Architektur von Anwendungssystemen – Zusammenfassung

Sommersemester 2019

Lukas Heiland – last updated:

15. April 2019

Inhaltsverzeichnis

1 Definitions/Notions

1.1 Architecture

The architecture of an IT system is the structure or structures of the system which comprise software and hardware components, the externally visible properties of those components, and the relationships among them.

- Architecture isn't simply *good* or *bad*

- Architectue is fit or unfit *for a purpose*

1.1.1 Goals of an Architecture

- Producing a framework to support the development of software

- Creating an integration platform for future enhancements

- Producing the interface definitions for collaboration of components

1.1.2 Importance of Architecture

If the size and complexity of a software system increase, the global structure of the system becomes more important than the selection of specific algorithms and data structures.

1.1.3 Architecture vs. Design

An architecture provides a framework and a 'set of rules' for the act of designing a particular thing. So there can be many individually designed instances of each particular architectural style.

1.1.4 Architecture Levels

Conceptual Architecture

- direct attention at an appropriate decomposition of the system without delving into details

- provides a useful vehicle for communicating the architecture to non-technical audiences, such as management, marketing, and users

- consists of the Architecture Diagram (without interfaces) and an informal component specification for each component (see: ADL)

Logical Architecture

- adds precision, providing a detailed "blueprint"from which component developers and component users can work in relative independence

- incorporates the detailed Architecture Diagram (with interfaces)

Execution/Physical Architecture

- Shows the mapping of components onto the threads, processes, (virtual) machines, . . . of the physical system

- created for distributed or concurrent systems

1.2 Architect

Some interesting definitions/quotations/etc. from various slides. . .

- The IT Architect defines (i.e. architects) solutions to client business problems through the reasoned application of information technology.

- The task of an architect is reduction of complexity to orders of magnitude that can be realistically handled.

- The definition of Vitruvius ($\approx 25 B.C.$) adds, that an architect (of any kind) should have a lot of general knowledge.

The architect is the advocate of the client.

1.2.1 Where do Architects get ideas from?

Studies of work of other architects is key!
Importance of reference architectures, patterns, styles

1.2.2 Architectural thinking

Architectural thinking is based on basic architectural principles:

- Separation of concerns

- Information Hiding

- Design by interface

- Separation of interface and implementation

- Partitioning/distributing responsibilities

Architectural thinking involves

- Looking at the solution from the direction of requirements, not technology

- Understanding all aspects of the requirements (functional and non-functional)

- Understandign all aspects of the solution (functional and non-functional)

- Using reference architectures and patterns whenever appropriate

- Compromising and balancing; every solution to a requirement will cause other problems

1.3 System

Composition of parts into a new whole which represents via the collaboration of the parts more than the sum of its parts.

1.3.1 Emergence

This is a central aspect of Systems: Emergence is the appearance of properties of a system which none of its constituents has; i.e. a shelf: it is comprised of wooden planks and screws, and after you finished building it, you can put stuff on it. This is emergence: the planks and screws themselves did not offer the possibility to store things, it emerged from the system that is called *a shelf*.

1.4 Views

- Views = different models of a single system. Can be built by abstraction

- Architecture consists of multiple different model descriptions of a single building. Different model descriptions target different participants (stakeholders) of the project:

