Betriebsinformatik und betriebliche Informationssysteme
BIBI, 4(3)

1.1 UML Reloaded

5.JG - Schuljahr 2015/2016

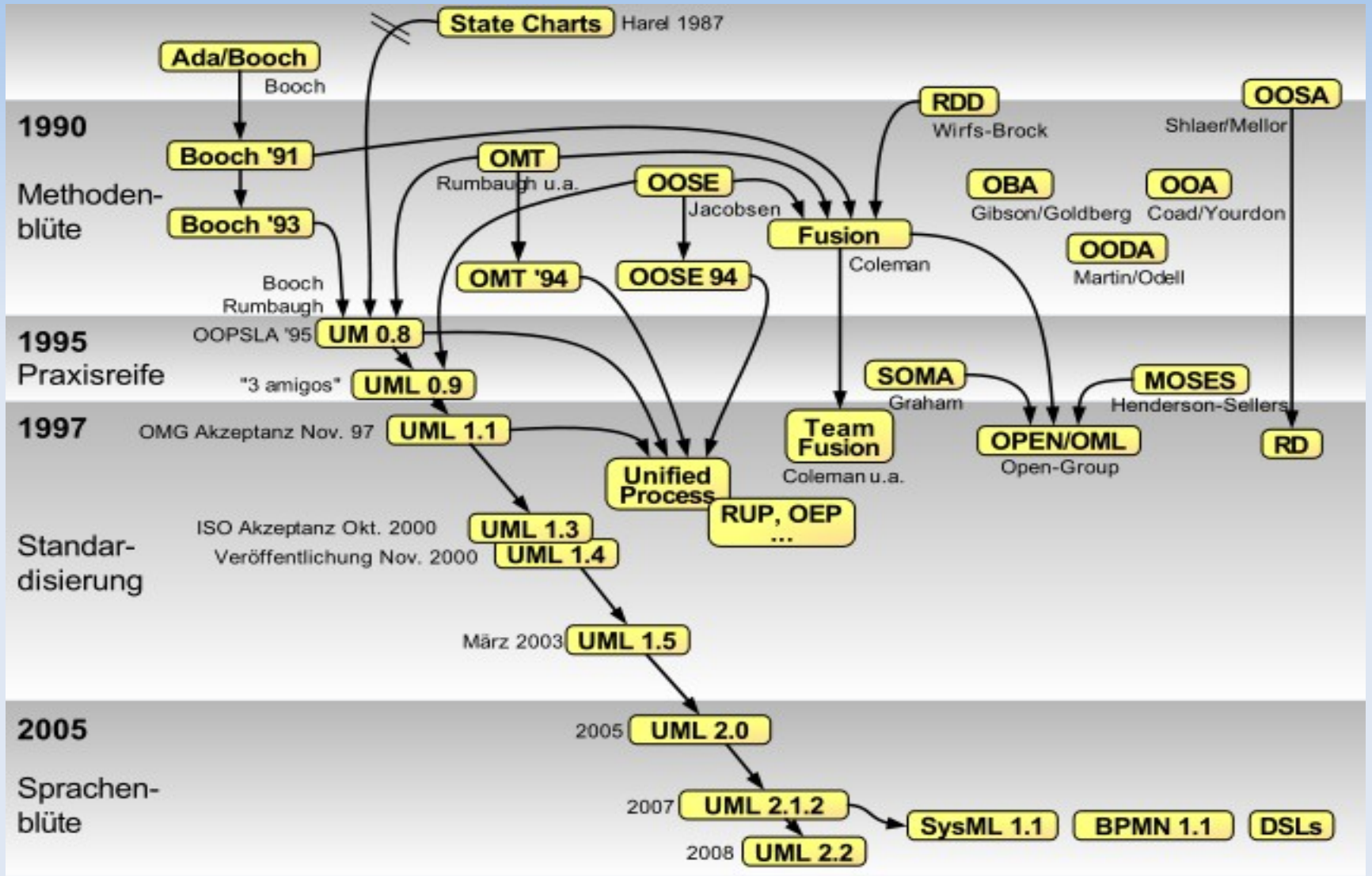
DI Diethard Kaufmann
DI Klaus Battlogg

UML-Reloaded / Versionen

- Version **1.x** ist in den 1990er Jahren entstanden
  Die Väter von UML, insbesondere Grady Booch, Ivar Jacobson und James Rumbaugh, auch „Die drei Amigos“ genannt, waren in den 1990er-Jahren bekannte Vertreter der objektorientierten Programmierung.
- 1999 Publizierung RFI (Request for Information) durch OMG für Version 2
- Schwieriger Prozess der Vereinheitlichung in mehreren Arbeitsgruppen und Taskforces bis 2004
- Am 21. Oktober 2008 wurde die **Beta 1 von UML Version 2.2** durch die OMG veröffentlicht, die wiederum im Februar 2009 in der finalen Version vorlag.
  Neu hinzugekommen ist in der Version 2.2 das Profildiagramm, um eigendefinierte Stereotypen-Sammlungen strukturieren zu können.
- Im Mai 2010 wurde **UML 2.3** veröffentlicht.
  Diese Version enthielt vor allem Bugfixes am Metamodell und Schärfungen der Semantik von Modellelementen im Spezifikationsdokument der UML
- ISO/IEC 19505 für Version 2.4.1
- Die aktuelle Version **2.5** wurde im **Juni 2015 veröffentlicht.**

[https://de.wikipedia.org/wiki/Unified_Modeling_Language]

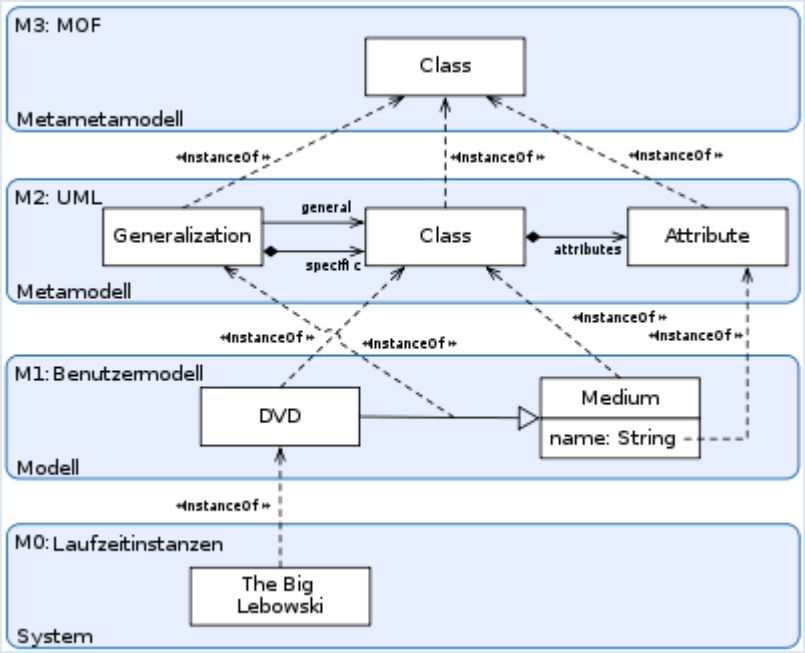
UML-Reloaded / Versionen



[https://de.wikipedia.org/wiki/Unified_Modeling_Language]

