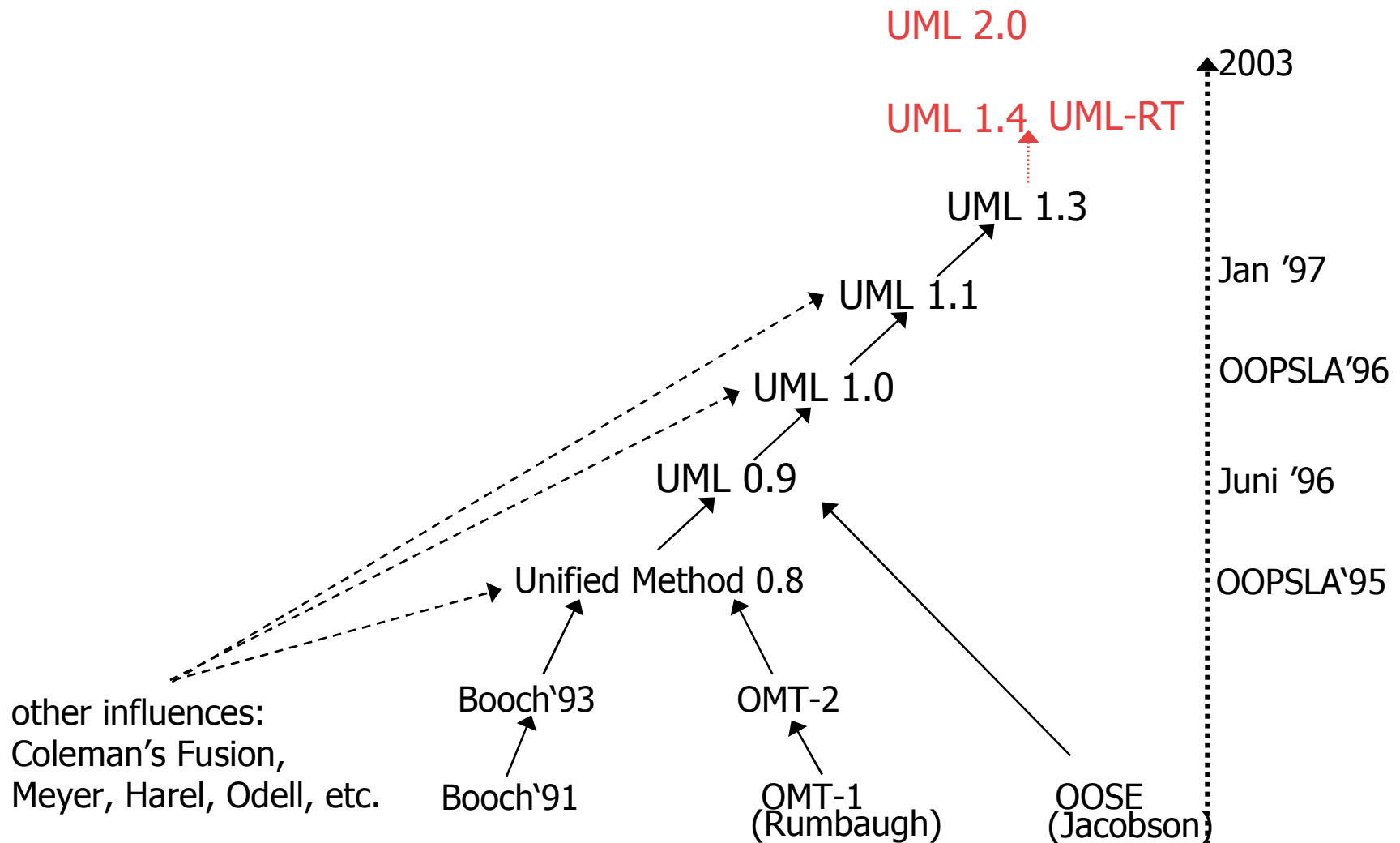
Why UML?

- Unified Modeling Language
 - Graphical modeling language for OO systems
 - Industry-wide standard
- Build “blueprint” of key system properties
- Means of communication
 - Among developers
 - Between developers and users
- Applicable across development phases