$$\text{Ground plan} \mapsto \text{Decorator}$$
$$\text{Wiring} \mapsto \text{Electrician}$$
$$\text{Plumbing} \mapsto \text{Plumber}$$
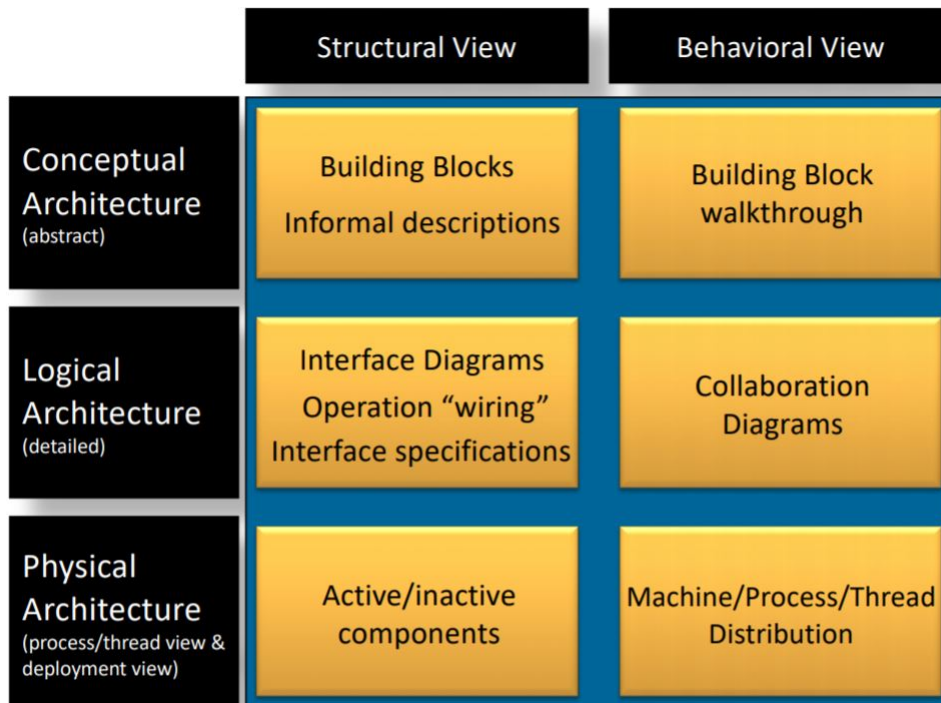$$\ldots \mapsto \ldots$$

1.4.1 Structural and Behavioral Views

These are used to enhance understandability of the architecture's levels (see 1.1.4).

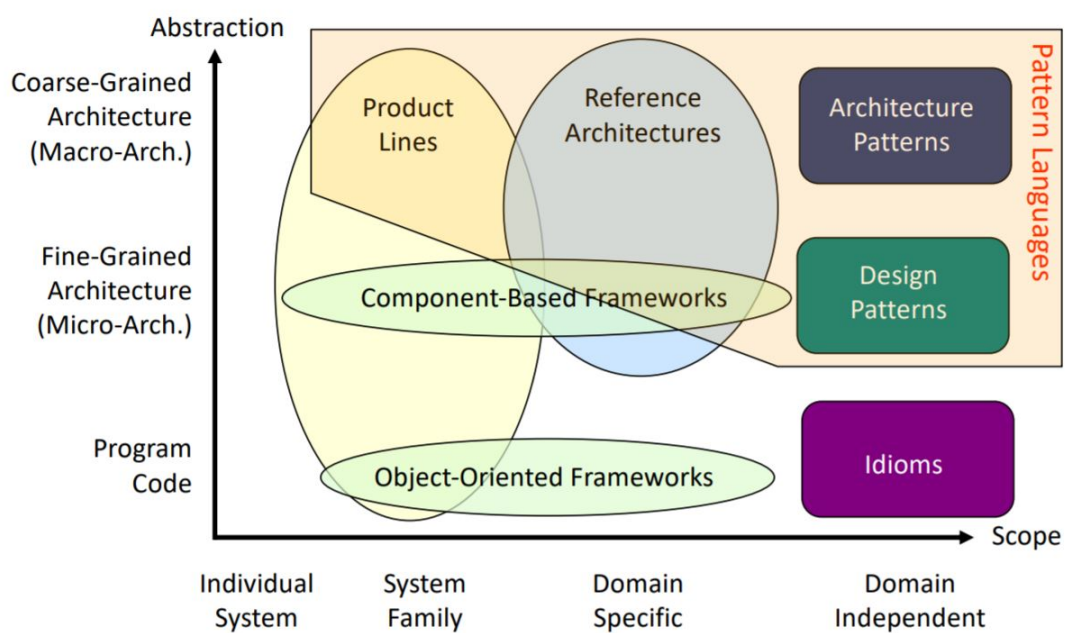**Structural Views**   consist of the Architecture Diagram, and Component and Interface Specifications

Behavioral Views   Contain Component Collaboration or Sequence Diagrams; they answer the question *'How does this work?'*

1.5 Architectural Levels and Views Together

	Structural View	Behavioral View
Conceptual Architecture (abstract)	Building Blocks Informal descriptions	Building Block walkthrough
Logical Architecture (detailed)	Interface Diagrams Operation "wiring" Interface specifications	Collaboration Diagrams
Physical Architecture (process/thread view & deployment view)	Active/inactive components	Machine/Process/Thread Distribution

1.6 Reuse

Classification

2 Diagrams and Styles

2.1 Basic elements of an architecture

2.1.1 Components

Components are the result of decomposition of a system.