UML-Reloaded / Metamodellierung

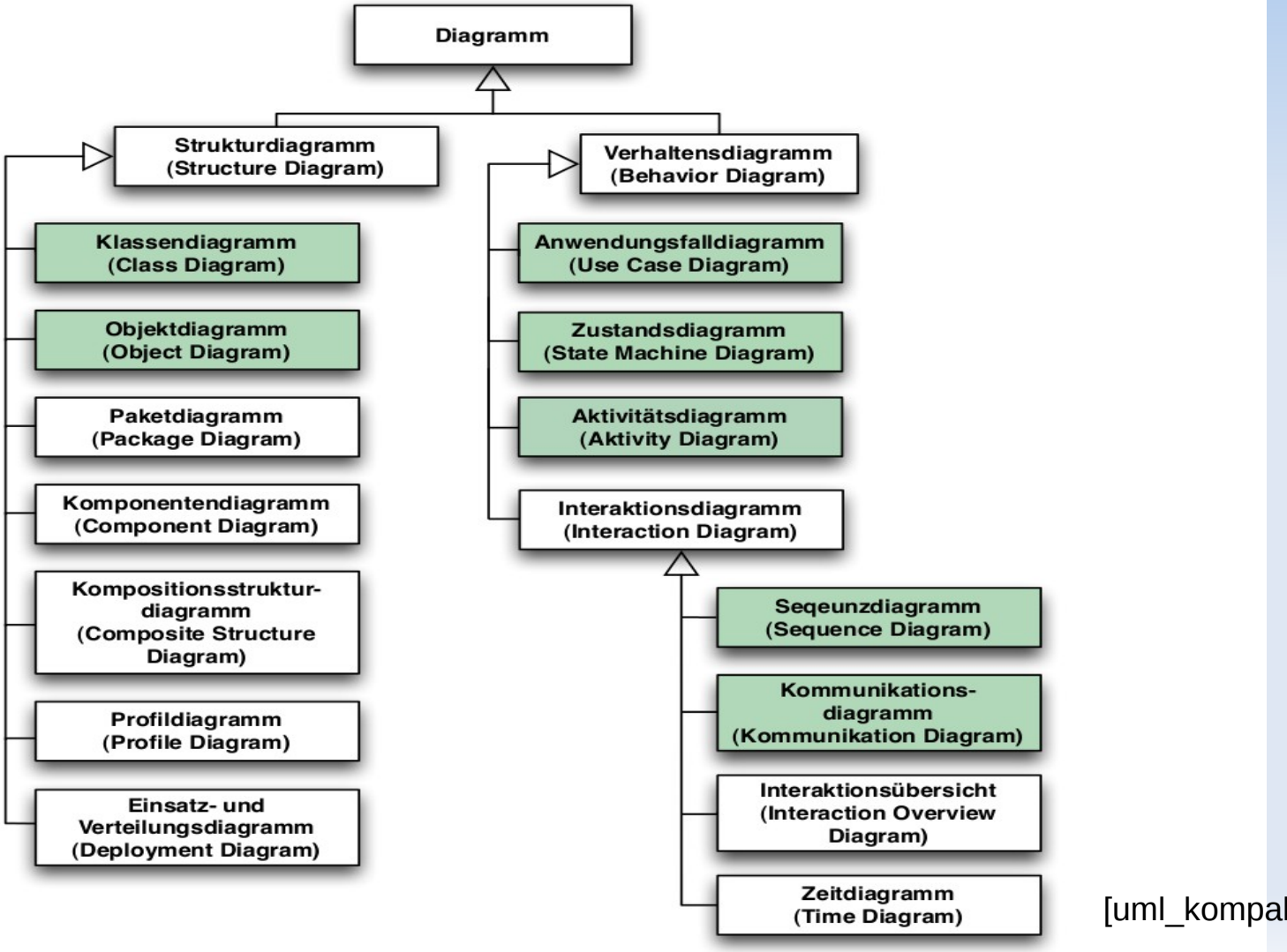
- Ähnlich wie sich natürliche Sprachen in Lexika oder Grammatiken selbst beschreiben, wurde auch UML als ein Sprachwerkzeug konzipiert, das sich mit einigen Sprachbestandteilen selbst erklärt.
- Die Sprachkonzepte sind dazu in vier Schichten M0 bis M3 gegliedert.
- Mit der Meta Object Facility (MOF) werden Modellelemente von UML2 spezifiziert und dadurch zum Beispiel mit dem Format Meta Interchange XMI austauschbar.



[https://de.wikipedia.org/wiki/Unified_Modeling_Language]

