HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN MÜNCHEN

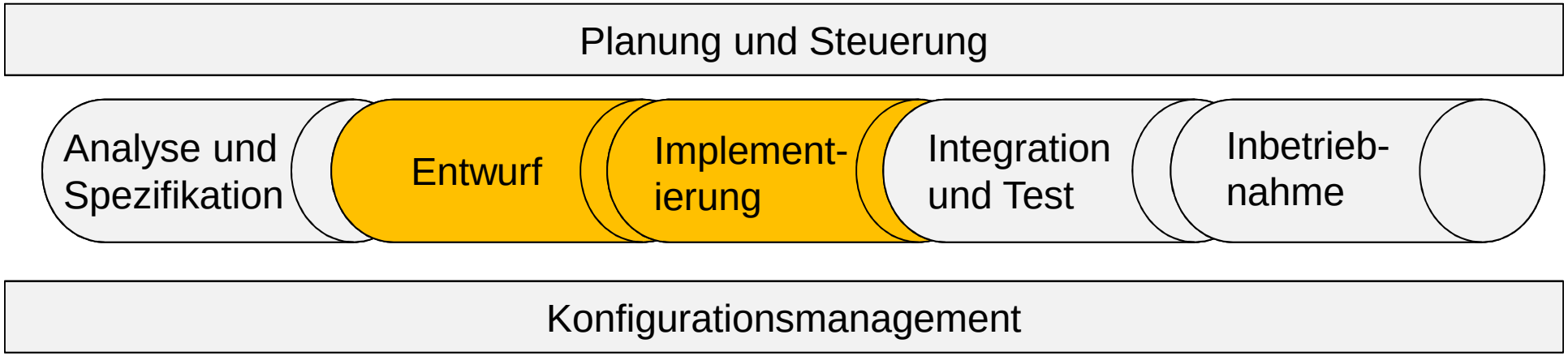
QAWARE SOFTWARE ENGINEERING

Software Engineering II

Softwarearchitektur

Montag, 15. April 2013
Dr. Josef Adersberger
Version 1.0

Wir sind heute auf unserer Landkarte des Software Engineering bei dem Entwurf.



Was ist Softwarearchitektur

- Die Softwarearchitektur beschreibt die Komponenten eines Systems und ihr Zusammenspiel. Eine Softwarearchitektur wird im Rahmen des Entwurfs entwickelt.

- Prinzipien
  - **Separation of Concerns**: Zusammenbringen was zusammen gehört (Kohäsion) und trennen, was unterschiedlich ist (Entkopplung).
  - **Problem Hiding**: Verstecken von Komplexität. KISS: Keep it simple and stupid.
  - **Emergent Design**: Variabilität ermöglichen. Erosion der Softwarearchitektur verhindern.

- Schnittstelle zur Spezifikation für die wichtigsten Architektursichten
  - **Technische Architektur**: Nicht-funktionale Anforderungen (Modifizierbarkeit, Wartbarkeit, Performance, Sicherheit, Zuverlässigkeit, Robustheit, …) und technische Rahmenbedingungen.
  - **Fachliche Architektur**: Funktionale Anforderungen

Die Quasar-Methode

Software-Kategorien (Blutgruppen)

Software kann sein ...

0 bestimmt von gar nichts (Behälter, Strings)
ideal wieder verwendbar, für sich alleine nutzlos

A bestimmt von der Anwendung (Kunde, Auftrag, Bestellung)
das eigentliche Projektziel

T bestimmt von mindestens einem technischen API (z.B. Datenverwaltung)
muss sein

AT bestimmt von der Anwendung und mindestens einem technischen API
vermeiden; im Notfall sorgfältig abgrenzen

R Repräsentations-Software (Transformation zwischen A und T; milde Art von AT)

Kombinationen:

A + 0 = A
T + 0 = T
A + T = AT

Die drei Architektursichten des Systementwurfs

1. Anwendungsarchitektur (A)

- Welche fachlichen Komponenten gibt es?
- Wie arbeiten diese zusammen?
- Worauf haben Benutzer und Drittsysteme Zugriff?

2. Technische (Software-) Architektur (T)

- Wie kann zwischen A-Architektur und TI-Architektur vermittelt werden?
- Welche technischen Komponenten gibt es? Welche Technologien und Open-Source-Bausteine werden dafür verwendet?

3. Technische Infrastruktur (TI)

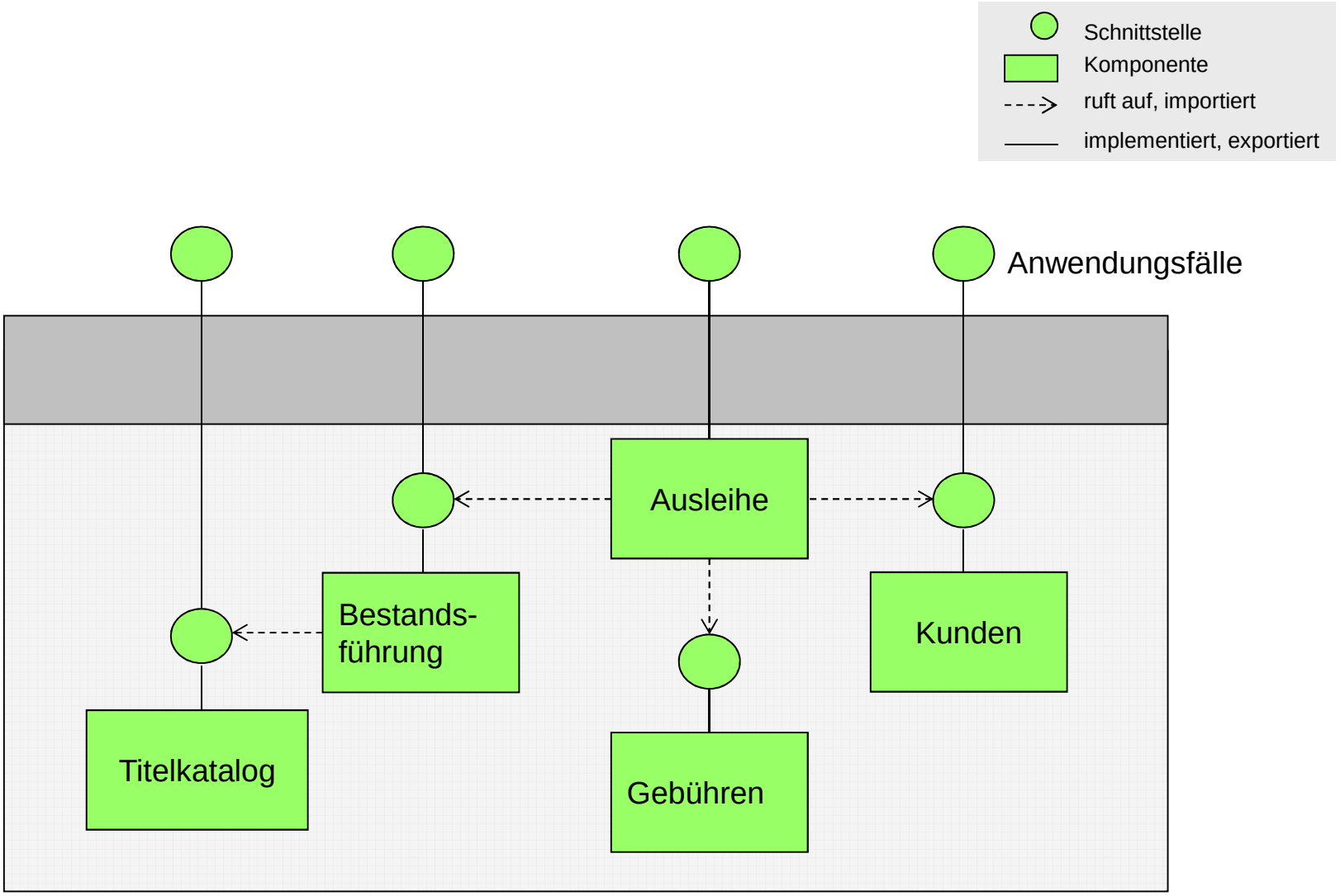
- Welche Verbindungssoftware, Datenbanken und andere Systemsoftware wird verwendet?
- Welche Maschinen und Verbindungen gibt es ?
- Wie wird das System drauf ausgeliefert?
- Welche Protokolle sind vorzusehen?