2.1.2 Connectors

Connectors connect components.

2.1.3 Constraints

Components must be constrained to provide that

- the required functionality is achieved

- no functionality is duplicated

- the required performance is achieved

- the requirements are met

- modularity is realized (e.g. which modules interact with the operating system)

2.1.4 Rationales

3 Model Driven Architecture (MDA)

3.1 Origins

- There are so many (not necessarily interoperable) technologies

- These evolve and get obsolete very quickly

$\Longrightarrow$ desire to have ones business logic (processes, rules, ...) to be as independent as possible from any one technology (*future-proof* business logic)

3.2 Terminology

3.2.1 Architecture

specification of the parts and connectors of the system and the rules for the interactions of the parts using the connectors.

3.2.2 Platform

Set of subsystems/technologies that provide a coherent set of functionality through interfaces and specified usage patterns.
Any subsystem that depends on the platform can use it without concern for the details of how the functionality provided by the platform is implemented.

3.2.3 Implementation

A specification which provides all the information needed to construct a system and to put it into operation.

3.3 MDA Models

3.3.1 Computation independent model (CIM)

- a.k.a. *domain model* or *business model*

- focuses on the system and its environment; details of the structure of the system are hidden or undetermined

- specified using a vocabulary that is familiar to the practitioners of the domain in question

- may hide information about the use of automated data processing systems

3.3.2 Platform Independent Model (PIM)

Exhibits platform independence and is suitable for use with a number of different platforms of similar type.
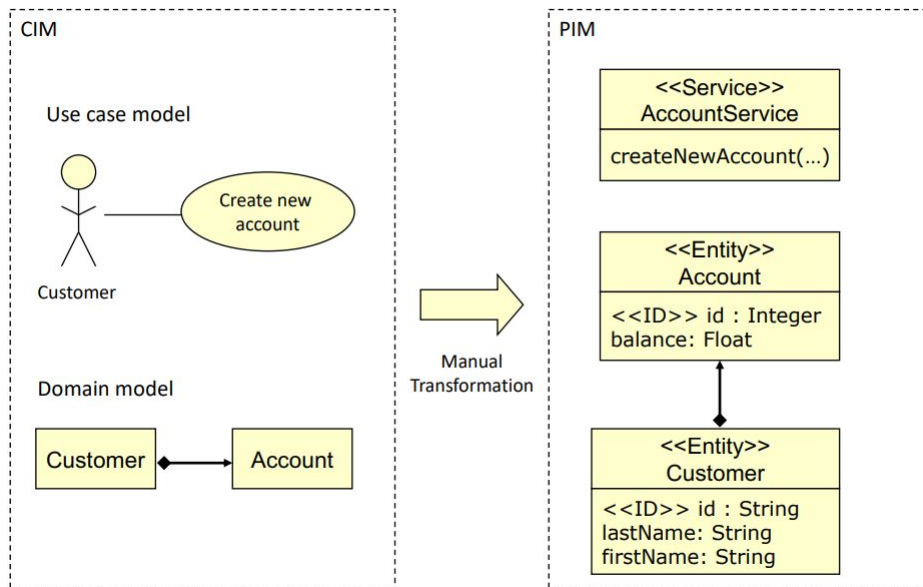

Abbildung 3.1: How to convert a CIM to a PIM

3.3.3 Platform Specific Model (PSM)

Combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.
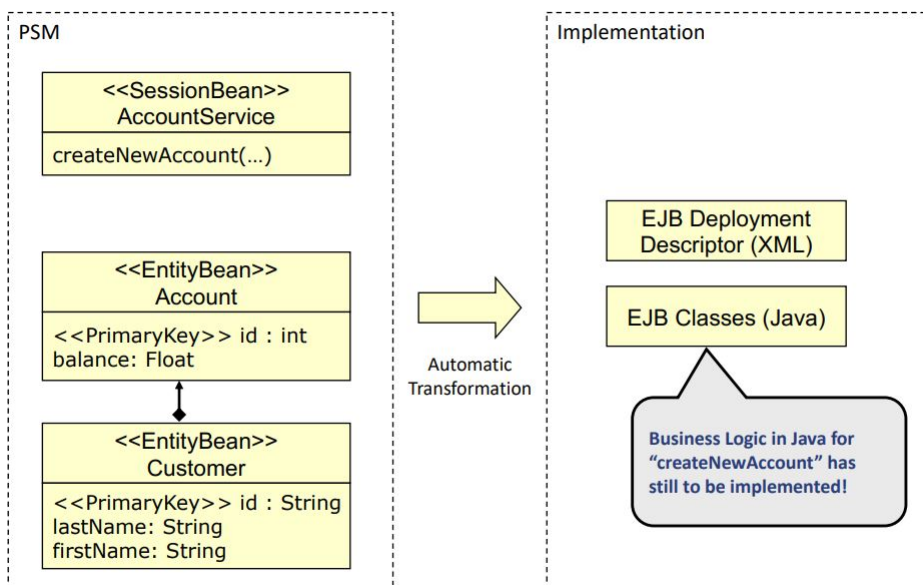