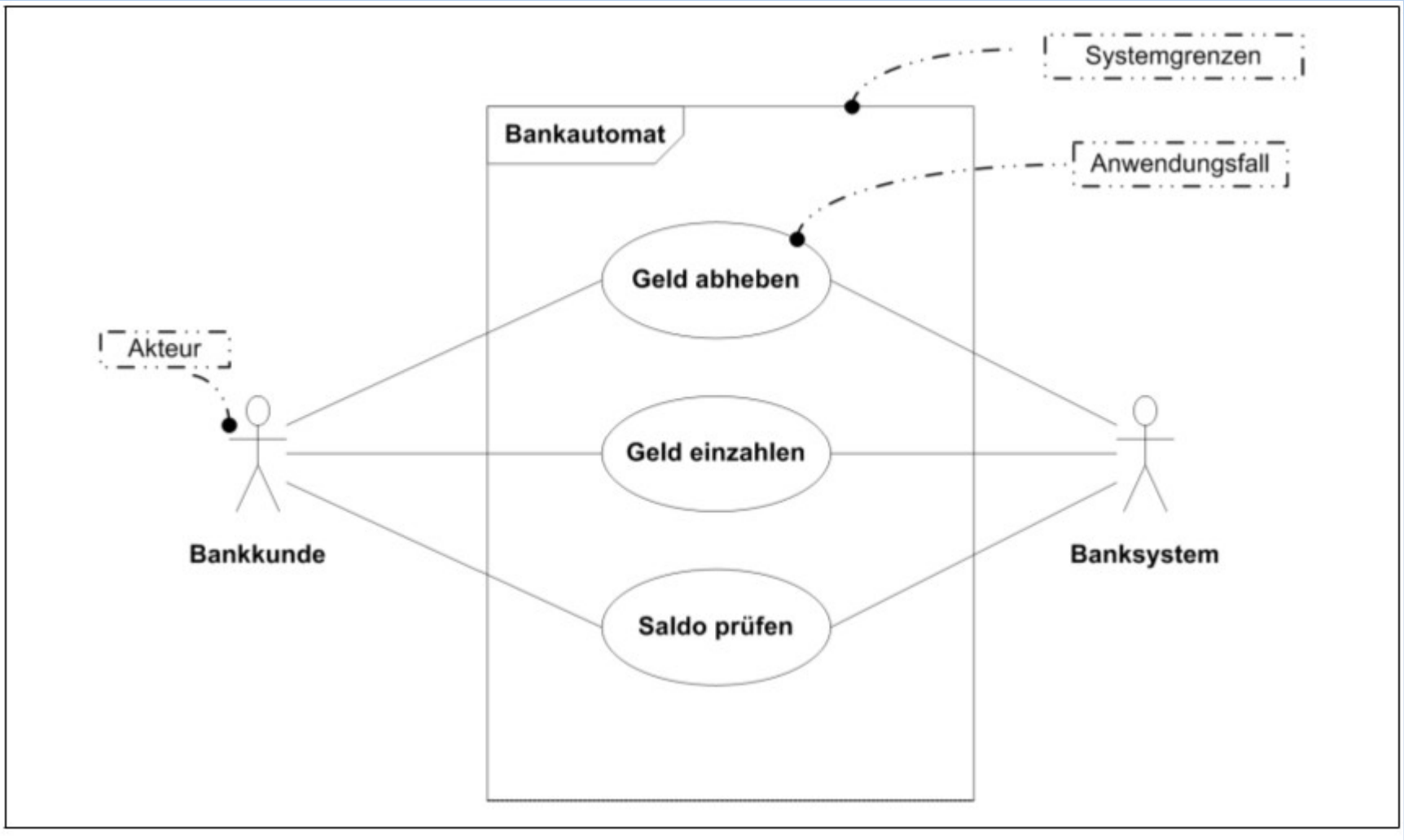
UML-Reloaded / Diagramme

- Aufteilung in 2 Hauptgruppen: Struktur- u. Verhaltensdiagramme



[uml_kompakt_thm.pdf]

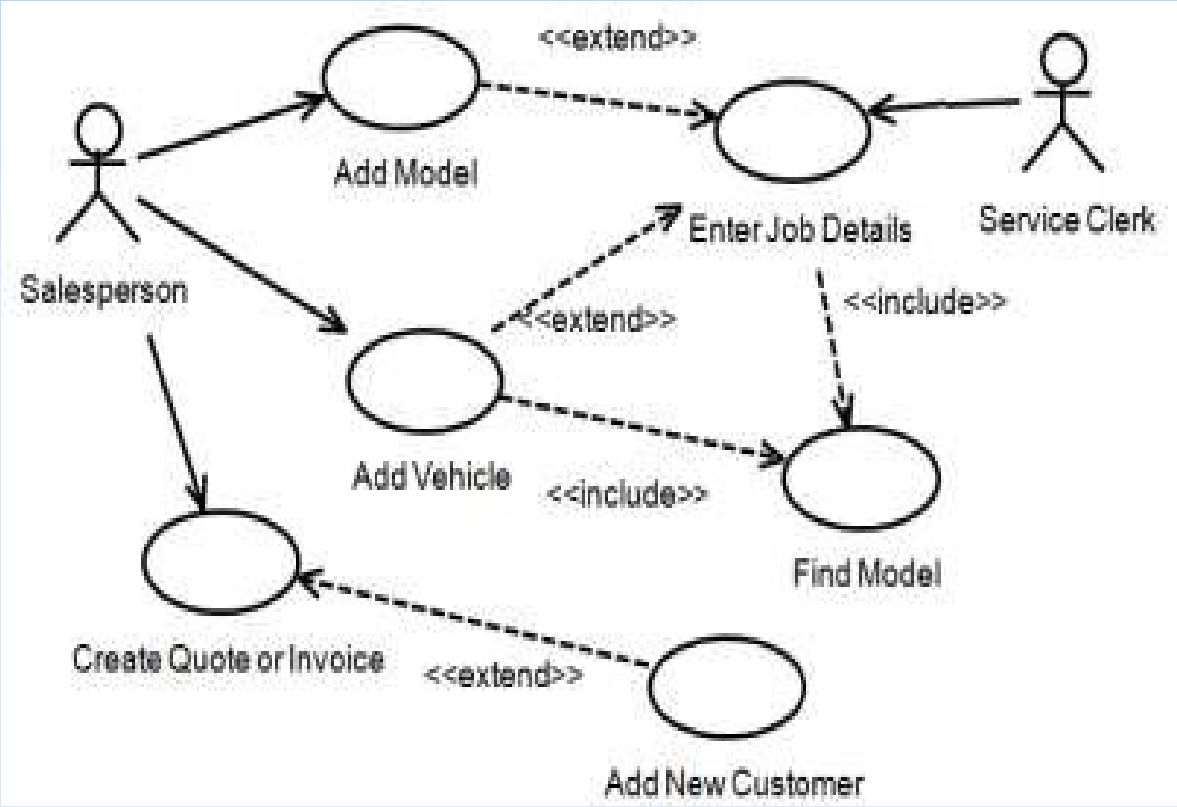
UML-Reloaded / Use Case



UML-Reloaded / Use Case

- The use case diagram shows the functionality of the system from an outside-in viewpoint.
- Actors (stick men) are anything outside the system that interacts with the system.
- Use Cases (ovals) are the procedures by which the actors interact with the system.
- Solid lines indicate which actors interact with the system as part of which procedures.
- Dashed lines show dependencies between use cases, where one use case is 'included' in or 'extends' another.

