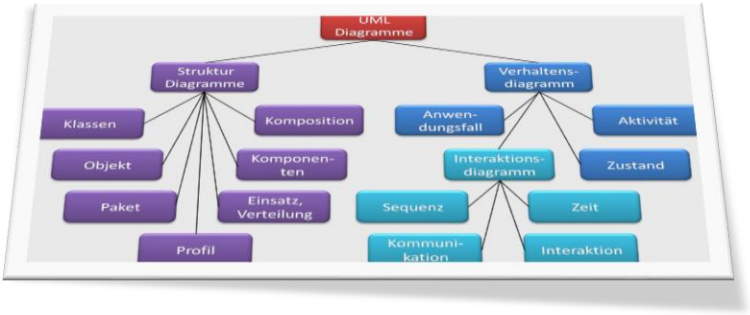
ITEN ENGINEERING
Software Development

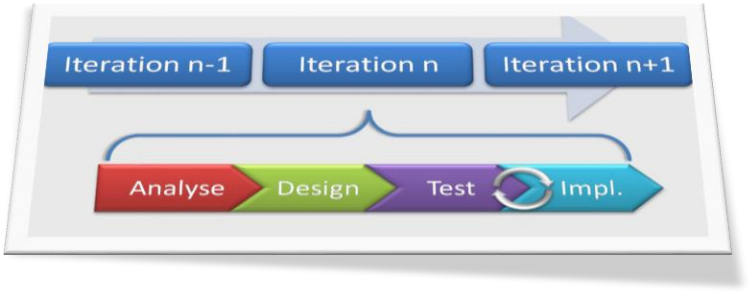
Lösungen

Grundlagen

UML



Analyse & Design



Ausgabe 08.2017

Inhaltsverzeichnis

1. Lösungen Grundlagen

1.1. SOLID (Prinzipien)

Die Abkürzung SOLID steht für die folgenden fünf Design Prinzipien:

- **S**ingle Responsibility Principle (SRP)
  "There should never be more than one reason for a class to change."

  Jede Klasse soll genau für einen fachlichen Aspekt verantwortlich sein. Dass Bedeutet dass Klassen keine Eigenschaften und Operationen enthalten sollen, die nicht zu Ihrem Verantwortungsbereich gehören.

- **O**pen Closed Principle (OCP)
  "Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification."

  Bestehende Entitäten sollen nicht verändert werden können sondern Erweiterbar sein. Damit soll sichergestellt werden, dass das ursprüngliche Verhalten nicht geändert wird, sondern wo nötig erweitert wird. Hierzu können die objektorientierten Konzepte wie zum Beispiel Vererbung, Komposition, Delegation, Schnittstellen, etc. eingesetzt werden.

- **L**iskov Substitution Principle (LSP)
  "Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it."

  Mit diesem Prinzip wird das Verhalten von abgeleiteten Klassen / Typen so eingeschränkt, dass sie das ursprüngliche Verhalten der Basis Klasse nicht verletzen. Damit können Funktionen die diese Klassen verwenden problemlos wiederverwendet werden ohne dass Sie über die Details der einzelnen Implementationen Bescheid wissen müssen.

- **I**nterface Segregation Principle (ISP)
  "Clients should not be forced to depend upon interfaces that they do not use."

  Schnittstellen sollten so gehalten werden dass die Benutzer (Clients) nicht von Funktionen abhängig sind, die Sie nicht benötigen. Dadurch wird die Kopplung zwischen den Komponenten auf ein Minimum beschränkt und unnötige Abhängigkeiten verhindert.

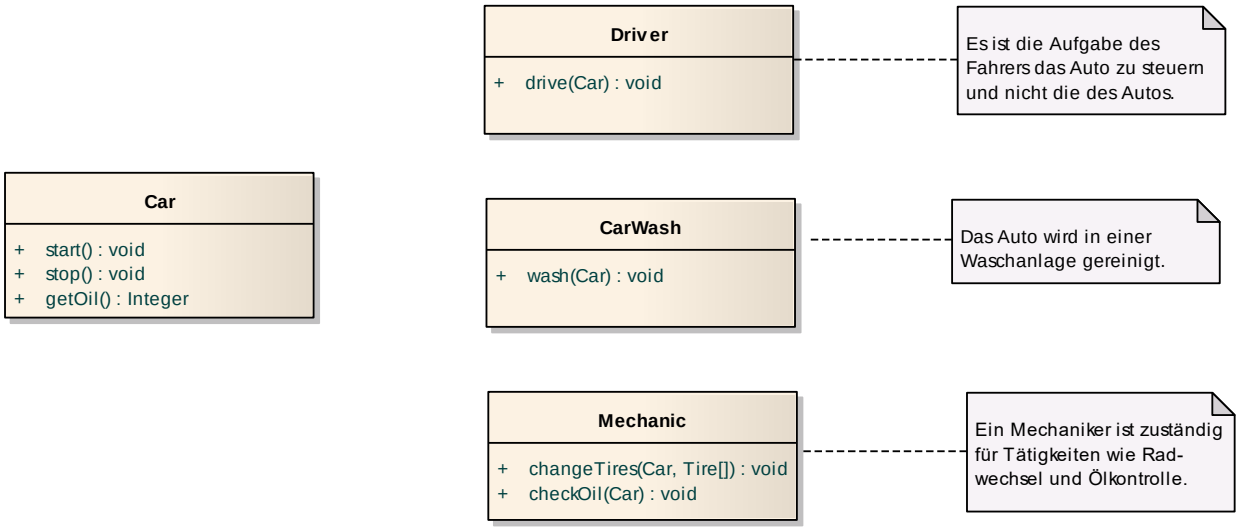
- **D**ependency Inversion Principle (DIP)
  "(A) High level modules should not depend upon low level modules. Both should depend upon abstractions. (B) Abstractions should not depend upon details. Details should depend upon abstractions."

  Dadurch, dass die höheren Module Schnittstellen definieren, welche von den tiefer liegenden Modulen Umgesetzt werden, werden die Abhängigkeiten invertiert. Die höheren Module mit den allgemeinen Abläufen sind nicht mehr von den Details der Umsetzungen abhängig.

Die Einhaltung der SOLID Prinzipien stellt sicher, dass die Software robuster, wartbarer und wieder verwendbarer wird. Für weitere Details siehe “The Prinziples of OOD“ [16] von Robert C. Martin.