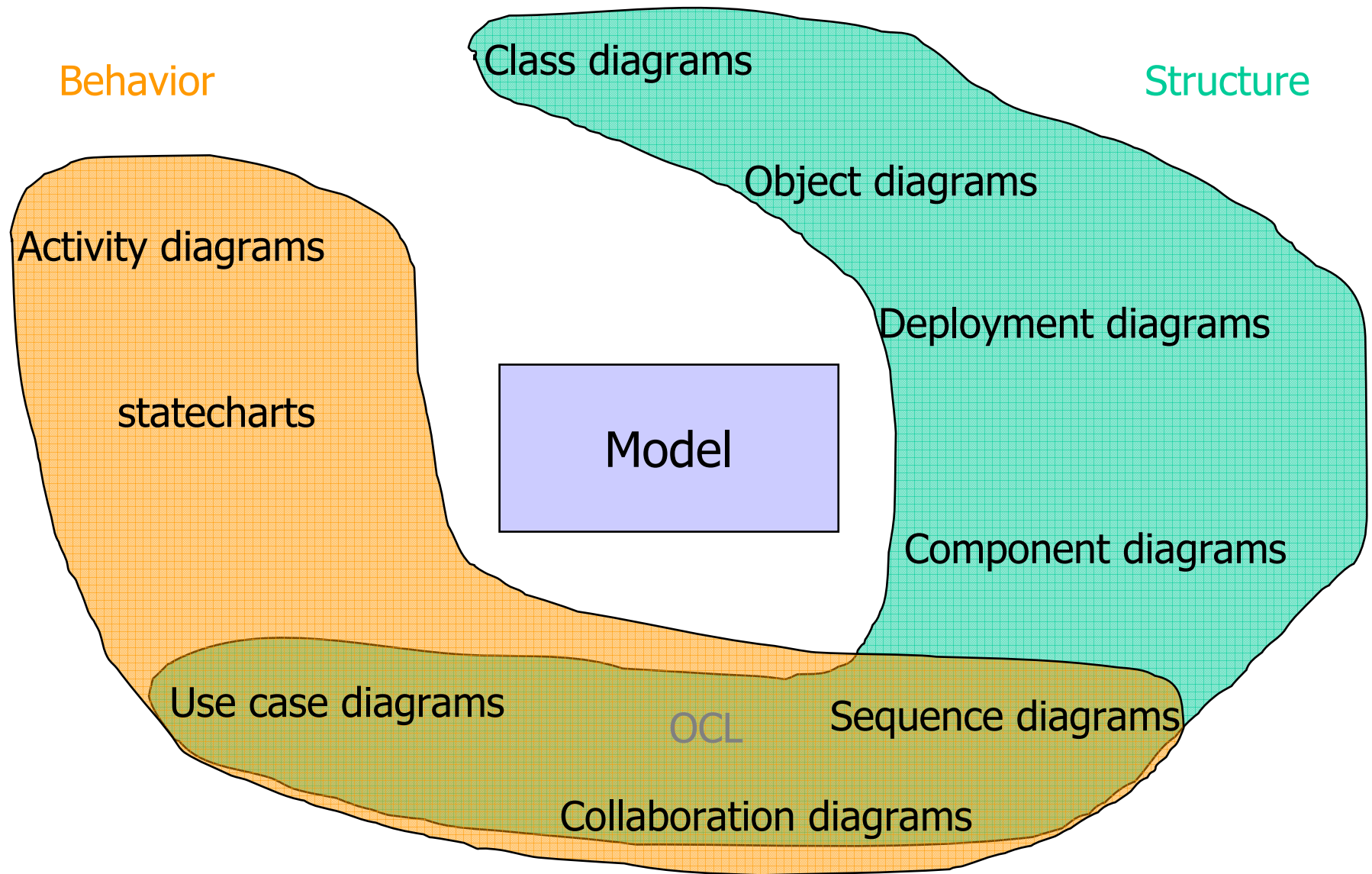
Why UML?

- Structuring of problem and solution
- Abstraction from implementation details possible
- Definition of multiple views:
 - Structure
 - Interaction
 - Behavior
 - Distribution
 - ...
- Many (but not all!) requirements at ADLs are met