Anwendungsarchitektur (A)
„Die fachliche Lösung“

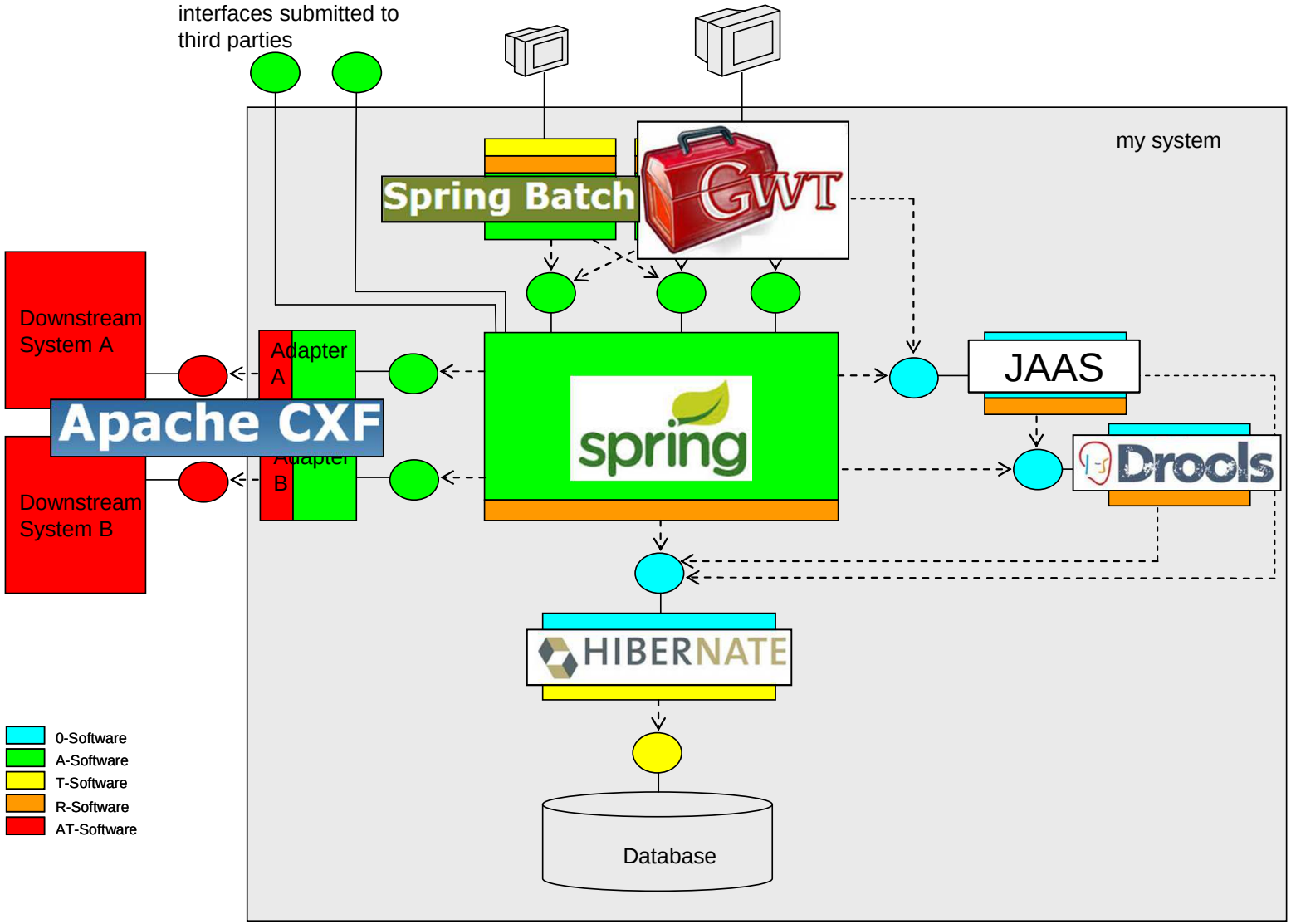
Technik-Architektur (T)
„Die Glue-Architektur“