1.2. Car (Verantwortlichkeiten)

Lösungsvorschlag:

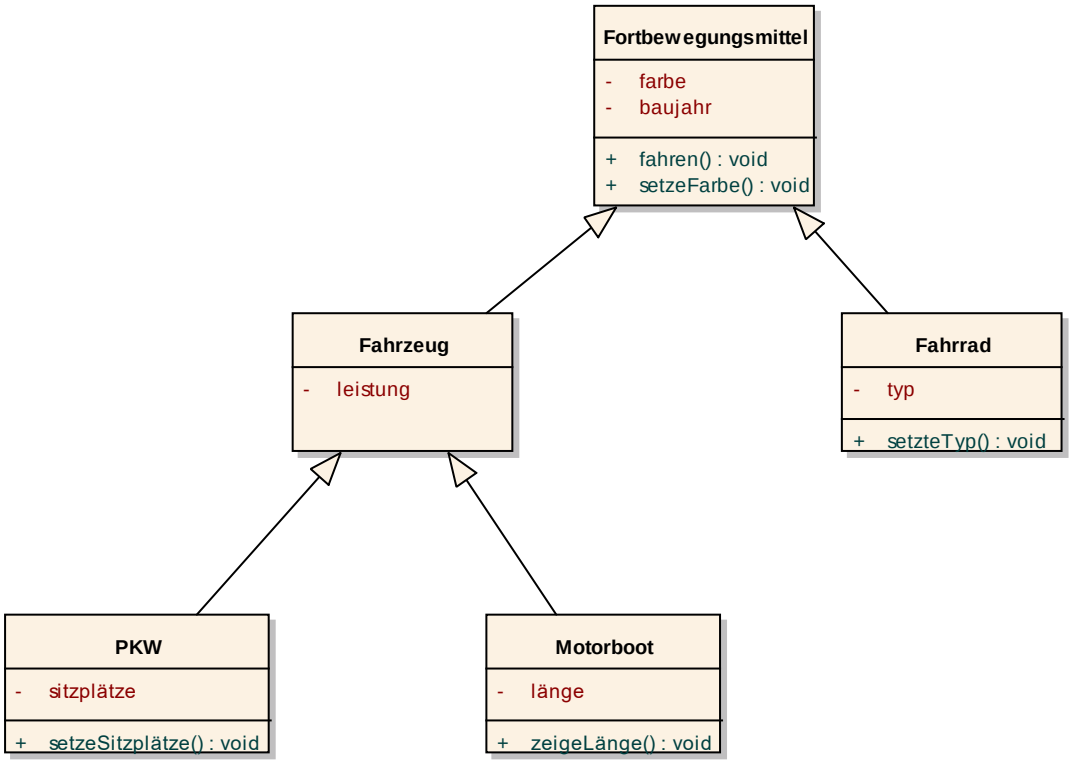


Jede Klasse ist verantwortlich für eine zusammengehörende Gruppe von Aufgaben. Zum Auto gehören noch die Operationen start(), stop() und getOil().

Die anderen Methoden werden in eigene Klassen ausgelagert. Diese erhalten einen entsprechenden Parameter für die Angabe des Autos, an welchem die betreffenden Aufgaben ausgeführt werden sollen (z.B. die wash() Methode der Waschanlage).

1.3. Fortbewegungsmittel (Vererbung)

Lösungsvorschlag:

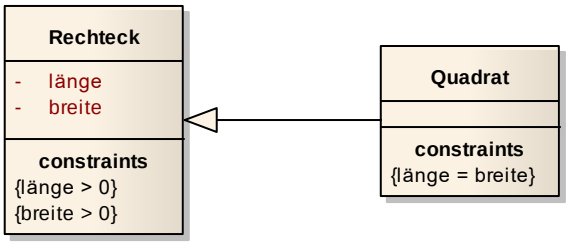


1.4. Quadrat (Vererbung)

Lösungsvorschlag:

Variante 1 - Quadrat als Unterklasse von Rechteck:

- Das Quadrat ist eine spezielle Form des Rechtecks und wird daher bei dieser Variante als Unterklasse des Rechtecks definiert:

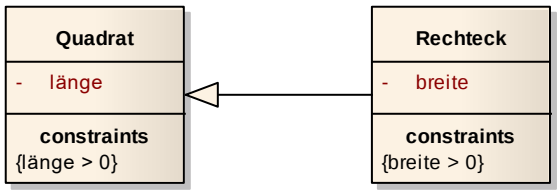


- Vorteile
  - Die Attribute vom Rechteck können wiederverwendet werden.
  - Man benötigt einzig eine zusätzliche Zusicherung, die sicherstellt, dass Länge und Breite gleich gross sind.

- Nachteile:
  - Für die Angabe der Abmessungen benötigt das Quadrat nur die Seitenlänge.
  - Die breite ist redundant. Dies wird aber in Kauf genommen, da nach unserer normalen Vorstellung ein Quadrat eine Spezialisierung eines Rechtecks ist.
  - In der Klasse Quadrat wird eine Zusicherung auf Eigenschaften der Oberklasse gemacht.

Variante 2 - Rechteck als Unterklasse von Quadrat:

- Um die Nachteile der ersten Variante zu beheben, kehren wir die Abhängigkeit um.
- Neu soll das Rechteck als Spezialisierung eines Quadrats modelliert werden:



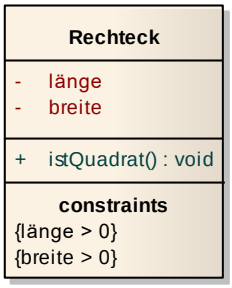
–

- Vorteile:
  - Das redundante Attribut der ersten Variante gibt es nicht mehr.
  - Die Zusicherungen sind auf die Eigenschaften der jeweiligen Objekte definiert.

- Nachteile:
  - Es kann kein sinnvolles Unterscheidungsmerkmal (Diskriminator) für die Erstellung der Hierarchie angegeben werden.
  - Gemäss Definition kann eine Spezialisierung immer anstelle der Oberklasse eingesetzt werden. Wenn wir nun zum Beispiel ein Objekt q1 von Typ Quadrat haben und ein Objekt r1 vom Typ Rechteck, so wäre q1 = r1 eine gültige Zuweisung. Diese Möglichkeit ist aber sicherlich nicht beabsichtigt.