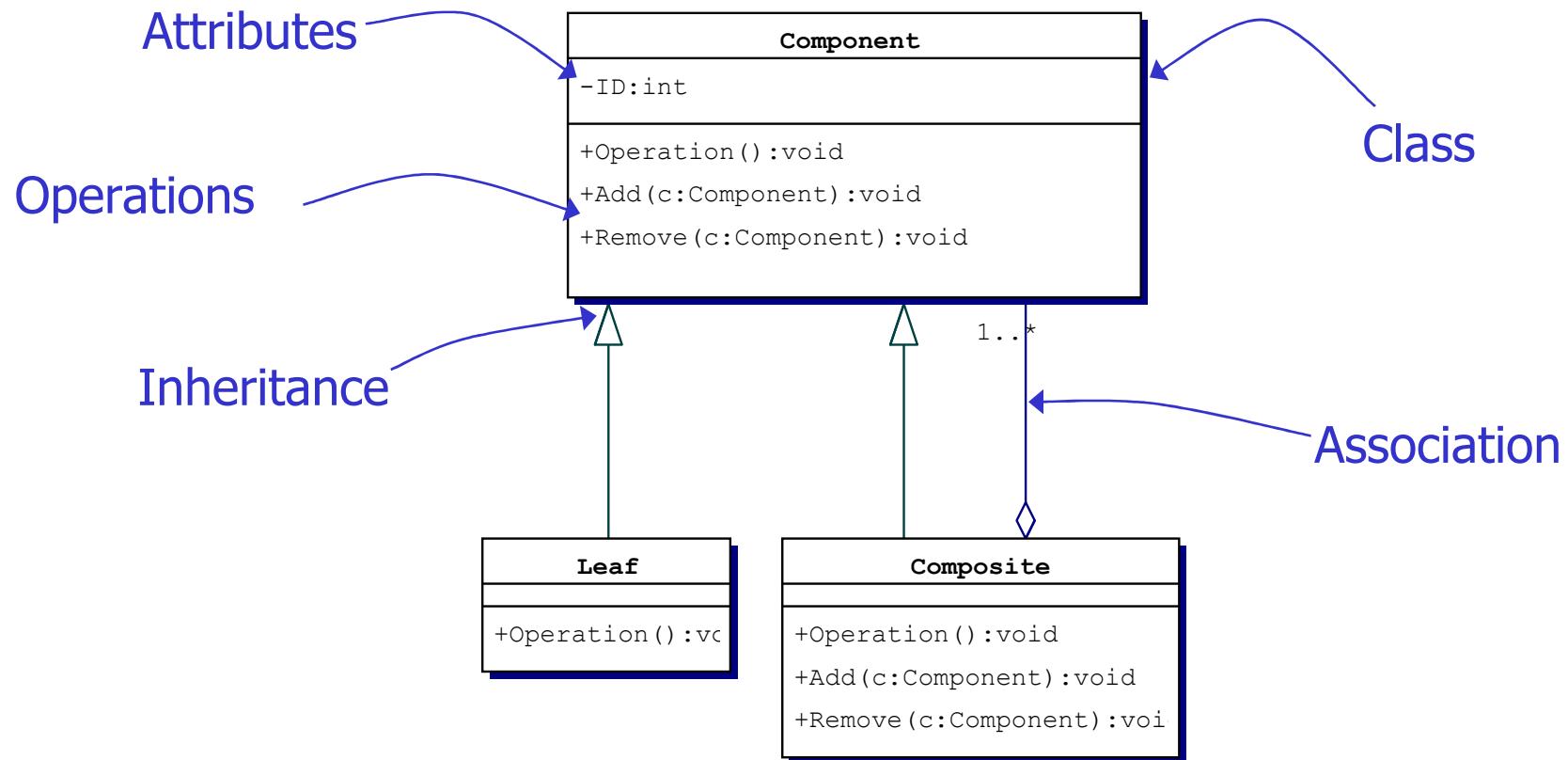
Evolution of the UML



Notations of the UML



Class Diagrams

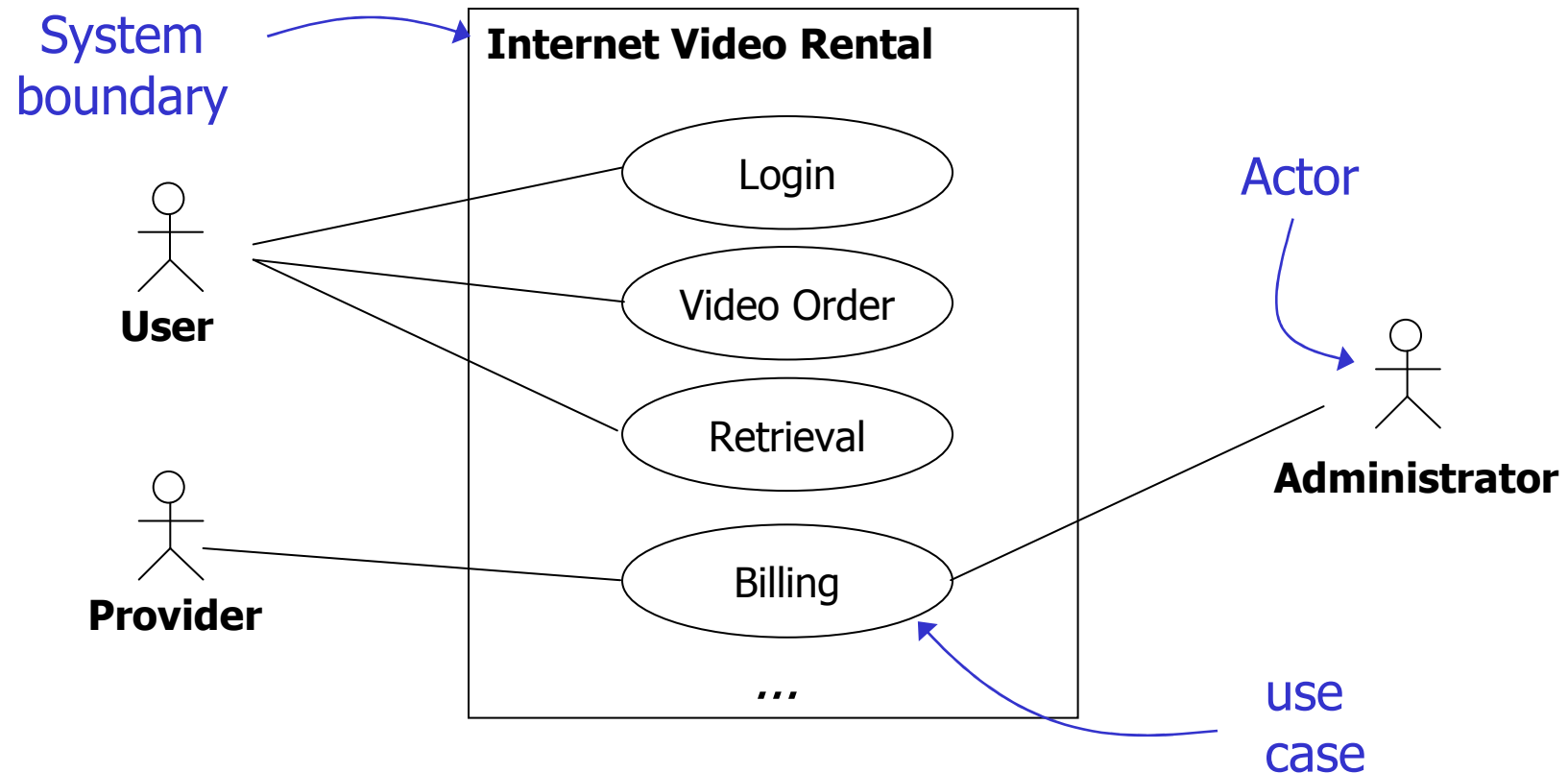


- View on static system structure
- Classes and their relationships