Abbildung 3.2: Conversion from a PSM to an implementation

3.3.4 Platform Model (PM)

Provides a set of technical concepts, representing the different kinds of parts that make up a platform and the services provided.
Influences the way a PIM is mapped to a PSM.

3.4 Model transformation

This is the process of converting one model to another model of the same system. It is done by a process called **mapping**. An MDA mapping is a set of specifications for transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping.
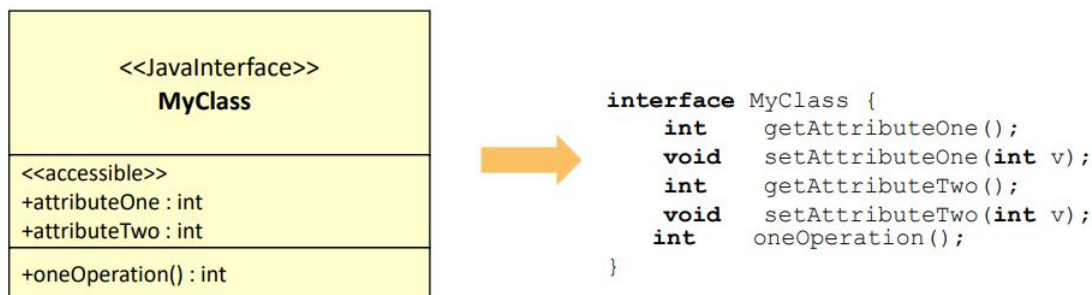

Abbildung 3.3: Model-to-code transformation – an example for a mapping

3.4.1 The MDA Pattern

The MDA pattern includes at least

- a PIM

- a PM

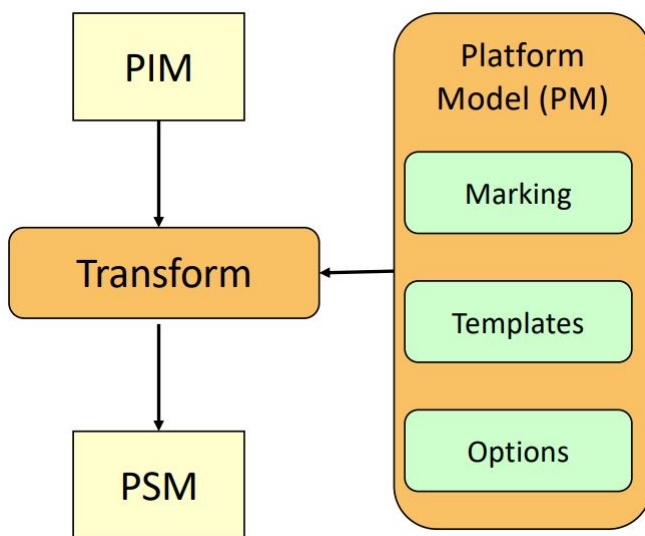- a Transformation

- a PSM

Abbildung 3.4: The MDA pattern (minimal form)

As shown in the figure, the PM influences the nature of the mapping. The PM does this through three concepts (first two described more elaborately in a later section):

1. Marking

2. Transformation Templates

3. Options: Adjust the transformation globally (similar to compiler options)

3.4.2 Mapping Concepts

Metamodel Mapping

mapping gives rules and/or algorithms how types of the PIM metamodel are to be transformed to types of the PSM metamodel. Not applicable if PSM has no metamodel specified, e.g. in Model-to-Code transformations, where the PSM is Code.