Technische Infrastruktur (TI)
„Die technologische Basis“

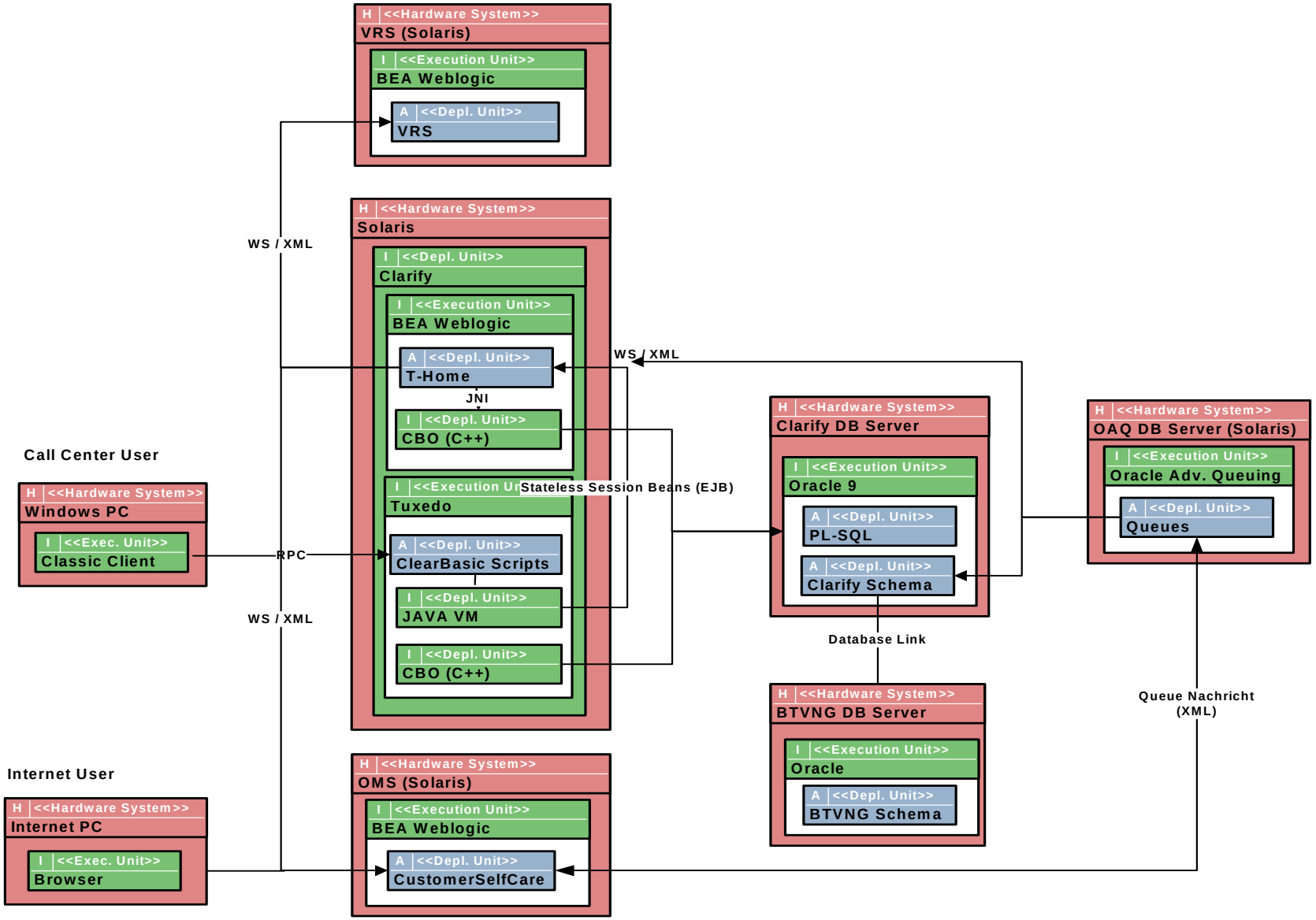
A-Architektur: Beispiel Büchereisystem



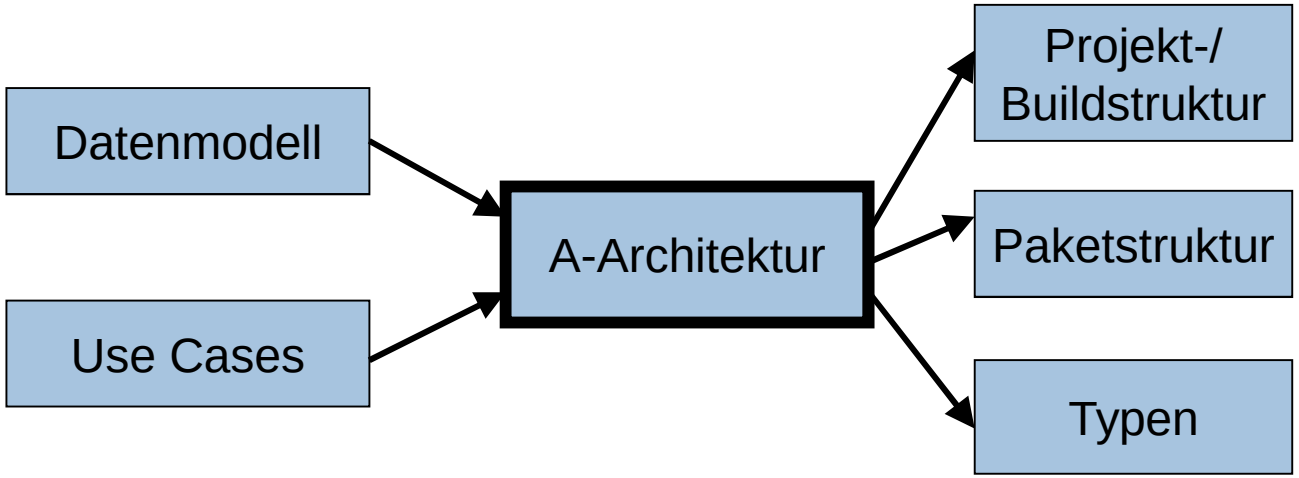
T-Architektur: Musterarchitektur für Informationssysteme



TI-Architektur: Beispiel CRM-System



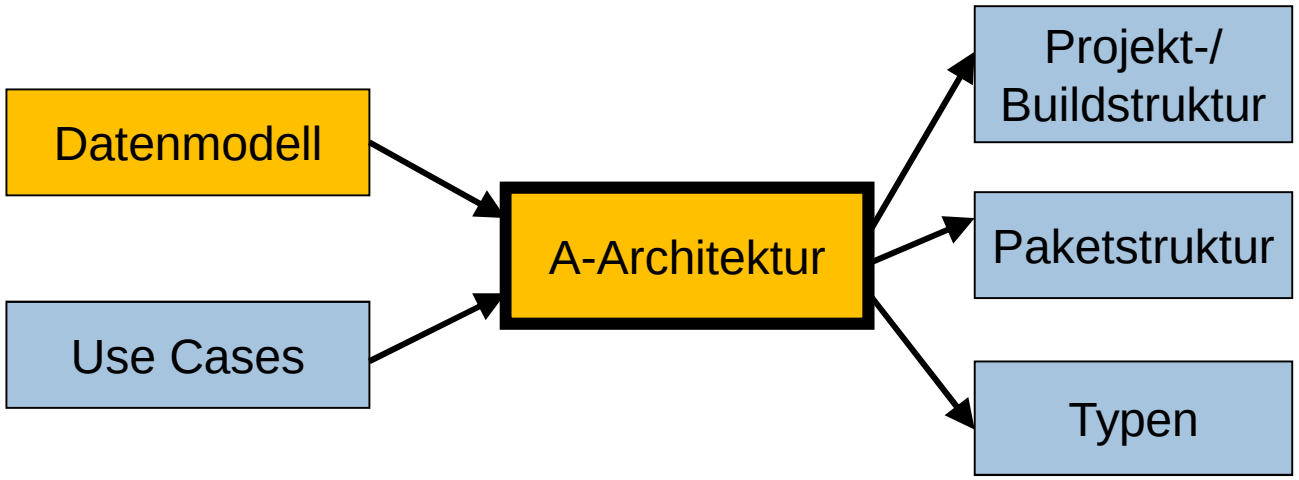
T- und TI-Architektur folgen oft einer Blaupause*. Eine Herausforderung ist der Entwurf der A-Architektur.