Variante 3 – Quadrat als Besonderheit des Rechtecks

- Als mögliche Lösung für unser Problem kann zum Beispiel folgender Ansatz gewählt werden: Das Quadrat wird gar nicht als eigene Klasse modelliert.



- Stattdessen wird das Quadrat als eine Besonderheit eines Rechtecks betrachtet. Dazu müssen einzig Länge und Breite übereinstimmen.

- Damit dieser Fall unterschieden werden kann, wird die Klasse Rechteck mit der Methode istQuadrat() ergänzt.

Fazit:

- Das Beispiel zeigt, dass die Vererbung zwar einfach einzusetzen ist, aber Ihre Tücken hat.

- Generell wird davon abgeraten, in Unterklassen, Zusicherungen auf Attribute und Operationen von Oberklassen zu definieren, da die Unterklasse nicht sicherstellen kann, dass sich alle Methoden der Oberklasse daran halten.

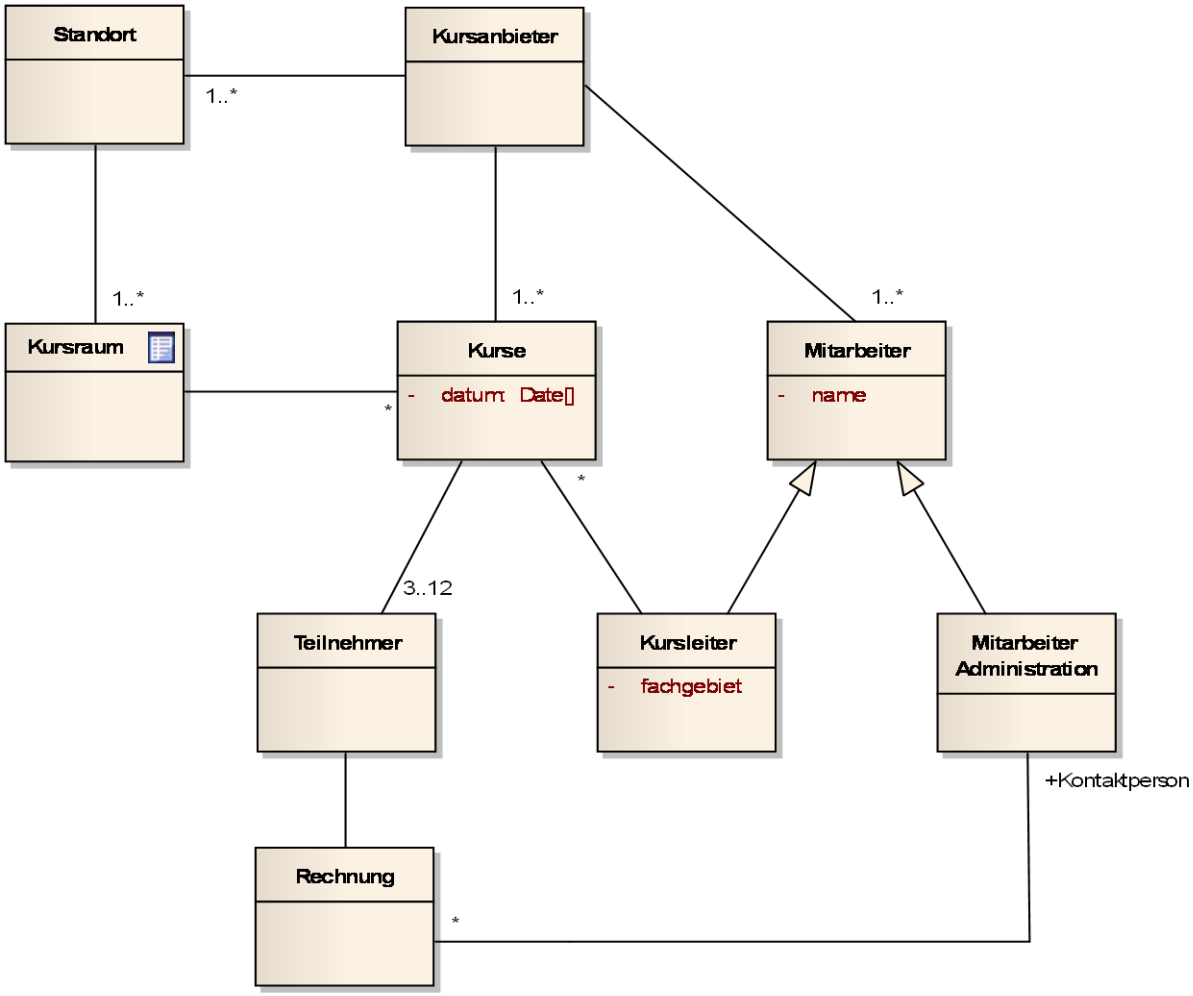
- Anstelle der Vererbung existieren alternative Lösungen (Aggregation, Delegation, Schnittstellen sowie generische und generative Ansätze), mit denen die verschiedenen Probleme der Vererbung umgangen werden können.

- Es gilt daher die Regel / Warnung:

Vermeide Vererbung, wenn es alternative Lösungen dazu gibt.