- Interfaces

Object Diagrams

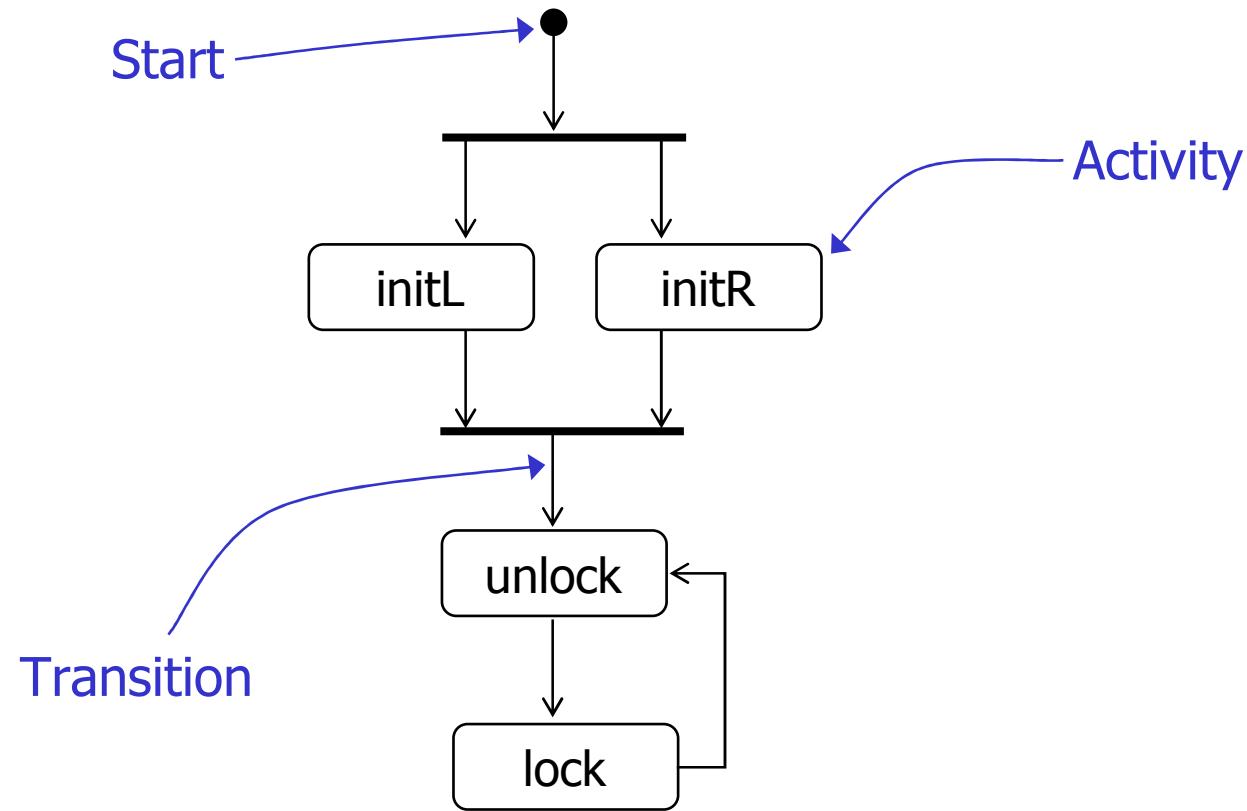
- View on static system structure
- Instantiation of class diagrams
- Objects and their relationships
- System “snapshot”