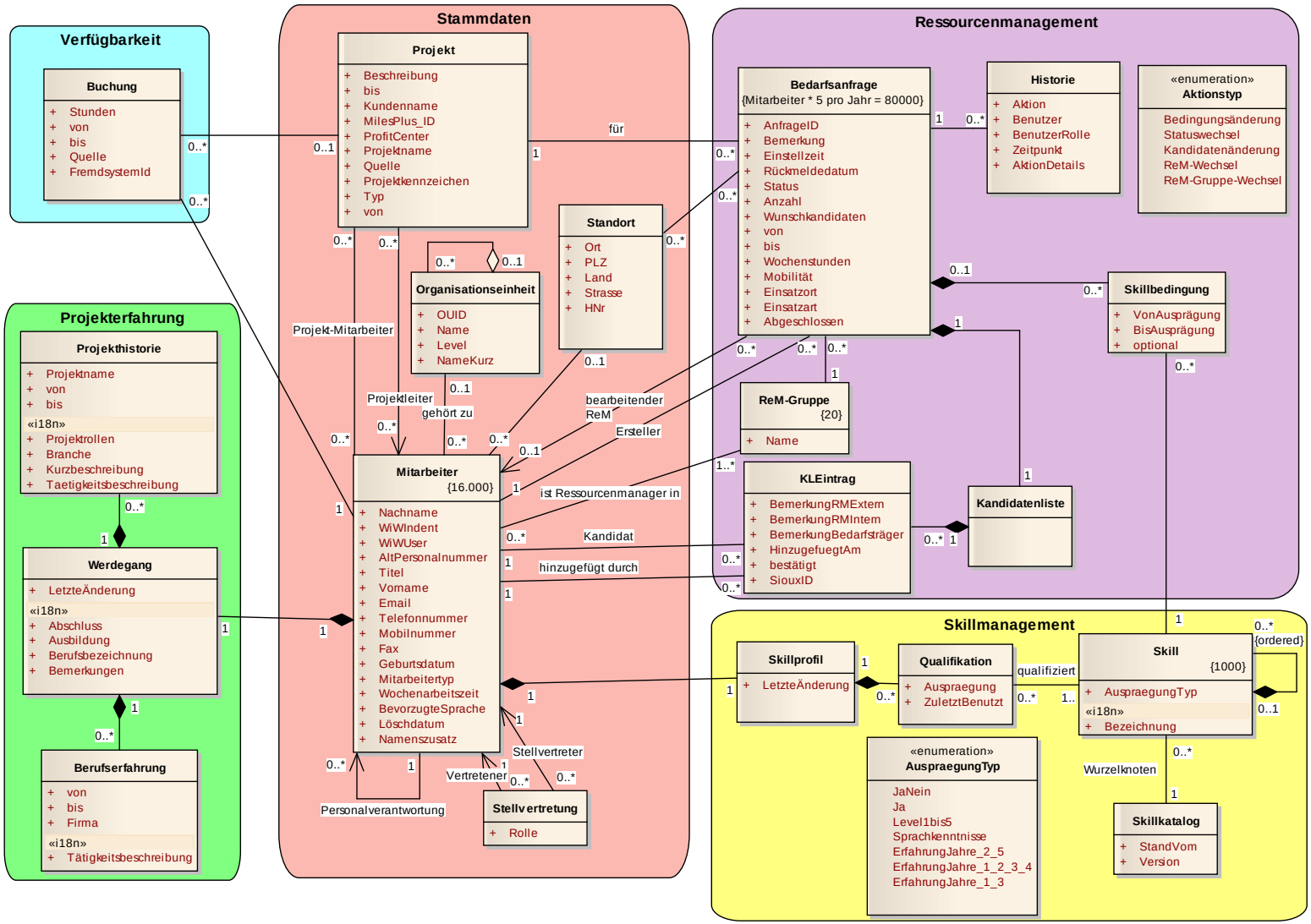
- **Datenmodell**
  - Einfluss auf Komponentenbildung: Informationshoheit
- **Use Cases**
  - Einfluss auf Komponentenbildung: Zuordnung eines Features zu genau einer Schnittstelle

* siehe Vorlesungsteil Software OEM

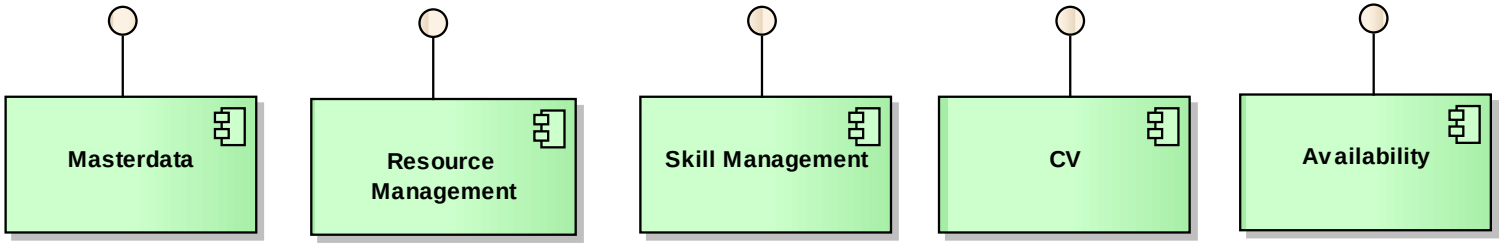
Die Quasar-Methode am Beispiel

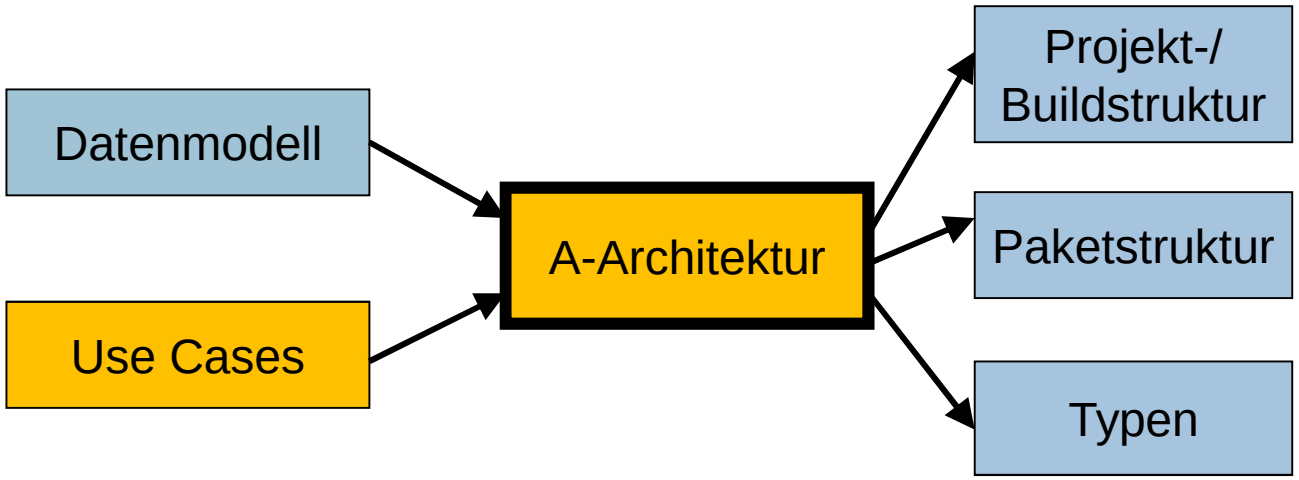

Datenmodell
Use Cases
A-Architektur
Projekt-/
Buildstruktur
Paketstruktur
Typen

Die A-Komponentenbildung erfolgt auf Basis des Datenmodells. Jede A-Komponente hat ihre Datenhoheit.