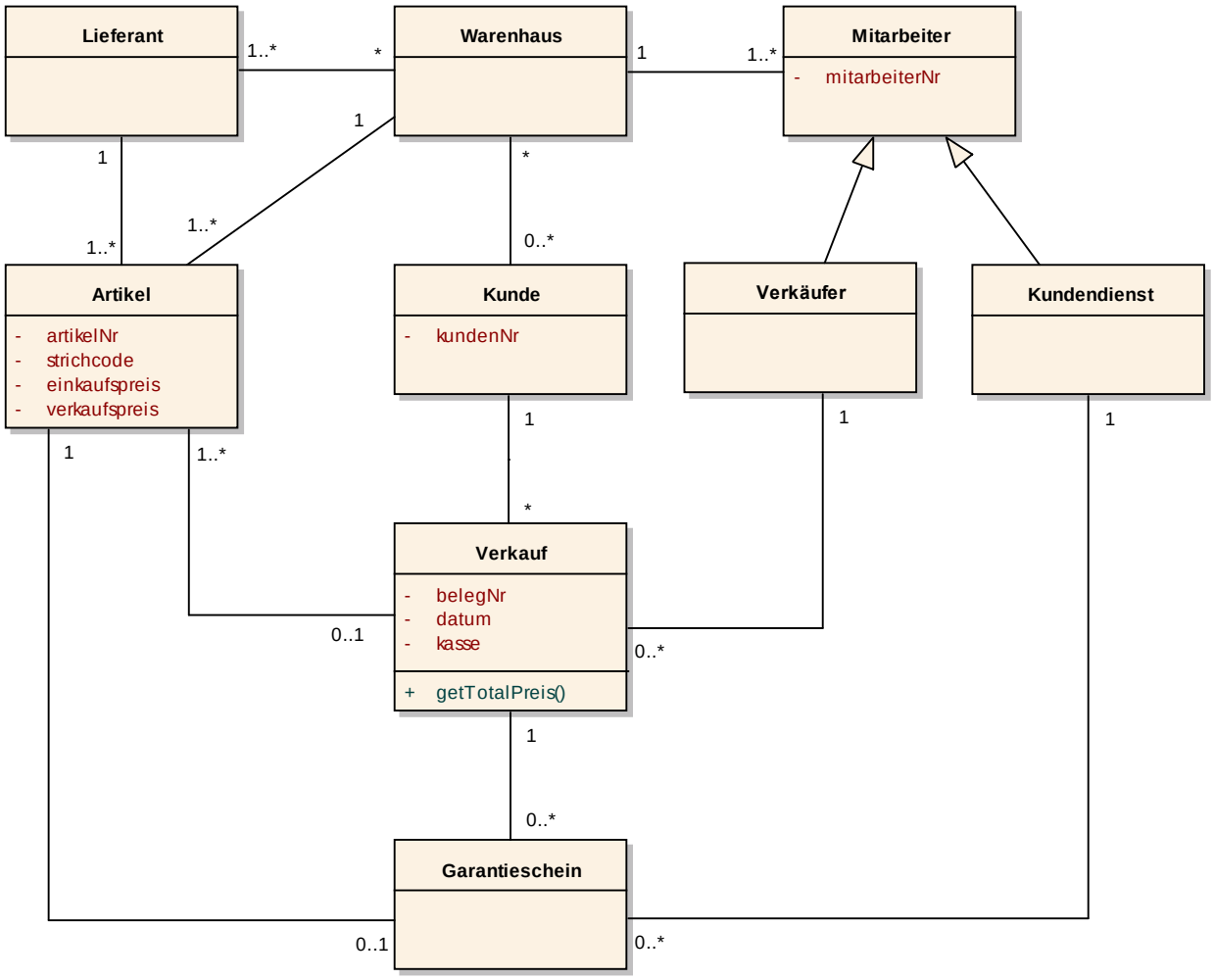
1.5. Kursanbieter (Klassendiagramm)

Lösungsvorschlag:



1.6. Warenhaus (Klassendiagramm)

Lösungsvorschlag:



1.7. Hermes 5 (Vorgehensmodell)

Antworten zum Bereich Verstehen:

1) Aus welchen Phasen besteht das Phasenmodell von Hermes?

   - Initialisierung, Konzept, Realisierung, Einführung

2) Wie lauten die Hierarchieebenen der Projektorganisation?

   - Steuerung, Führung, Ausführung

3) Ordnen Sie die folgenden Rollen der jeweiligen Hierarchieebene zu:
   Tester, Projektleiter, Betriebsverantwortlicher, Qualitäts- und Risikomanager, Business Analyst, Auftraggeber.

   - Rollenübersicht:

Hierarchie-Ebene	Rolle
Steuerung	Auftraggeber
	Projektausschussmitglied
	Qualitäts- und Risikomanager
Führung	Projektleiter
	Teilprojektleiter
	Projektunterstützung
	Fachausschussmitglied
Ausführung	*Fachspezialisten*
	Anwendervertreter
	Anwendungsverantwortlicher
	Betriebsverantwortlicher
	Business Analyst
	Entwickler
	Geschäftsprozessverantwortlicher
	IT-Architekt
	ISDS-Verantwortlicher
	Testverantwortlicher
	Tester

4) Unter welchen Umständen kann eine Person mehrere Rollen wahrnehmen?

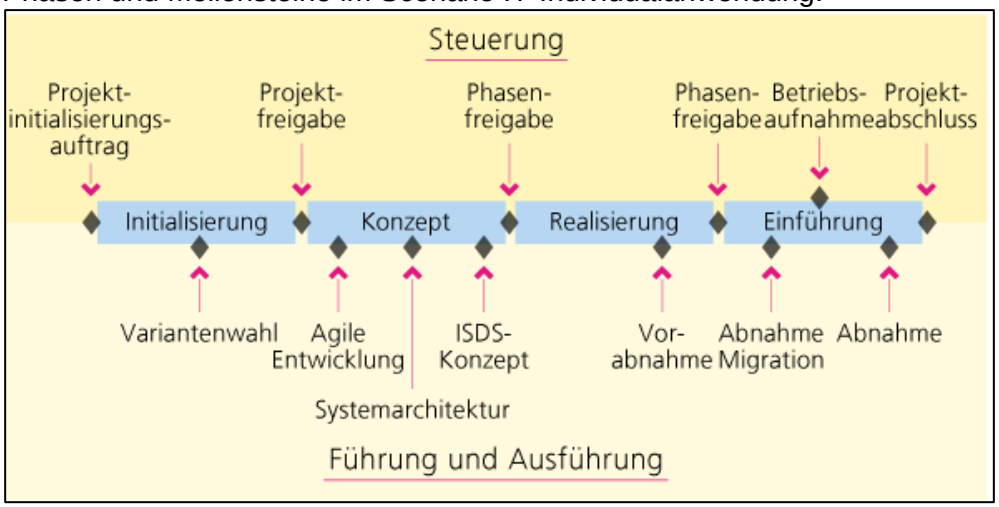
   - Bei der Rollenbesetzung müssen die folgenden Grundsätze beachtet werden, damit die Governance eingehalten werden kann:
     - Eine Person kann mehrere Rollen wahrnehmen, sofern keine Interessenkonflikte dadurch entstehen
     - Eine Rolle kann von mehreren Personen eingenommen werden (es gibt z.B. meistens mehrere Tester in einem Projekt)
     - Die Rollen Auftraggeber, Projektleiter und Fachspezialist müssen in jedem Projekt besetzt sein. Die weiteren Rollen werden abhängig von den Anforderungen des Projekts zugewiesen

5) Welches Standard Szenario von Hermes würden Sie verwenden für das Outsourcing von Dienstleistungen in ein Service-Center?