Use Case-Diagrams



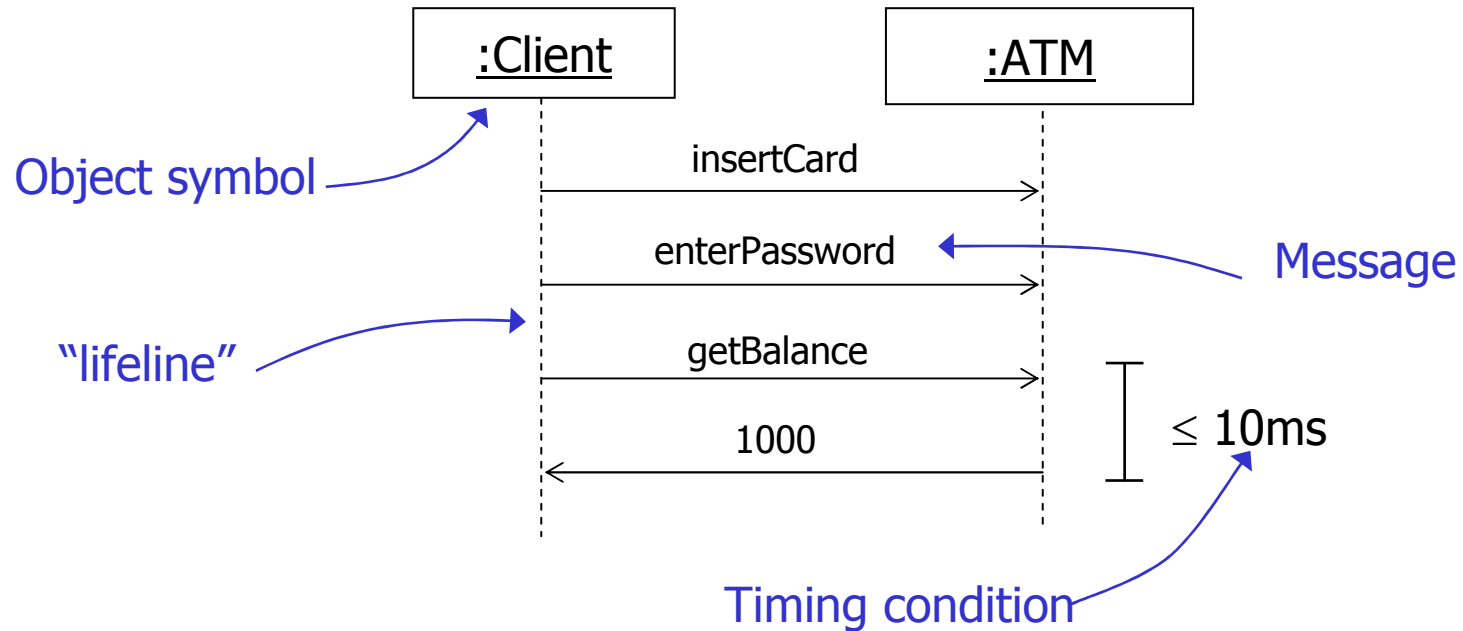
- View on
 - System services
 - Actors involved in services
- Depends on definition of system boundary