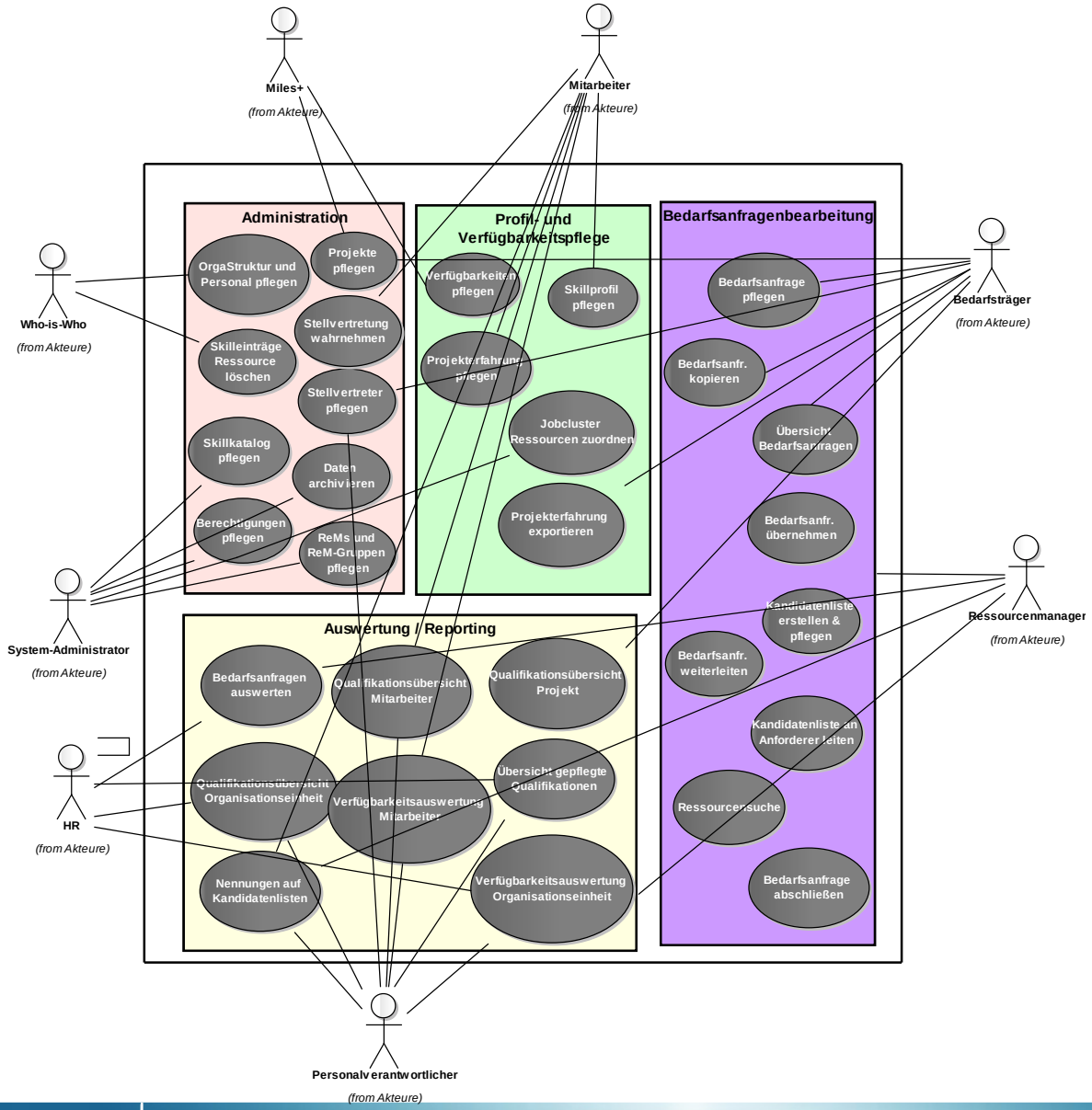


Die SI-OUX A-Architektur repräsentiert diesen Komponentenschnitt.