   - Organisationsanpassung

6) Definieren Sie zu welcher Phase die folgenden Meilensteine beim Scenario IT-Individualanwendung gehören: Variantenwahl, Abnahme, Systemarchitektur.

- Phasen und Meilensteine im Scenario IT-Individualanwendung:



7) Was ist das ISDS Konzept und wer trifft den Entscheid zur Freigabe?

- Informationssicherheit und Datenschutz Konzept.
- Zuständige Controlling- und Vorgabestelle.

8) In welcher Phase wird das Testkonzept erarbeitet und wann werden Tests durchgeführt?

- Testaktivitäten:

Initialisierung	Konzept	Realisierung	Einführung
	Testkonzept erarbeiten	Testinfrastruktur realisieren Test durchführen	Test durchführen Testkonzept und -infrastruk-tur überführen

9) Welches Ergebnis bildet die verbindliche Grundlage für die Projektfreigabe?

- Projektauftrag

10) Was sind die möglichen Entscheidungsoptionen der Phasenfreigabe?

- die Phase abgeschlossen wird oder ob vor dem Phasenabschluss weitere Ergebnisse zu erarbeiten sind
- die nächste Phase freigegeben wird
- das Projekt beendet wird

11) Zu welchem Modul gehört das Managen von Risiken

- Modul Projektführung

12) Welches Protokoll dokumentiert die Erfüllung der Vereinbarung über die Produkt-/ Systemeigenschaften und ist rechtlich verbindlich?

- Abnahmeprotokoll

1.8. Vergleich XP, Scrum, RUP und V-Modell (Vorgehensmodelle)

Nachfolgende Tabelle zeigt den Vergleich von XP, Scrum, RUP und dem V-Modell aus Sicht eines Software Architekten:

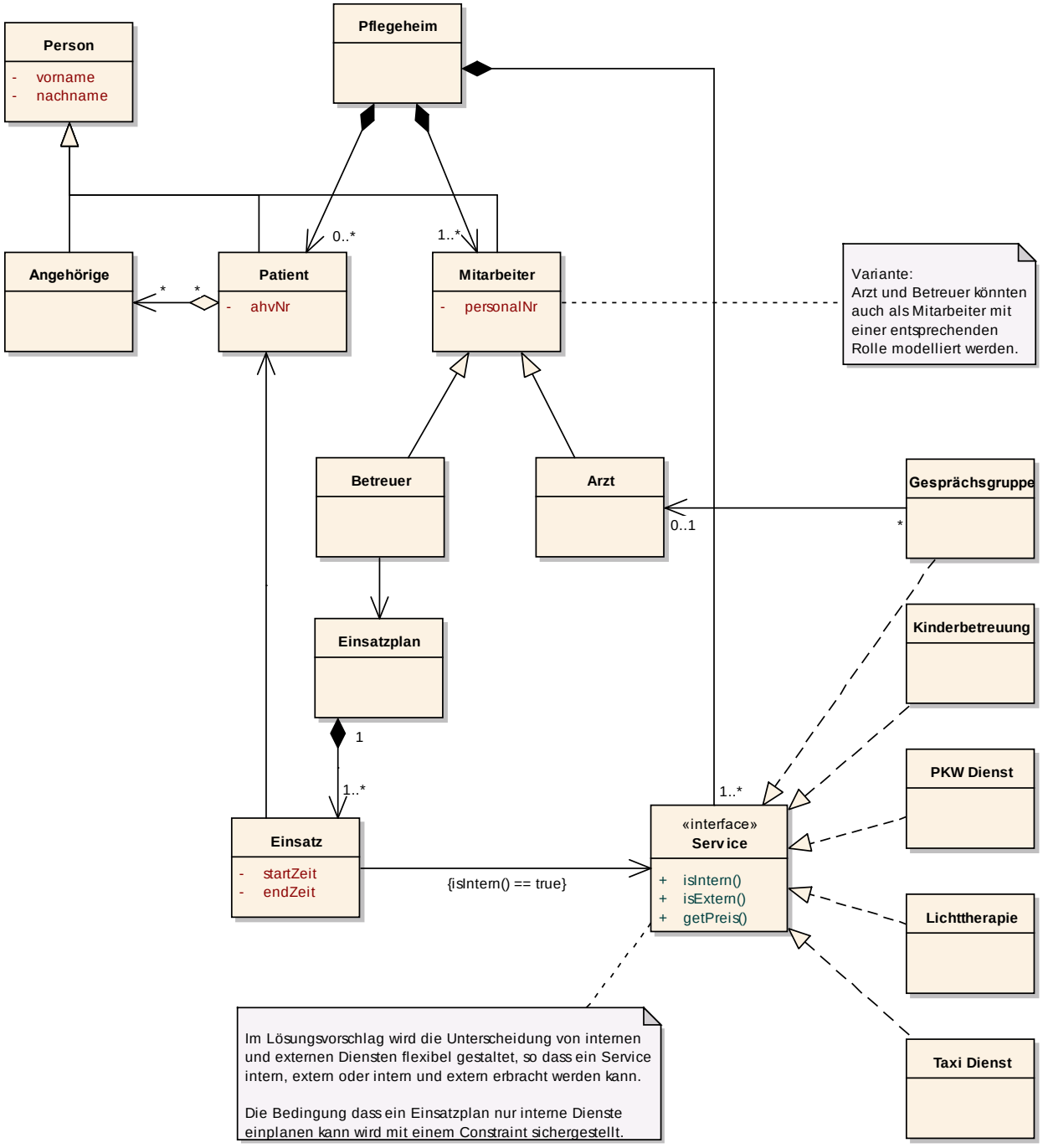
Merkmal	*XP*	*Scrum*	*RUP*	*V-Modell*
Fokus des Modells	Entwicklungs-prozess	Entwicklungs-prozess	Projektprozess	Unternehmens-prozess
Wichtige Rollen	Kunde, Softwareentwickler	Product Owner, Scrum Master, Team	Softwarearchitekt	Systemdesigner, Softwareentwickler, technischer Autor
Machbarkeit, Risiko	Alle Beteiligte	Product Owner mit Team	Architekt	Projekt-management
Priorisierung, Iterationsplanung	Stories mit Kunde vor jeder Iteration	Product Owner mit Team	Architekt	Systemdesigner, Projektleiter
Architektur-konzept	Entwickler, nur so viel wie nötig	Team mit Product Owner	Architekt	Entwickler
Testplanung	Entwickler	Team	wenig Unterstützung durch Architekt	Qualitätssicherung
Änderungen	Entwickler	Product Owner	Architekt	Konfigurations-management
Statusberichte, Dokumentation	Alle Beteiligte, möglichst wenig Dokumentation	Product Owner, ScrumMaster	Architekt	mehrere Rollen
Bewertung	- leichtgewichtig - wenig Doku - agil, iterativ - Architektur wird von allen definiert - Funktioniert nur mit homogenem Team	- leichtgewichtig - wenig Overhead - agil, iterativ - Architektur wird vom Team mit Product Owner definiert	- schwergewichtig - formalisierter strukturierter Prozess - Use Case Driven - iterativ, - Klare Rolle für Architekten	- sehr formalisiert - Dokumenten-zentriert - Architekten-aufgaben auf mehrere Rollen verteilt