Activity Diagrams



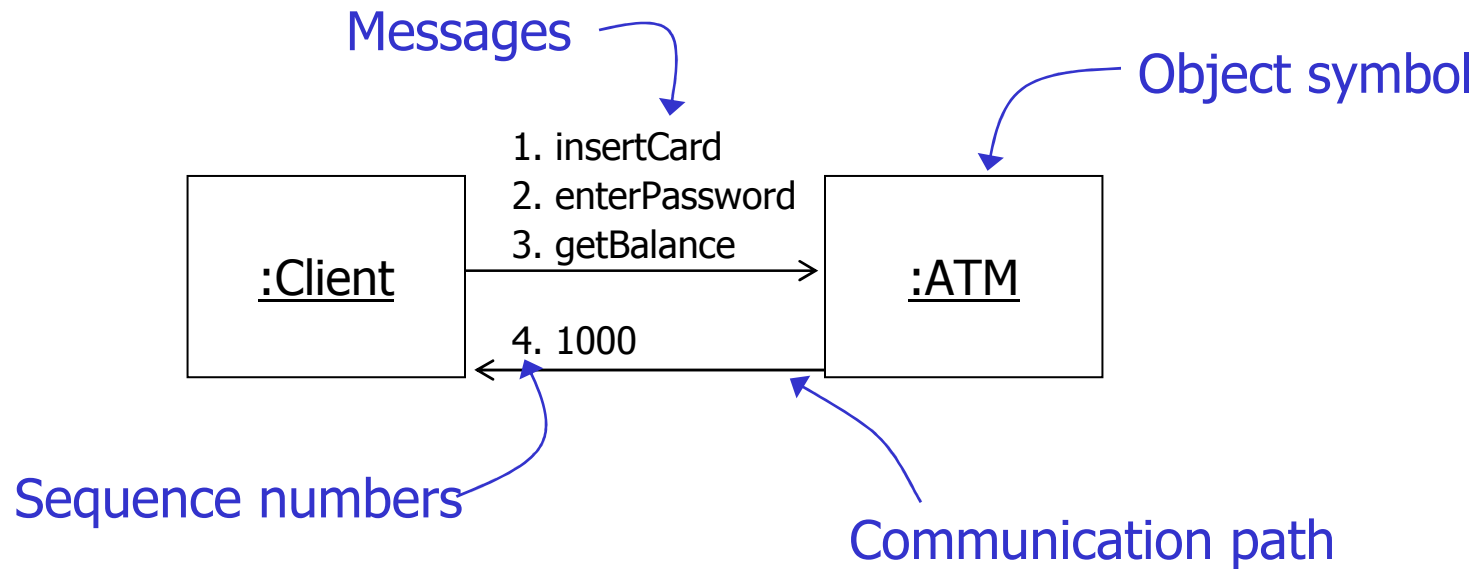
- View on system activities
- Behavioral view
- No assignment of activities to components

Sequence Diagrams



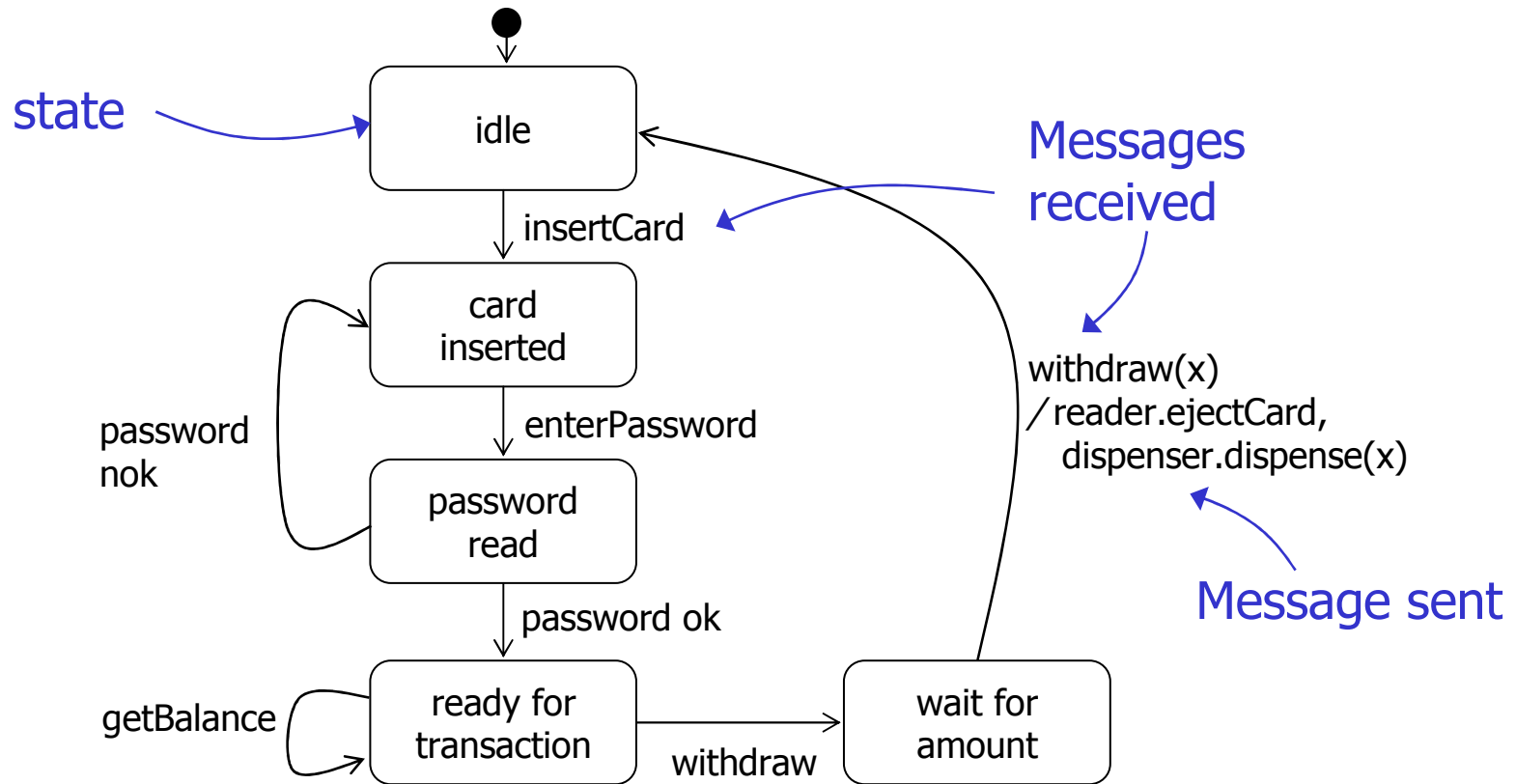
- View on inter-component interaction behavior
- Behavioral view
- Assignment of responsibilities
- Focus on temporal sequence