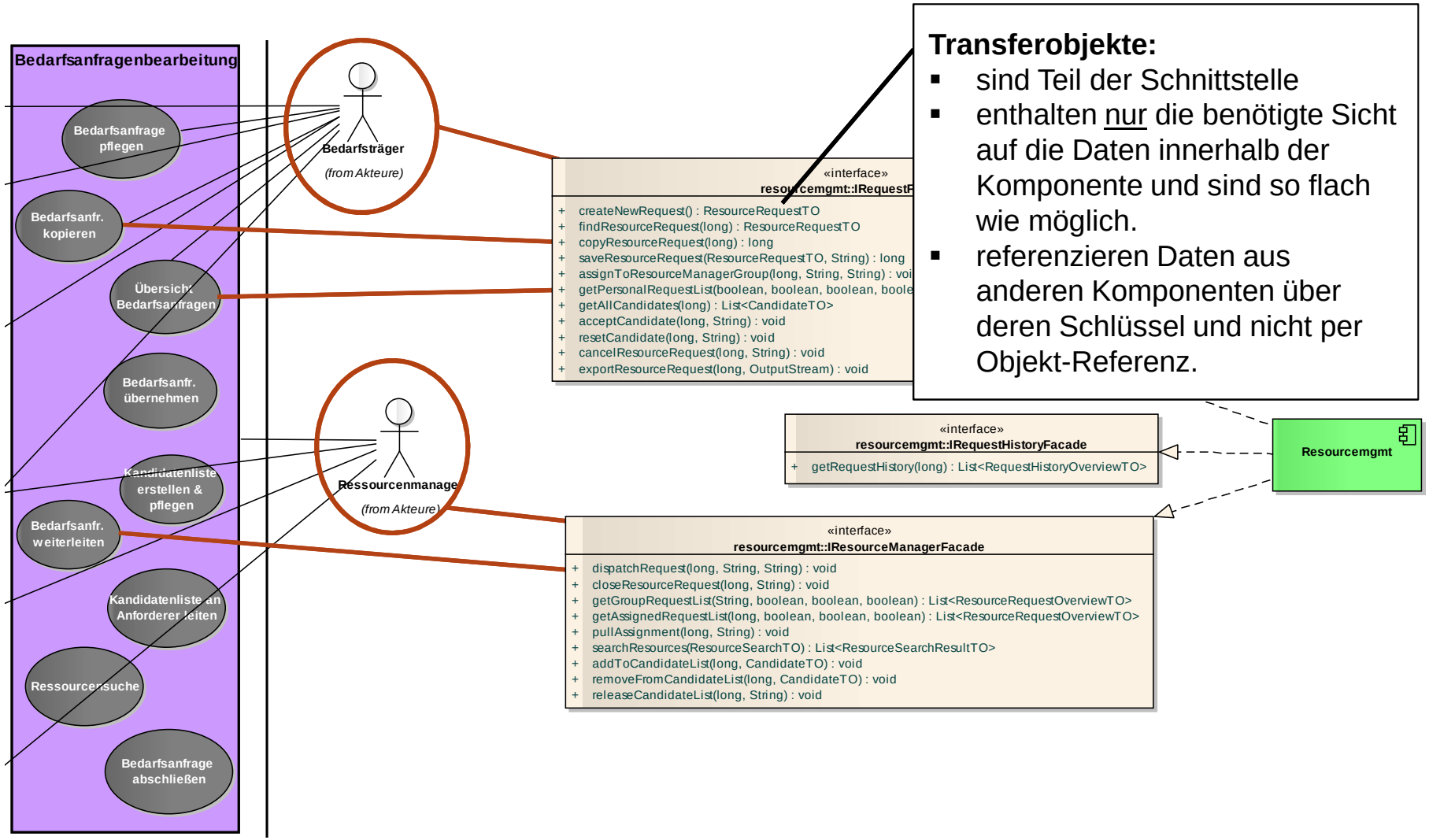



Datenmodell
Use Cases
A-Architektur
Projekt-/
Buildstruktur
Paketstruktur
Typen

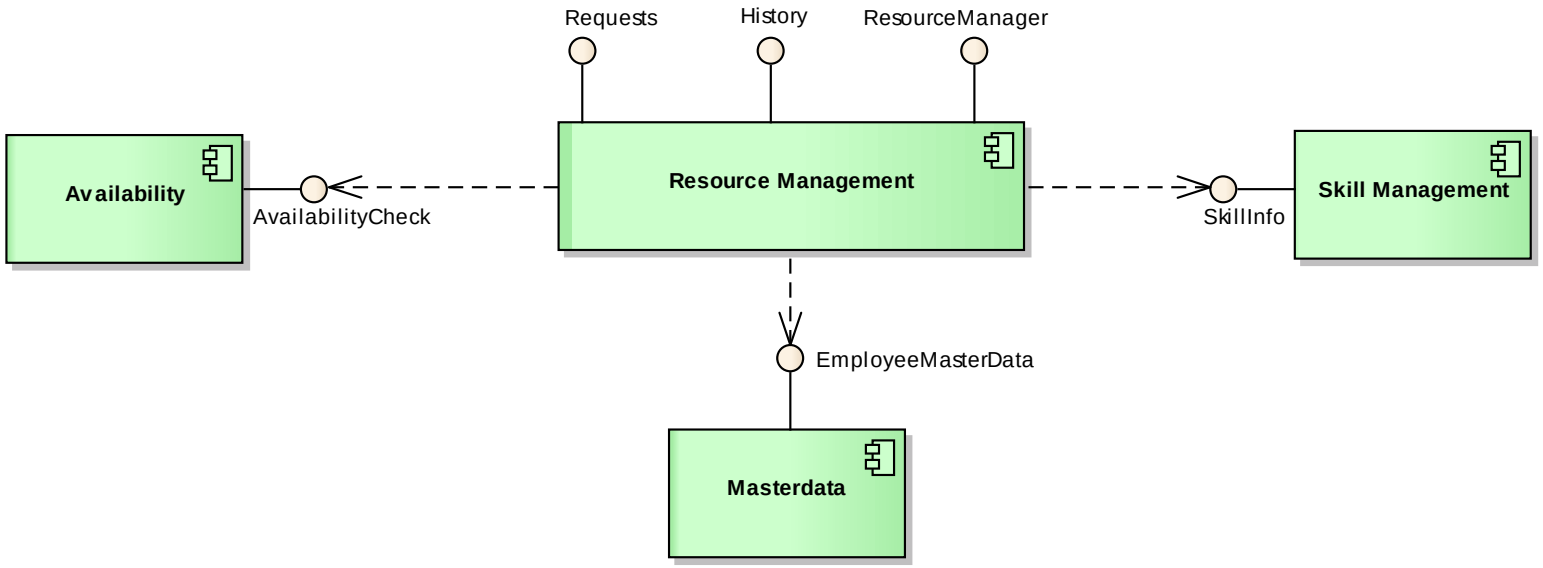
Jeder Use Case wird einer Komponente zugeordnet.

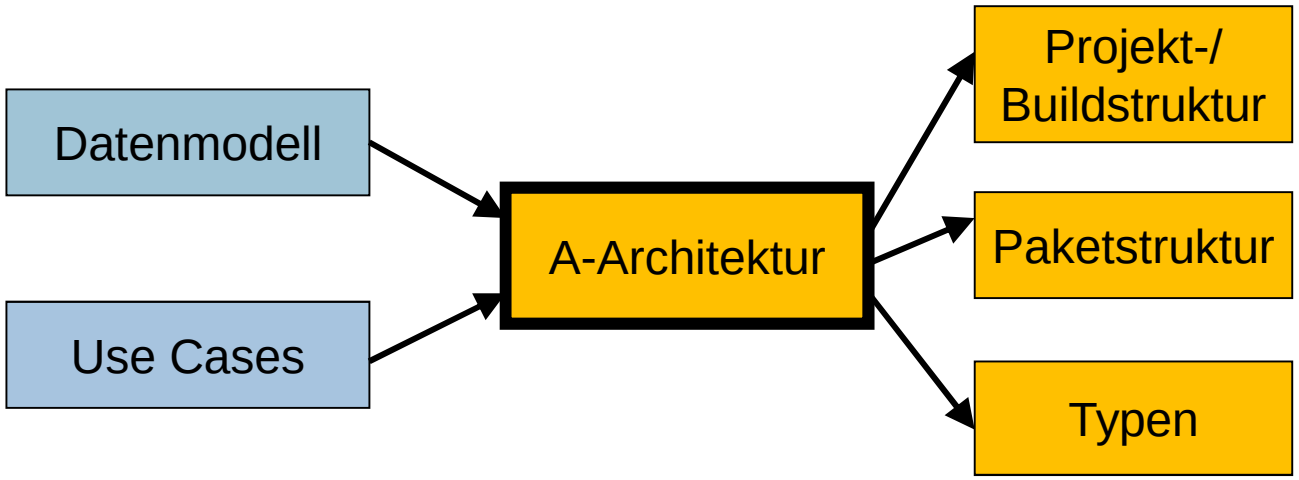


Aus dieser Zuordnung heraus werden dann rollenorientiert die Komponenten-Schnittstellen entworfen.