Marking

A mark represents a concept in the PSM, which can be applied to an element of the PIM to indicate how that element is to be transformed:
if more than one PSM-alternative for something in the PIM exists, the mark indicates which alternative should be taken.
Also, different platform mappings may require different markings.
Example from the lecture: In J2EE, there are two types of EJBs. Marking defines which one to use.

Transformation Templates

Parameterized models that specify particular kinds of transformations (a bit like design patterns). Typically creates groups of elements out of one element in the PIM.
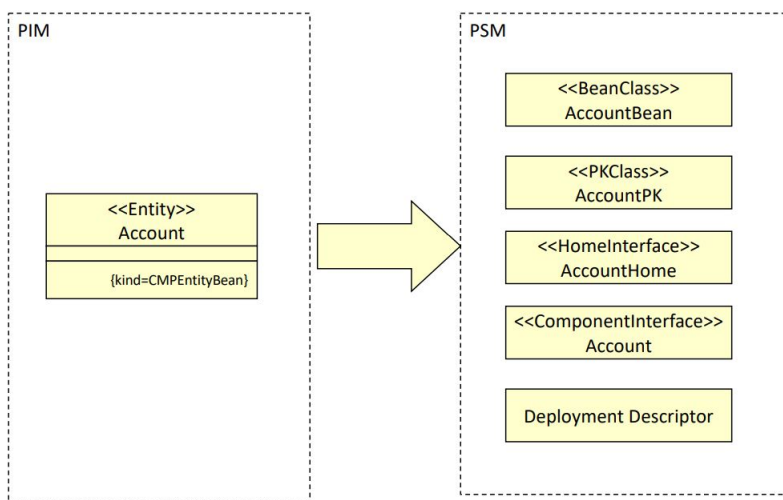

Abbildung 3.5: Example for a Transformation template

Multi-staged Transformation

= Applying MDA Pattern in a cascade. The MDA pattern can (and usually has to) be applied several times in succession; the output PSM from one iteration will then become a PIM for the next one
$\implies$ PIM and PSM are relative concepts, they depend on the platform in use

Multi-platform Transformation

Many systems are (can be) built on more than one platform. Multi-platform Transformation means using different PMs to transform a PIM into different PSMs with parts of the system on different platforms with connections/adapters between them.

3.5 Advantages of MDA

- Each model is independent of the rest

- Software development gets increasingly closer to model transformation

- Transformations can be automated

- We gain modularity, flexibility, and facilitate evolution

- Application models capturing business logic and intellectual properties (IP) become **corporate assets**, independent from the final implementation technologies

3.6 MDA Standards

3.6.1 Meta Object Facility (MOF)

Meta-Meta-Model for the construction of metamodels in MDA
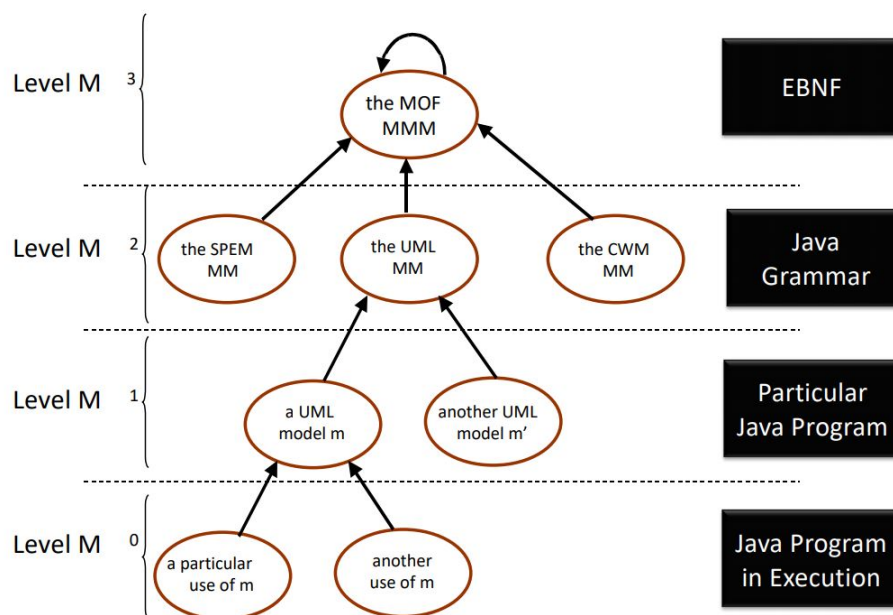
3.6.2 Unified Modeling Language (UML)

UML is central for MDA because many tools are based on UML and its extension capabilities (UML Profiles). From version 2.0 on, UML is formally defined via the MOF.