Collaboration Diagrams



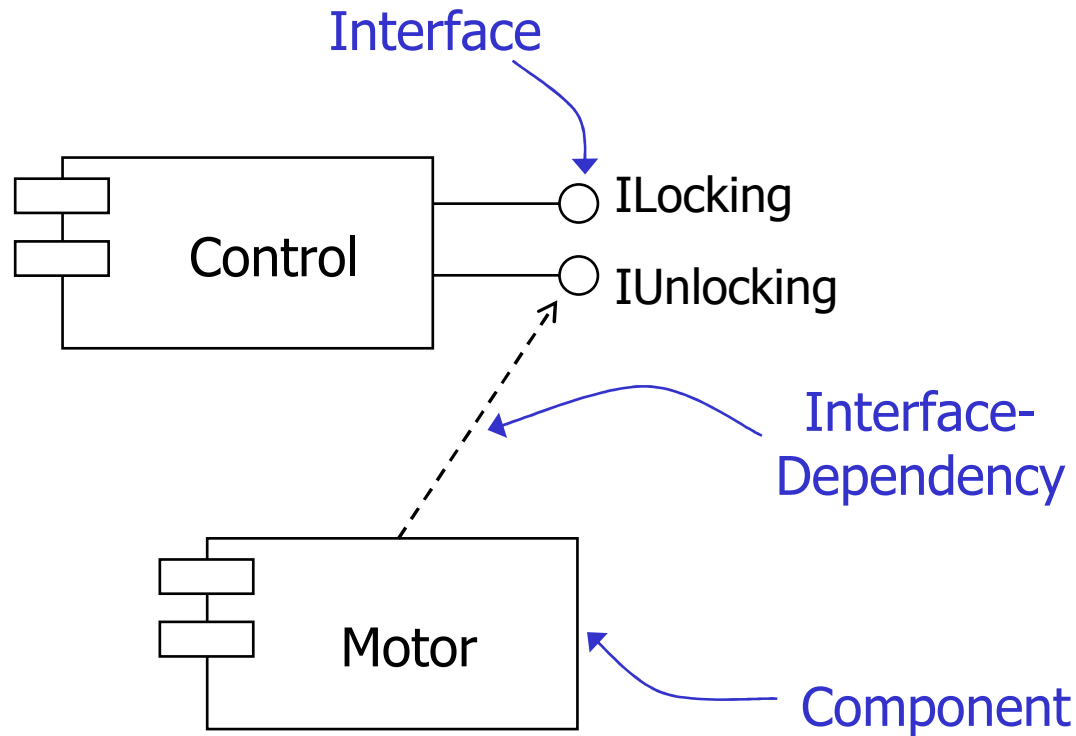
- View on inter-component interaction behavior
- Behavioral view
- Assignment of responsibilities
- Focus on component structure