UML-Reloaded / Use Case



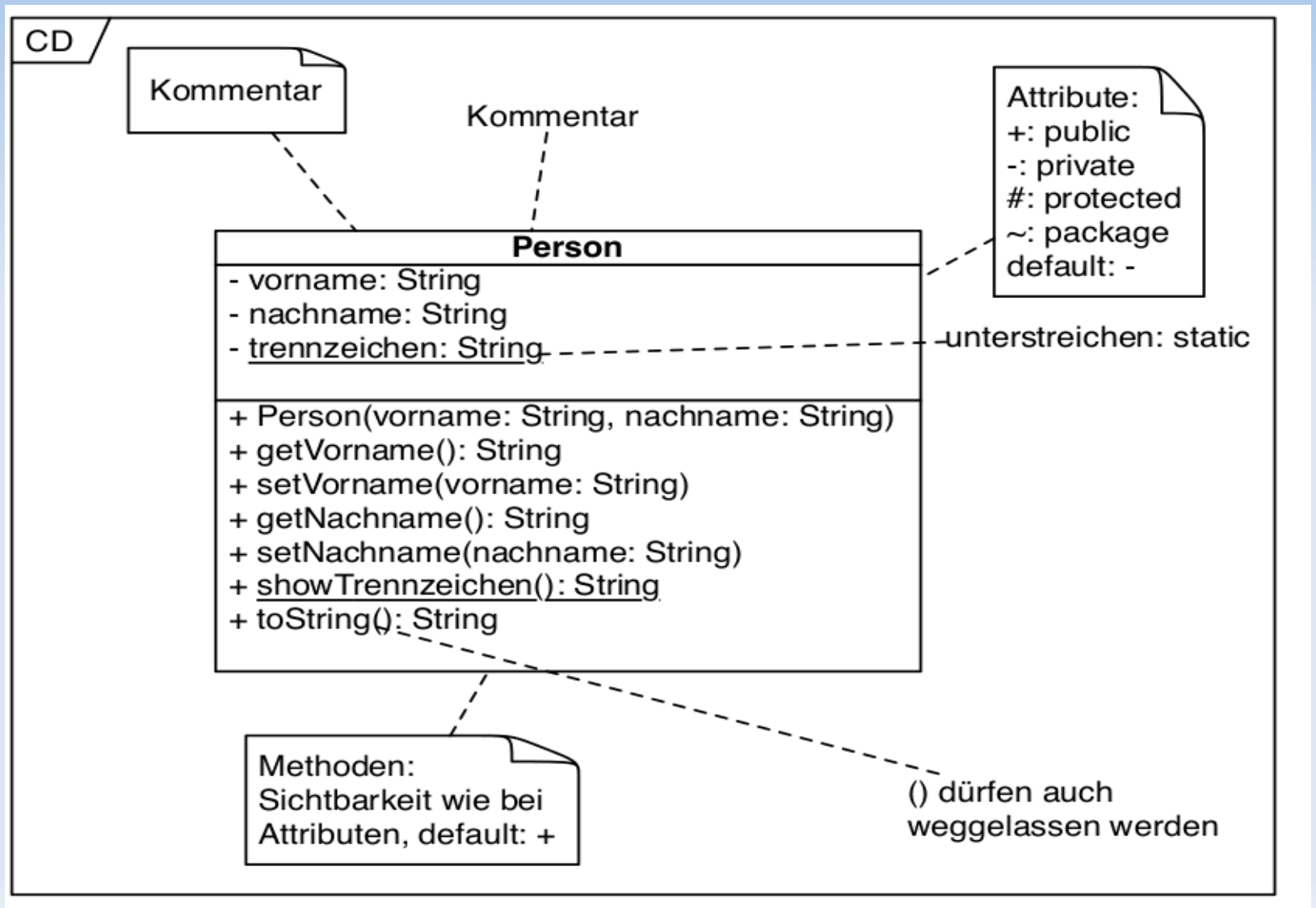
UML-Reloaded / Use Case

Beispiel einer textuellen Use-Case Beschreibung: Administrator einer SocialNetwork App kann bestimmte Aktionen (z.B: Message versenden) für selektierte Mitglieder durchführen

Field	Description
Name	Perform action on selected members
Primary Actor	Administrator
Preconditions	Administrator is logged in to the system.
Postconditions	Action is performed only on members that fit the specified criteria.
Main Success Scenario	1. Administrator specifies criteria of members on which to perform a certain action. 2. Administrator specifies an action to perform on those selected members. 3. Administrator selects the **Submit** button. 4. The system finds all members that match the specified criteria. 5. The system performs the specified action on all matching members.
Extensions	1a. Administrator has an option to preview those members who match the specified criteria before he or she specifies the action to be performed or before selecting the **Submit** button.
Frequency of Occurrence	Many times during the day.

[http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html]

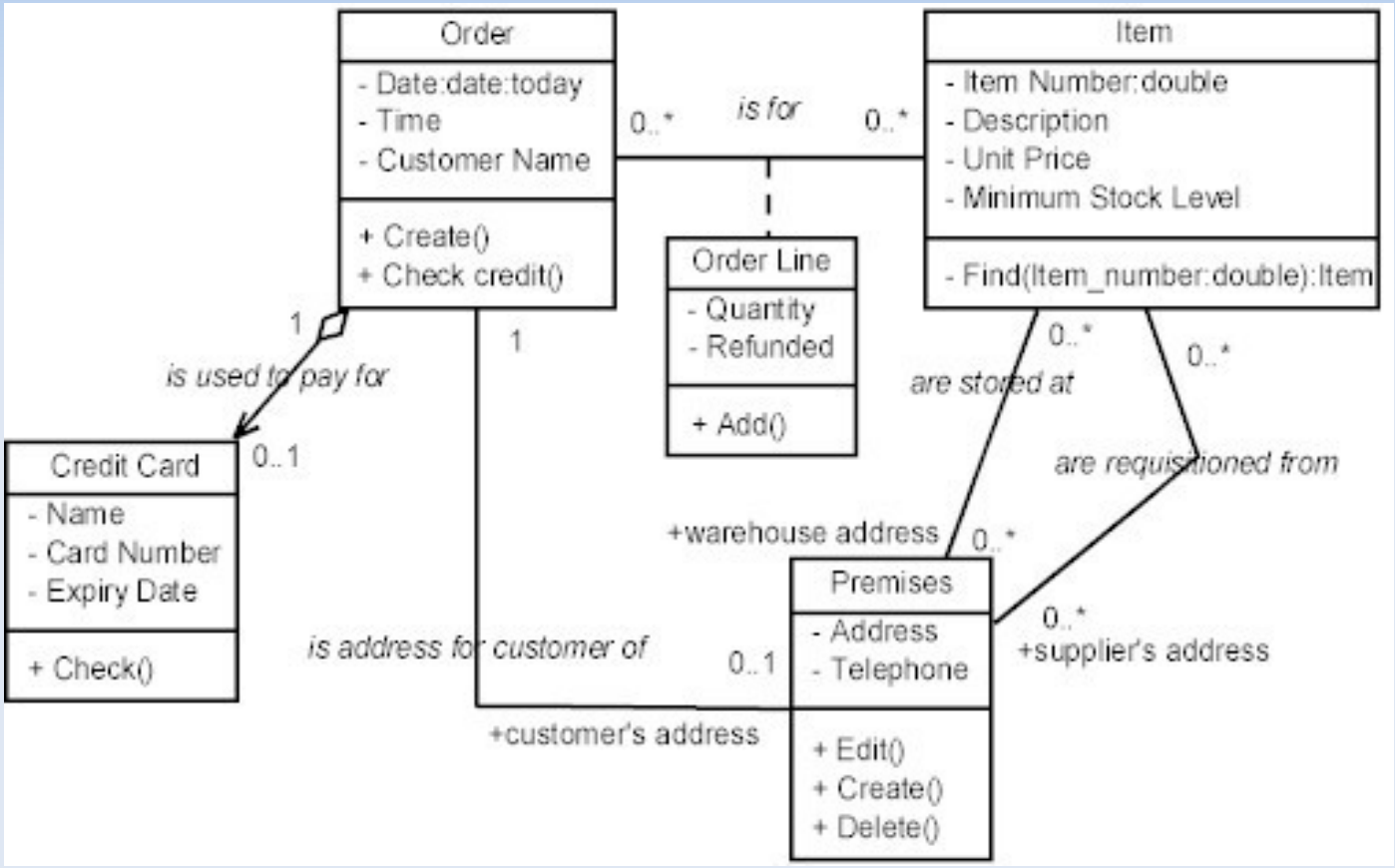
UML-Reloaded / Class Diagr.