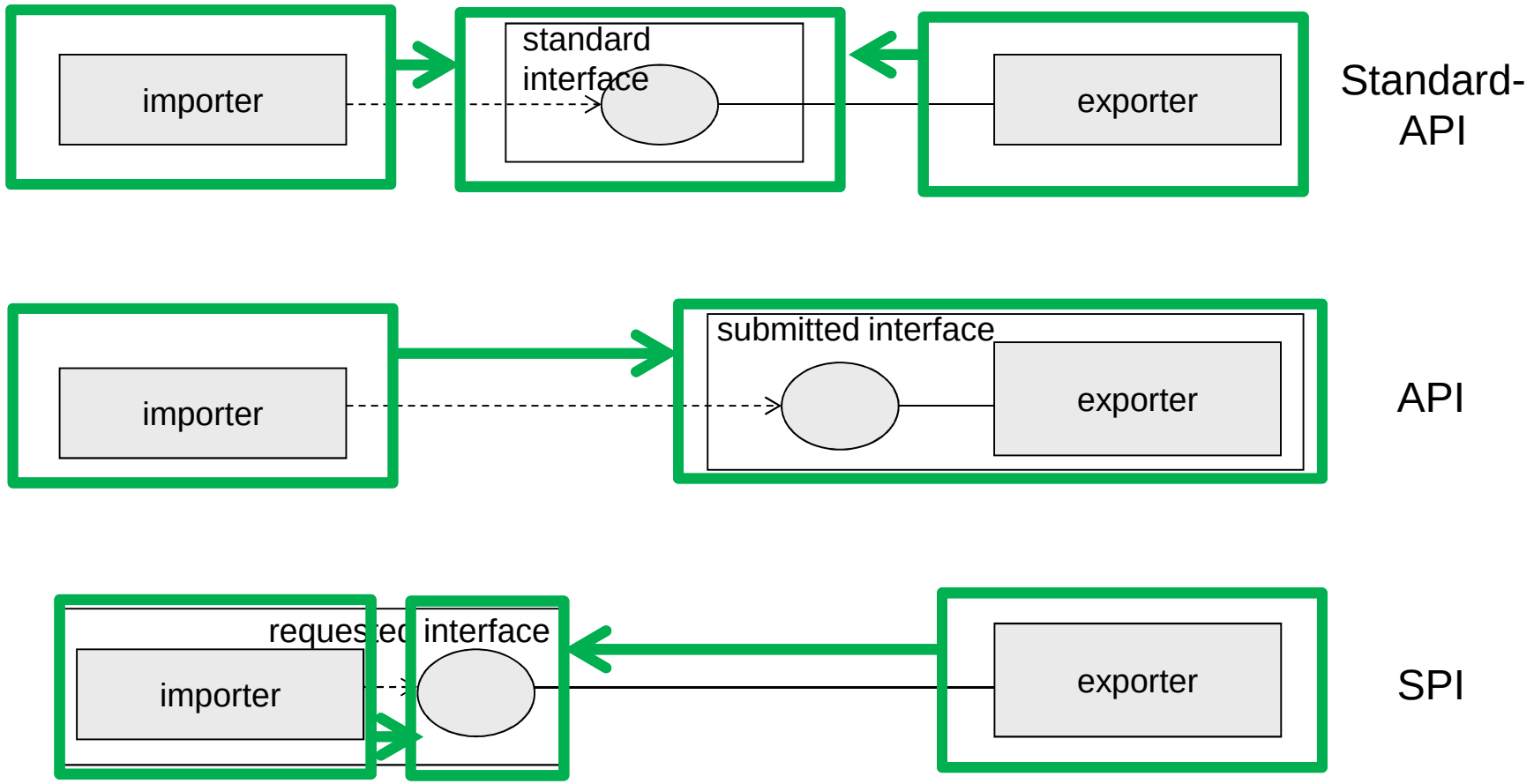


Die Ablaufbeschreibungen der Use Cases und die Abhängigkeiten im Datenmodell bestimmen die Verschaltung der Komponenten.




Datenmodell
Use Cases
A-Architektur
Projekt-/
Buildstruktur
Paketstruktur
Typen

Die Projektstruktur schneidet man auf Basis der definierten Architekturkomponenten.



Pro Komponente und eigenständiger Schnittstelle wird ein Sub-Projekt in der Codestruktur angelegt.

A-Architektur

Fachliche Business-Kompnenten

- sioux-business-availability (Verfügbarkeiten)
- sioux-business-cv (Projekterfahrung)
- sioux-business-masterdata (Stammdaten)
- sioux-business-reporting (Reporting)
- sioux-business-resourcemgmt (Resourcemanagement)
- sioux-business-rightsmgmt (Berechtigungen)
- sioux-business-skillmgmt (Skillmanagement)

T-Architektur

Technische Komponenten

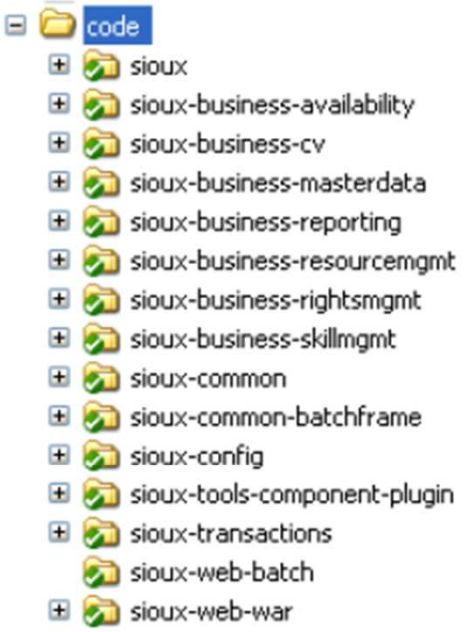
- sioux-common-batchframe (JEE Batchframework)
- sioux-web-war (JEE Webapp)
- sioux-web-batch (JEE Batchsteuerung)
- sioux-web-common (JEE Infrastruktur)

Infrastruktur Komponenten

- sioux-config (Systemkonfiguration)
- sioux-common (Querschnitt)
- sioux-transactions (AOP Transaktionsdefinitionen, JPA Mapping (persistence.xml))

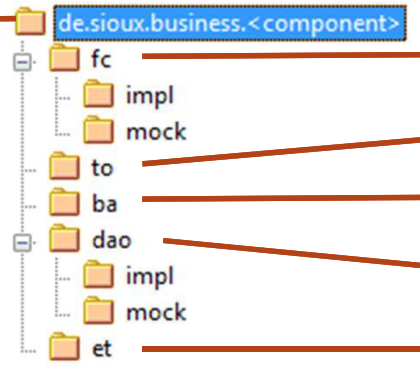
Wurzelprojekt

- sioux (IntelliJ Projektdateien)



Ein solches Sub-Projekt ist nach einer standardisierten Paketstruktur mit festgelegten Abhängigkeiten organisiert.

Top Level Package
mit Interfaces



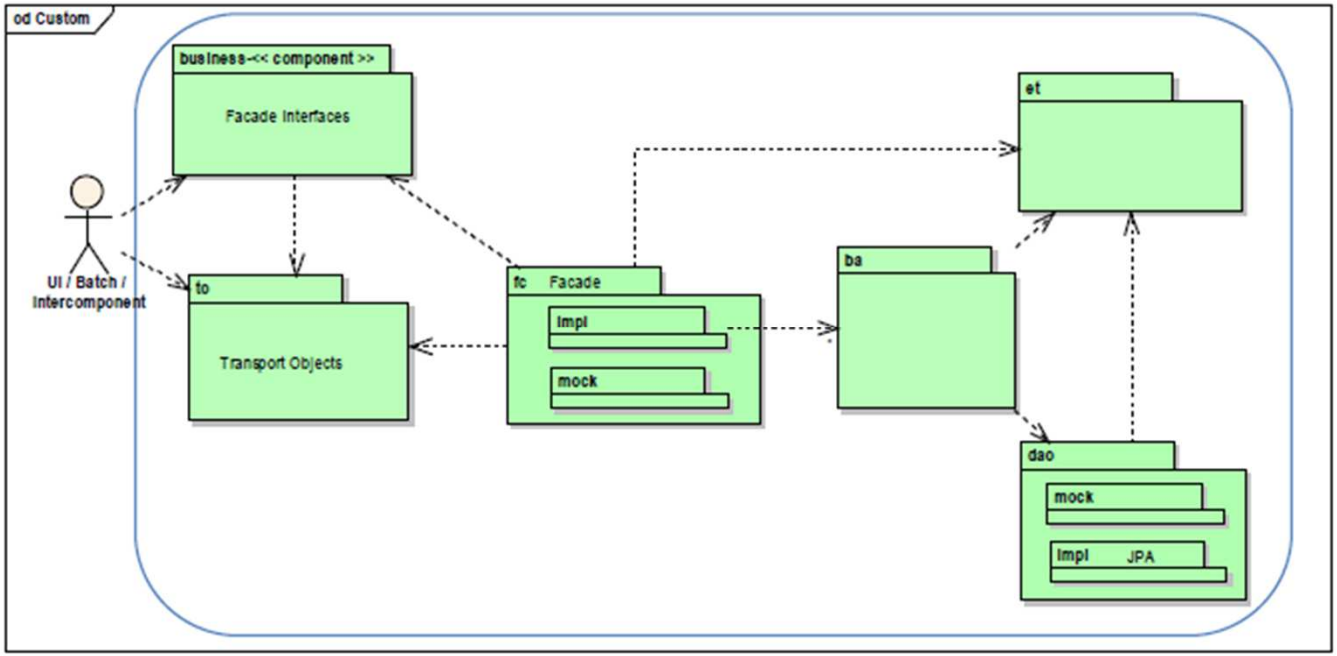
Business Facades (Dienste)