Statecharts



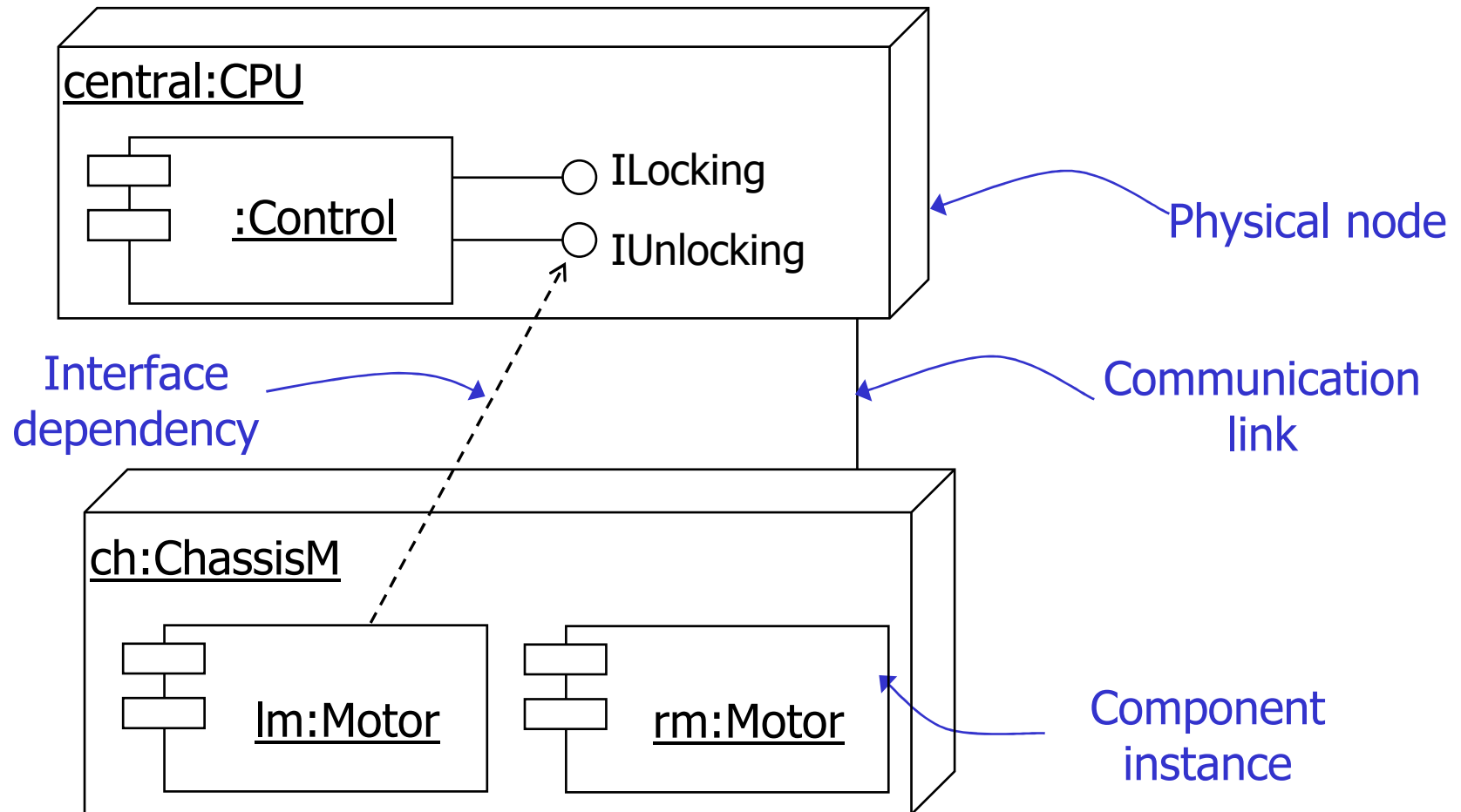
- View on local behavior of one object/component
- Behavioral view
- Focus on states/state changes

Component Diagrams



- View on components and interfaces
- What interfaces does a component **offer**?
- What interfaces does a component **use**?

Deployment Diagrams



- View on physical deployment structure
- Mapping from components and objects to physical nodes

UML Overview – Summary

- Graphical notation for
 - System structure
 - System behavior
- Covers multiple system views
 - Static structure (class/object/use case diagrams)
 - Distribution structure (component/deployment diagrams)
 - Interaction (sequence/collaboration diagrams)
 - State change (statecharts)
 - Use cases/services (Use case/activity diagrams)
- Tool for systematic development of component-oriented software architectures