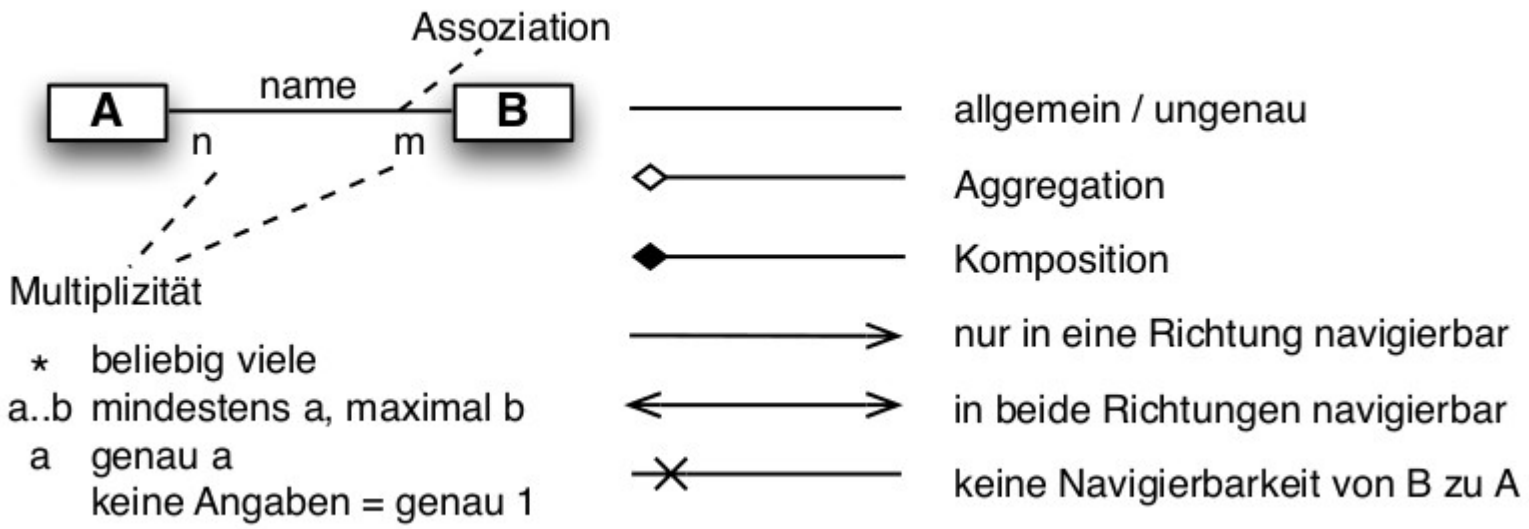
UML-Reloaded / Class Diagr.

- **Class diagrams** show the static structure of the systems. Classes define the properties of the objects which belong to them. These include:
- **Attributes** - (second container) the data properties of the classes including type, default value and constraints.
- **Operations** - (third container) the signature of the functionality that can be applied to the objects of the classes including parameters, parameter types, parameter constraints, return types and the semantics.
- **Associations** - (solid lines between classes) the references, contained within the objects of the classes, to other objects, enabling interaction with those objects.

UML-Reloaded / Class Diagr.



UML-Reloaded / Class Diagr.

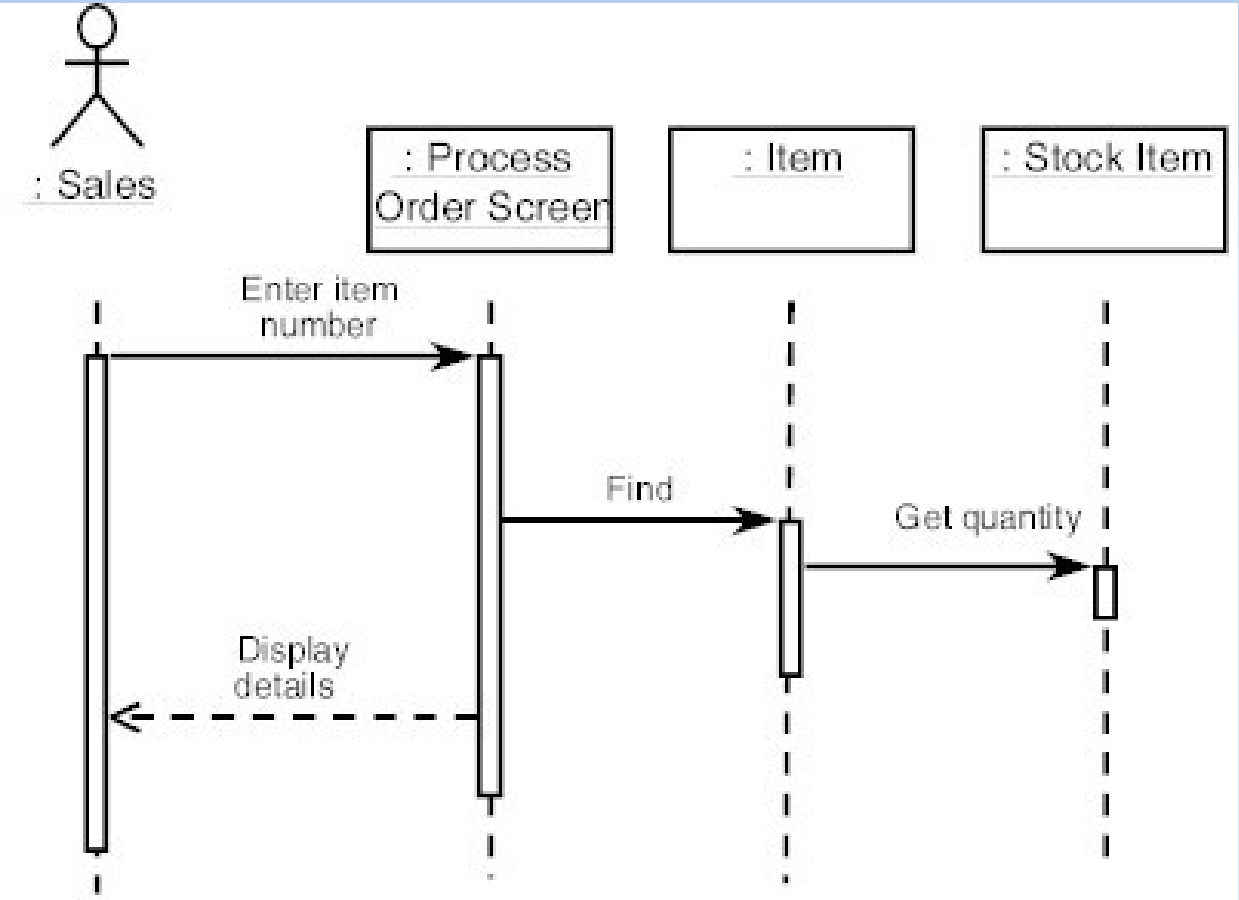


Auswahl an möglichen Assoziationen bzw. Multiplizitäten

UML-Reloaded / Sequence Diagr.