Extending UML

UML can not be complete, because it's not feasible to specify every details. There are two ways to extend UML/MOF:

1. Heavyweight: completely new meta-model based on MOF (Not automatically supported by modeling tools); essentially creating a new modeling language from MOF

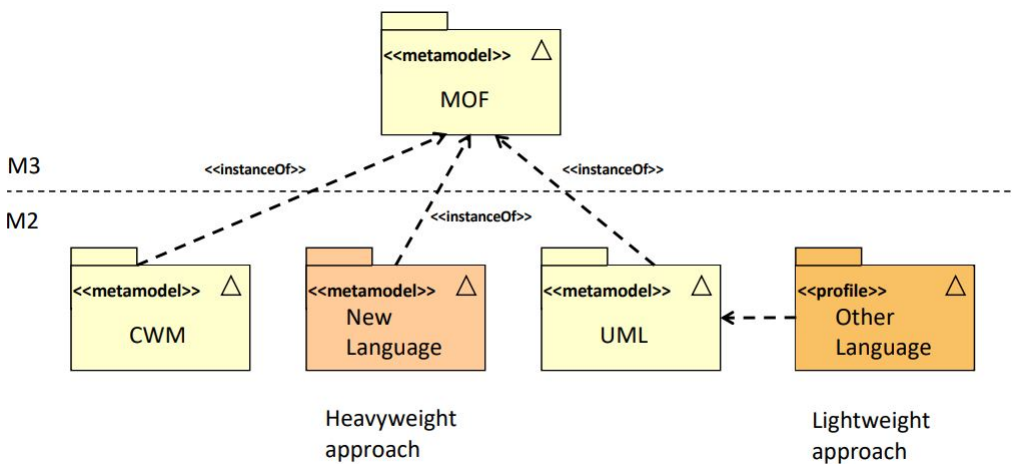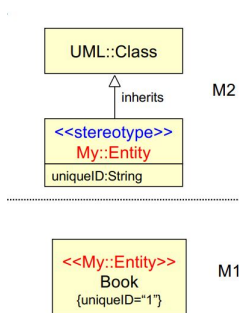2. Lightweight: Extension based on the UML Metamodel or with UML Profiles

Abbildung 3.6: Comparison of the two extension approaches

Extension based on the metamodel

Two new features (can be interpreted by code generators!!!!1!!1!!!):

- Stereotype: represented by «...»; Specifies the metaclass

- Tagged Value: represented by {...}; Specifies an attribute of the metaclass

UML Profiles

Mostly used to specialize UML for specific domains, when there is no need to change UML metamodel and semantics. They are an excellent mechanism for defining MDA 'Platforms'. A UML profile consists of:

- Stereotypes: Used to refine meta-classes (or other stereotypes) by defining supplemental semantics

- Tagged values: Attributes of stereotypes with user-defined semantics; Rendered as tagged values in the model in which the stereotype is used

- OCL constraints: Predicates (e.g., OCL expressions) that reduce semantic variation; Can be attached to any meta-class or stereotype

There is a UML language construct for an extension: a filled inheritance arrow.
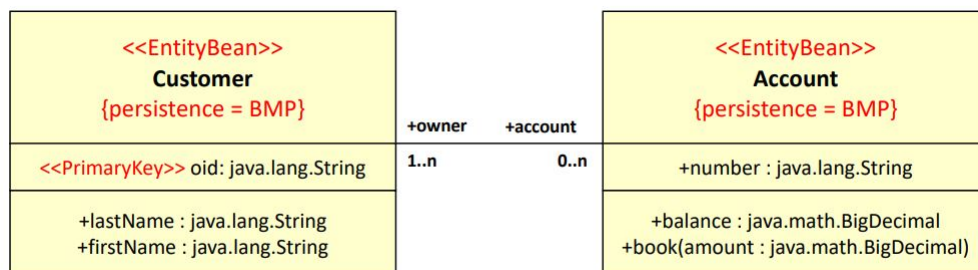An extension conforming to the UML standard must not violate the standard UML semantics

Abbildung 3.7: Usage of the EJB profile