Transfer Objects

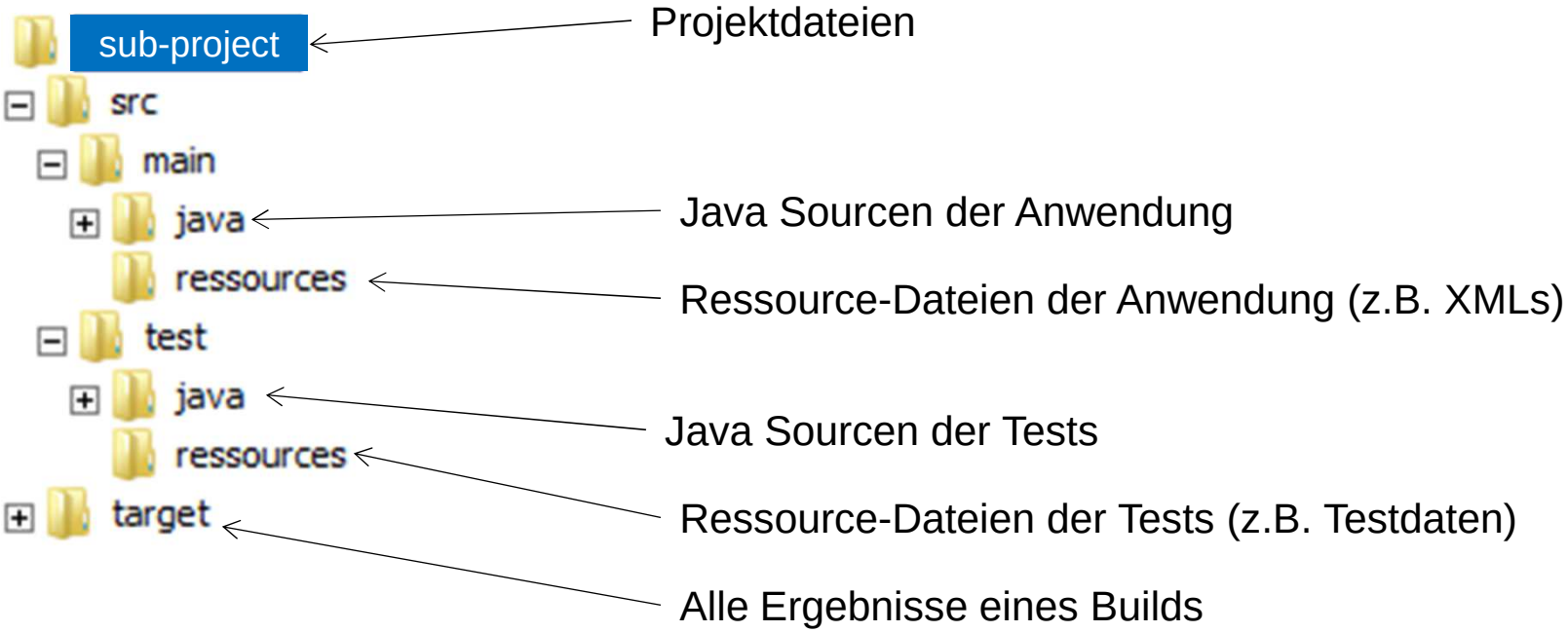
Business Activities

Data Access Objects

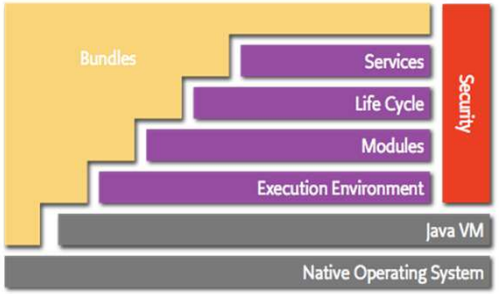
Entity Types (Datenhoheit)



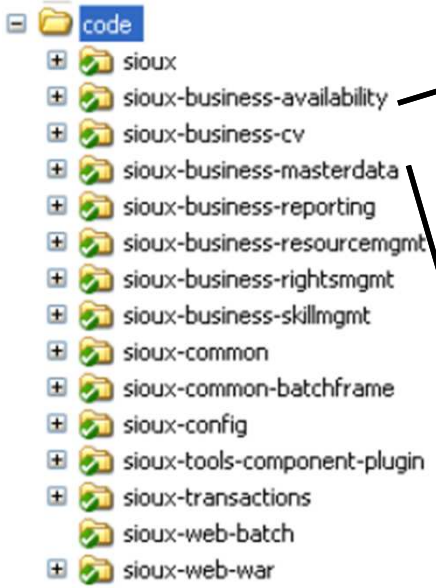
Die Verzeichnisstruktur innerhalb der Sub-Projekte sollte standardisiert und immer gleich sein.



Mit Hilfe eines OSGi-Containers kann eine Komponente auch zur Laufzeit leben.



OSGi ist eine Laufzeitumgebung oberhalb der Java Virtual Machine, die Laufzeit-Komponenten und Laufzeit-Dienste und ihre Abhängigkeiten verwaltet.



OSGI-INF/blueprint/availability.xml

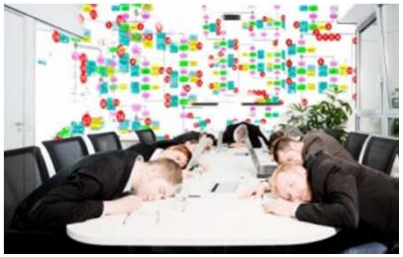
```
<blueprint>
      <reference id=„business.masterdata.requestfacade“
                 interface=„de.sioux.business.masterdata.IRequestFacade“ />
      <bean id=„requestAvailability“
            class=„de.sioux.business.availability.ba.RequestAvailability“>
           <property name=„requestFacade“ ref=„masterdata.requestfacade“/>
      </bean>
</blueprint>
```

OSGI-INF/blueprint/masterdata.xml

```
<blueprint>
      <service id=„business.masterdata.requestfacade"
               interface=„de.sioux.business.masterdata.IRequestFacade">
           <bean class="de.sioux.business.masterdata.fc.impl.RequestFacadeImpl" />
      </service>
</blueprint>
```

Ein passender erster (Ent-)Wurf ist unwahrscheinlich: Es sollte kontinuierlich validiert werden.

Methoden:



Schreibtischtests:
Die Architektur in Gedanken laufen lassen.

Proof of Concepts:
Die Architektur implementieren und erproben.

Variabilitätsanalyse:
Änderungsszenarien durchspielen und Auswirkungen analysieren.

Architekturkonsistenz prüfen:
Abweichungen zwischen Architektur und Code ermitteln.

Proof of Concept: Schnittstelle programmieren und Testfälle gegen Attrappen entwickeln.

Die Schnittstelle:

```
public interface ICalculator {
    int add(int a, int b);
}
```

Der Testfall:

testAddition(CalculatorTest.java:19)

CalculatorTest (edu.hm.se.calculator)
testAddition

```
13      @Test
14      public void testAddition(){
15          ICalculator calculator = CalculatorMock.getInstance();
16          assertThat(calculator.add(0, 0), equalTo(0));
17          assertThat(calculator.add(1, 1), equalTo(2));
18          assertThat(calculator.add(2, 1), equalTo(3));
19      }
```

https://code.google.com/p/hamcrest

Die Attrappe:

```
public class CalculatorMock {

    public static ICalculator getInstance(){
        ICalculator mock = mock(ICalculator.class);
        when(mock.add(1,1)).thenReturn(2); //else return 0 (default)
        return mock;
    }

}
```

https://code.google.com/p/mockito