- **Sequence diagrams** show potential interactions between objects in the system being defined. Normally these are specified as part of a use case or use case flow and show how the use case will be implemented in the system. They include:
- **Objects** - oblong boxes or actors at the top - either named or just shown as belonging to a class, from, or to which messages are sent to other objects.
- **Messages** - solid lines for calls and dotted lines for data returns, showing the messages that are sent between objects including the order of the messages which is from the top to the bottom of the diagram.
- **Object lifelines** - dotted vertical lines showing the lifetime of the objects.
- **Activation** - the vertical oblong boxes on the object lifelines showing the thread of control in a synchronous system.

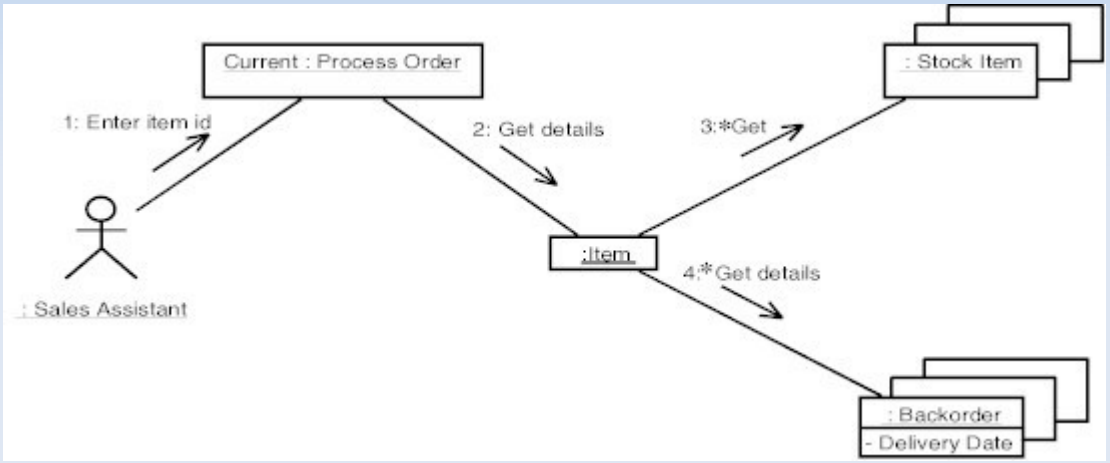
UML-Reloaded / Sequence Diagr.



UML-Reloaded / Communication Diagram

- **Communication Diagrams** show similar information to sequence diagrams, except that the vertical sequence is missing. In its place are:
- **Object Links** - solid lines between the objects. These represent the references between objects that are needed for them to interact and so show the static structure at object level. On the links are:
- **Messages** - arrows with one or more message name that show the direction and names of the messages sent between objects

UML-Reloaded / Communication Diagram

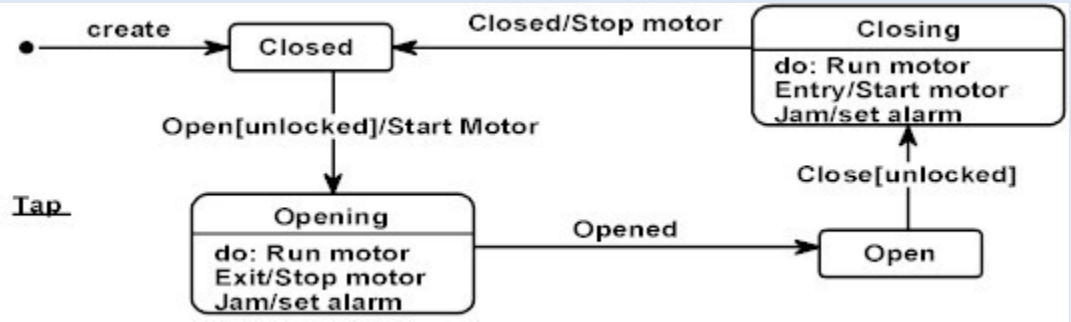


UML-Reloaded / State Diagram

- State Machine Diagrams, or Statecharts, are often used more in real-time embedded systems than in information systems, show for a class, the order in which incoming calls to operations normally occur and the conditions under which the operations respond and the response. They are a class-centric view of system functionality, as opposed to sequence and collaboration diagrams which are a use case-centric view of functionality. They include:
- States - oblong boxes which indicate the stable states of the object between events.
- Transitions - the solid arrows which show possible changes of state.
- Events - the text on the transitions before the '/' showing the incoming call to the object interface which causes the change of state.

UML-Reloaded / State Diagram

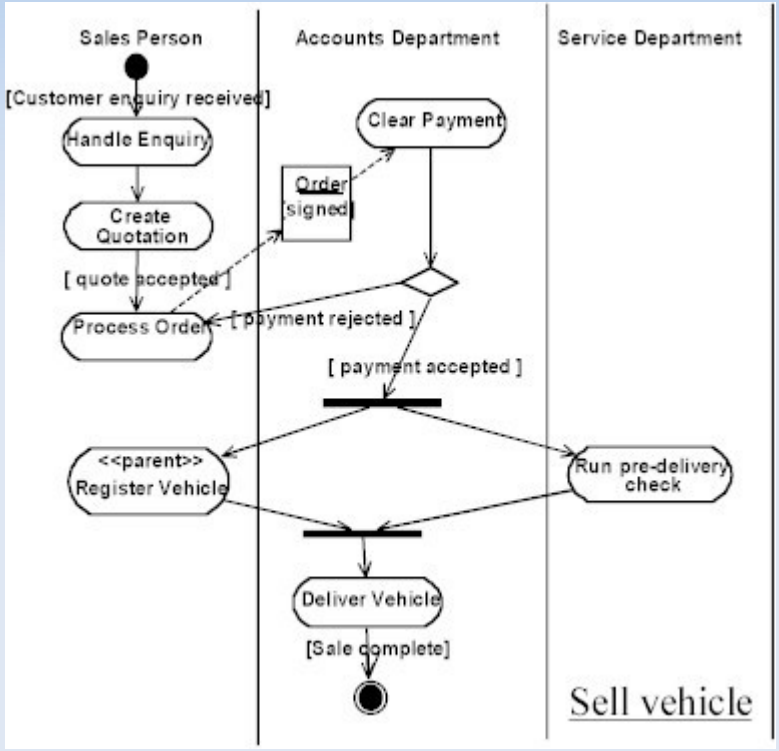
- Conditions - a boolean statement in square brackets which qualifies the event.
- Actions - the text after the '/' which defines the objects response to the transition between states.
- Extra syntax which defines state-centric functionality



UML-Reloaded / Activity Diagram

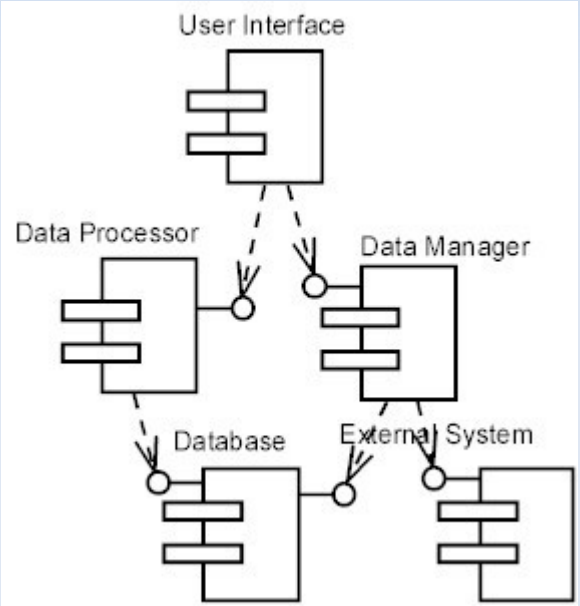
- A **UML Activity Diagram** is a general purpose flowchart with a few extras. It can be used to detail a business process, or to help define complex iteration and selection in a use case description. It includes:
- **Active states** - oblongs with rounded corners which describe what is done.
- **Transitions** - which show the order in which the active states occur and represent a thread of activity.
- **Conditions** - (in square brackets) which qualify the transitions.
- **Decisions** - (nodes in the transitions) which cause the thread to select one of multiple paths.
- **Swimlanes** - (vertical lines the length of the diagram) which allow activities to be assigned to objects.
- **Synch States** - horizontal or vertical solid lines which split or merge threads (transitions)

UML-Reloaded / Activity Diagram



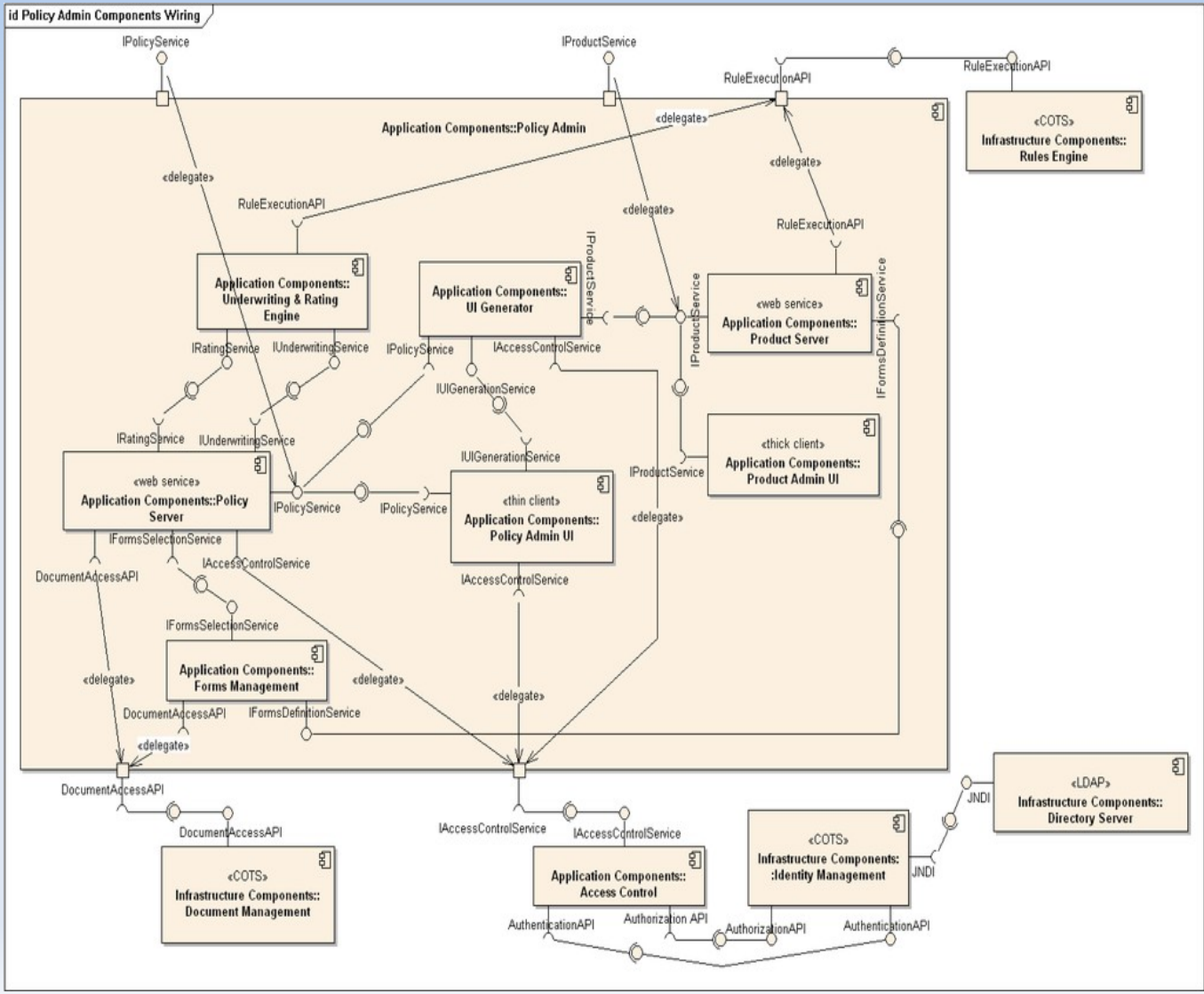
UML-Reloaded / Component Diagram I

- Component Diagrams show the types of software components in the system, their interfaces and dependencies.



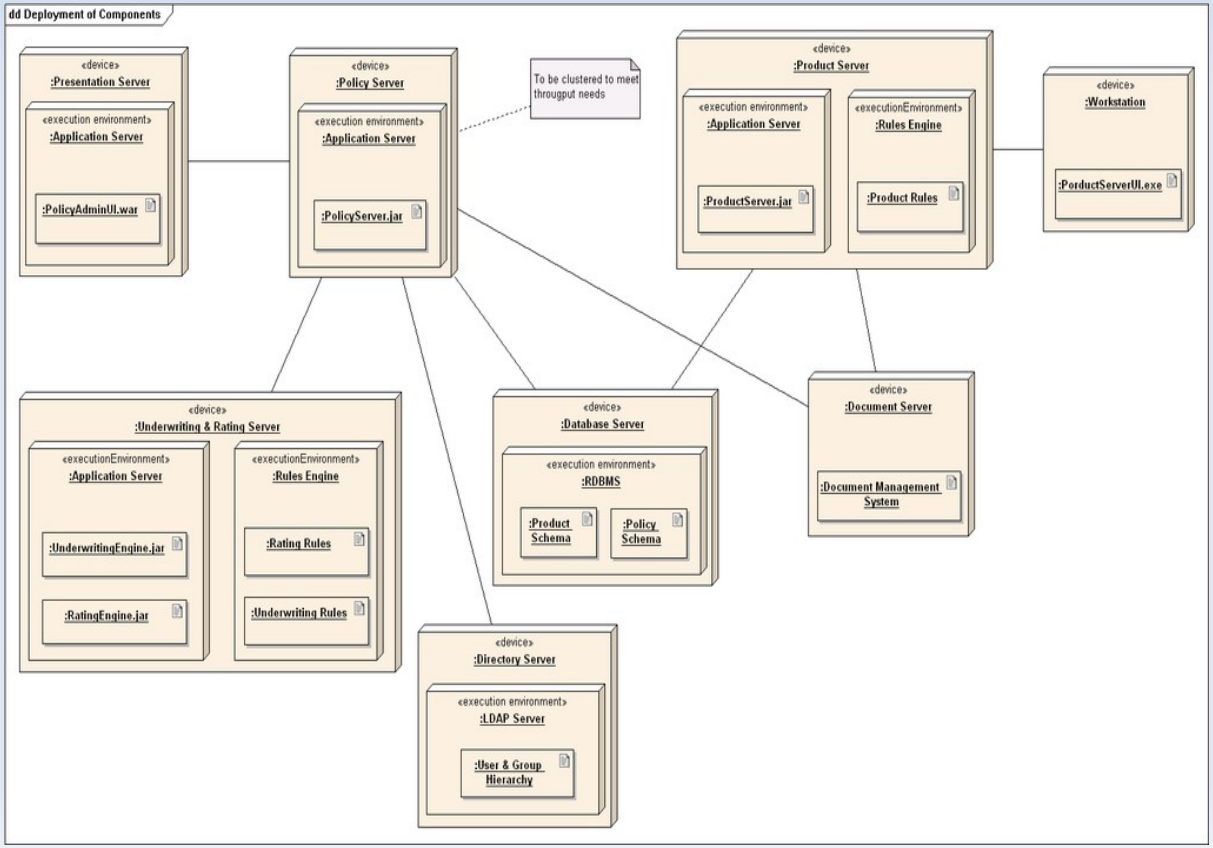
UML-Reloaded / Component Diagram II

- Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component
- The example above illustrates what a typical Insurance policy administration system might look like. Each of the components depicted in the above diagram may have other component diagrams illustrating their internal structure.

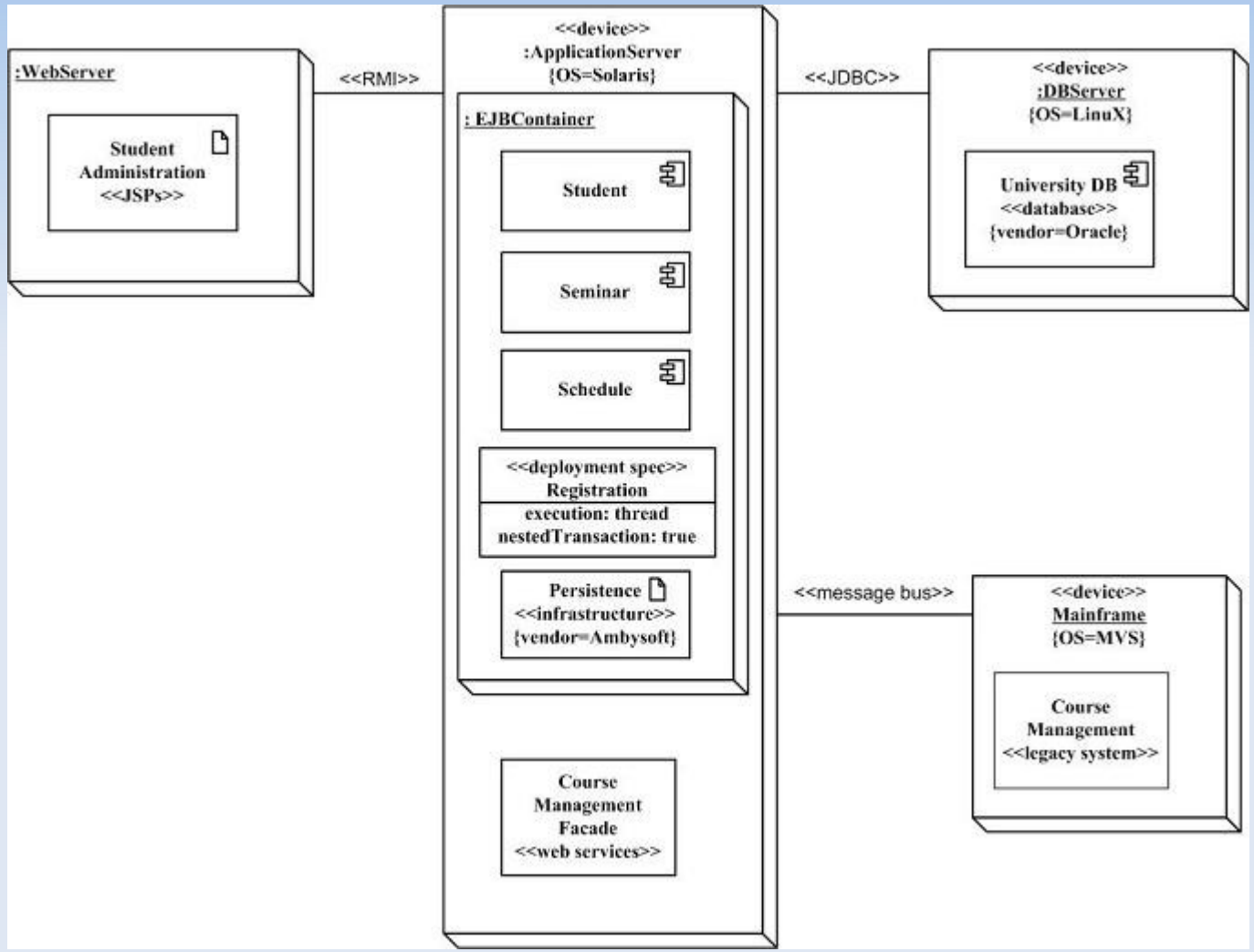


UML-Reloaded / Deployment Diagrams

- A deployment diagram in the Unified Modeling Language models the physical deployment of artifacts on nodes.[1] To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).



UML-Reloaded / Deployment Diagrams



[http://www.agilemodeling.com/artifacts/deploymentDiagram.htm]

Literatur/Quellen

- OOA + OOD / UML
  - http://www.uml-diagrams.org/
  - “Objektorientierte Softwareentwicklung” [Bernd Oestereich]
    http://www.oose.de/
  - http://de.wikipedia.org/wiki/Unified_Modeling_Language
  - Allen Holub's UML Quick Reference
    Version 2.1.2 (2007/08/10)
    http://www.holub.com/goodies/uml/
  - http://www.cragsystems.co.uk/uml_tutorial/
  - https://homepages.thm.de/~hg11260/mat/uml.pdf
  - http://www.it-ebooks.info/book/154/ [UML in a Nutshell]