Solche und ähnliche Vergleiche werden gerne angestellt, obwohl es eigentlich nicht ganz korrekt ist, die Modelle in dieser Form zu vergleichen, da der Fokus der Modelle auf unterschiedlichen Ebenen liegt. Es ist zum Beispiel möglich innerhalb vom V-Modell eines der anderen Modelle für den Teil der Softwareentwicklung anzuwenden.

2. Lösungen UML

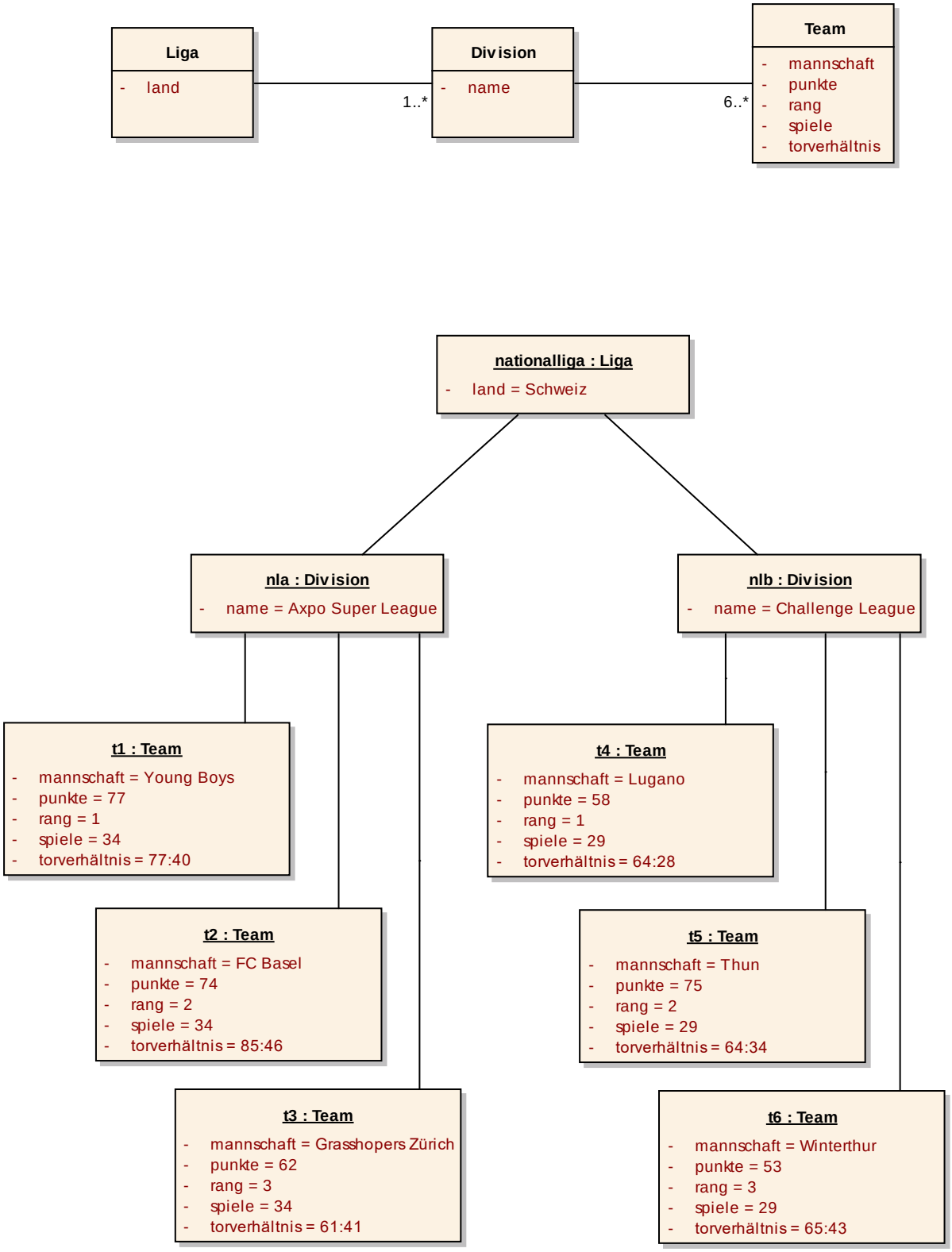
2.1. Pflegeheim (Klassendiagramm)

Lösungsvorschlag:



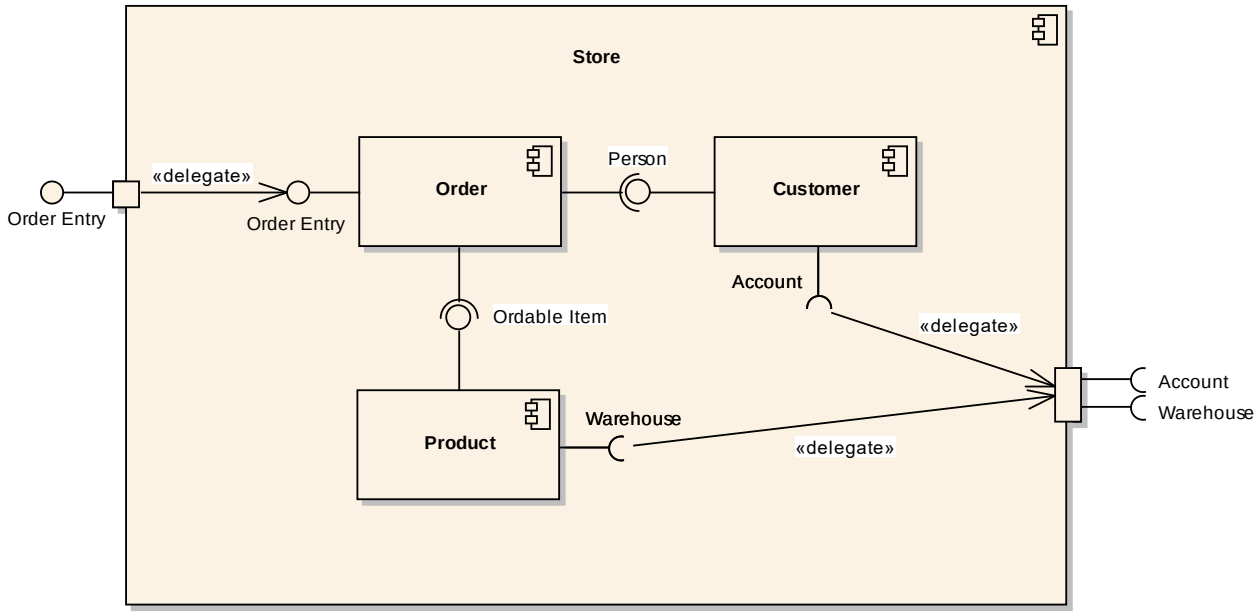
2.2. Nationalliga (Klassen- und Objektdiagramm)

Lösungsvorschlag:



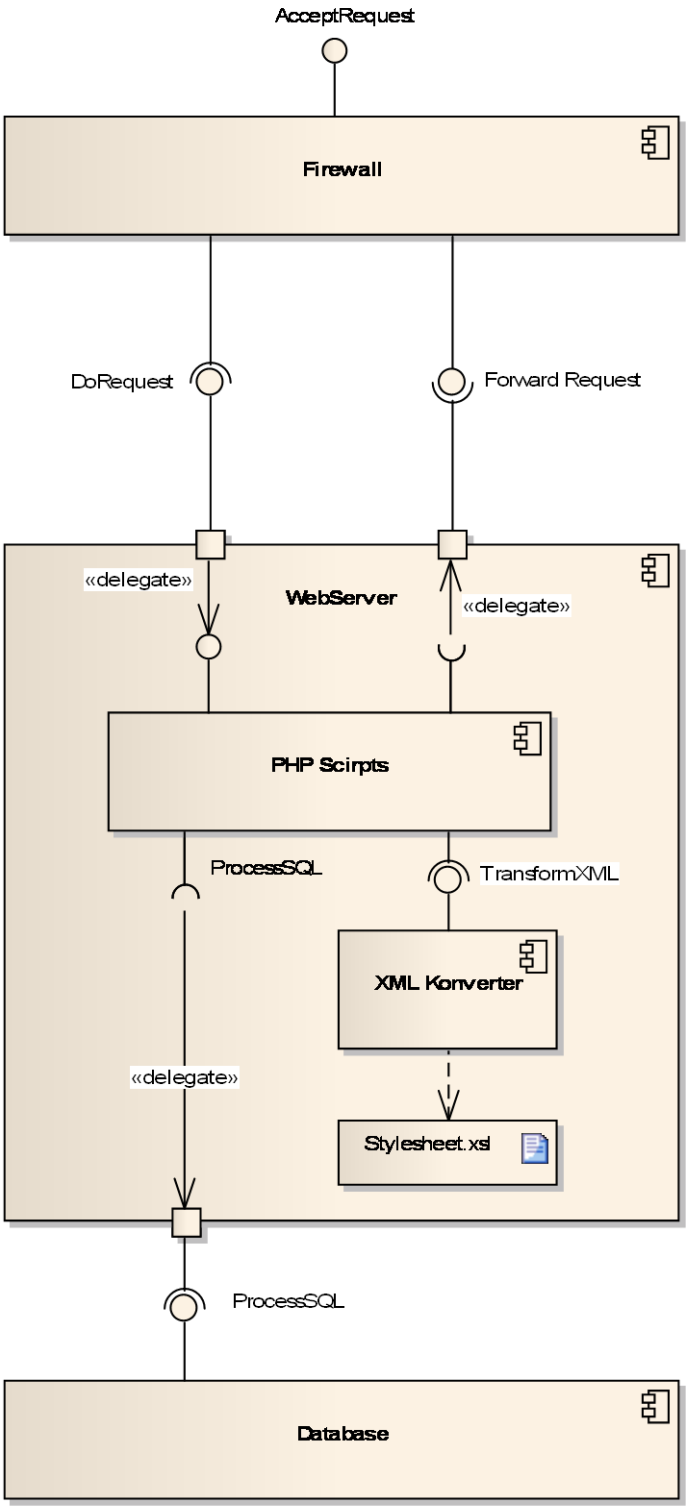
2.3. Store (Komponentendiagramm)

Lösungsvorschlag:



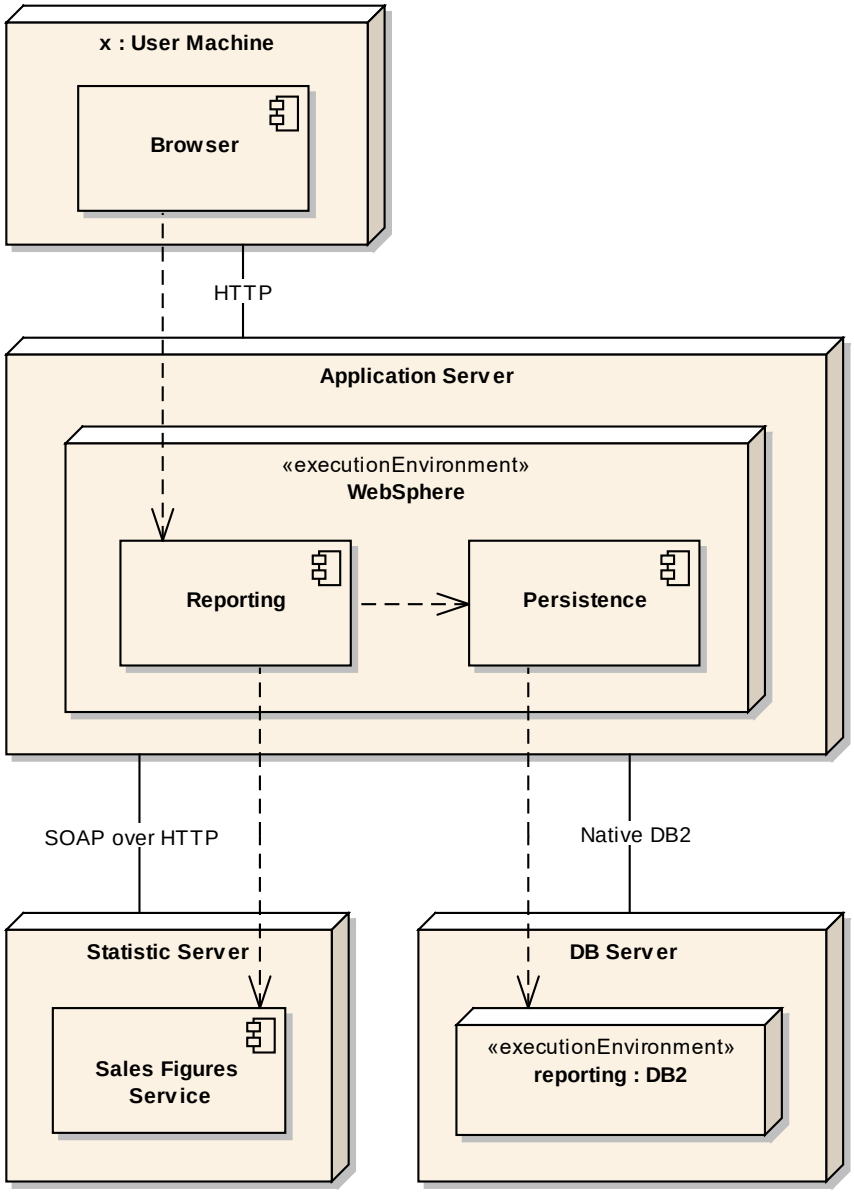
2.4. WebApp (Komponentendiagramm)

Lösungsvorschlag:



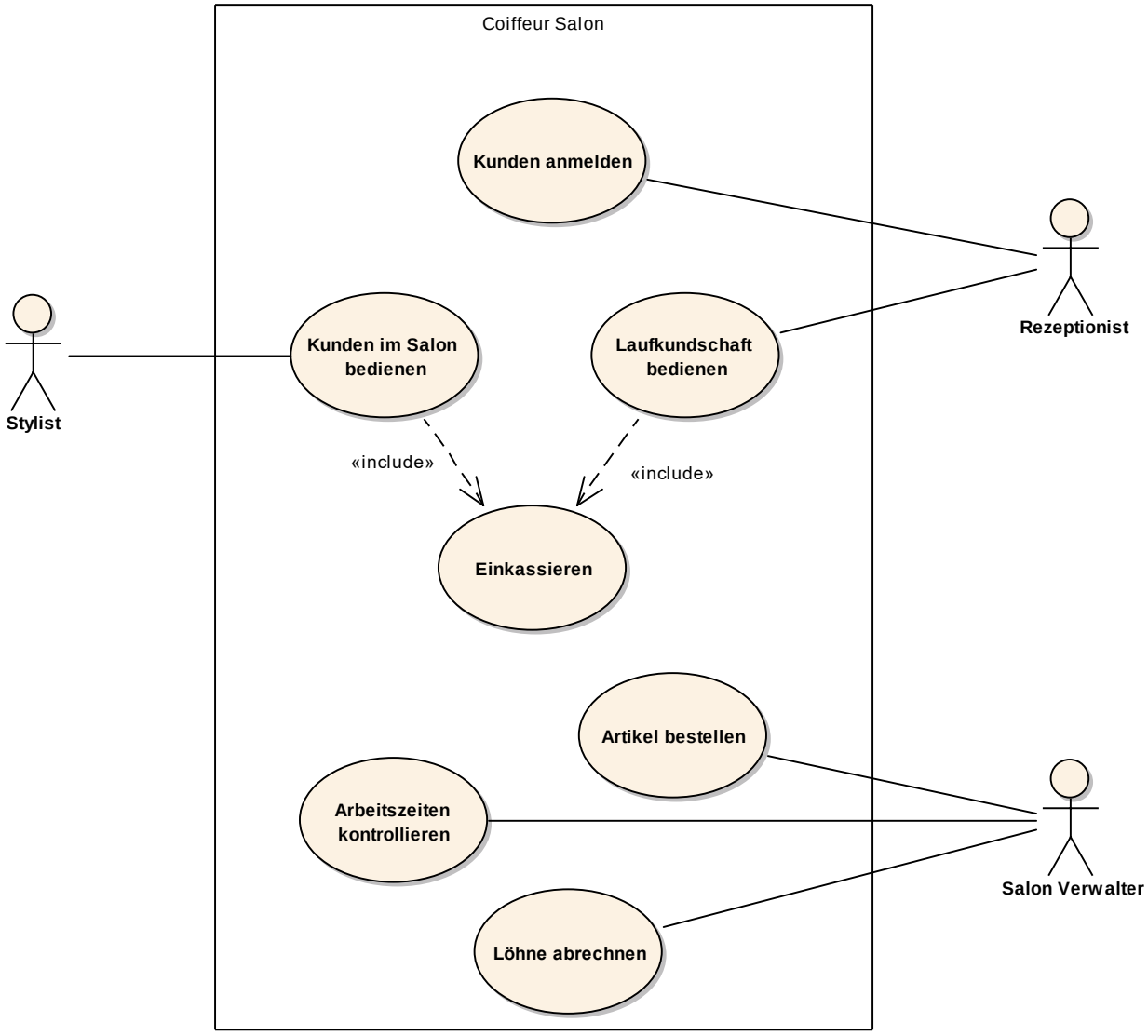
2.5. Reporting (Einsatz- / Verteildiagramm)

Lösungsvorschlag:



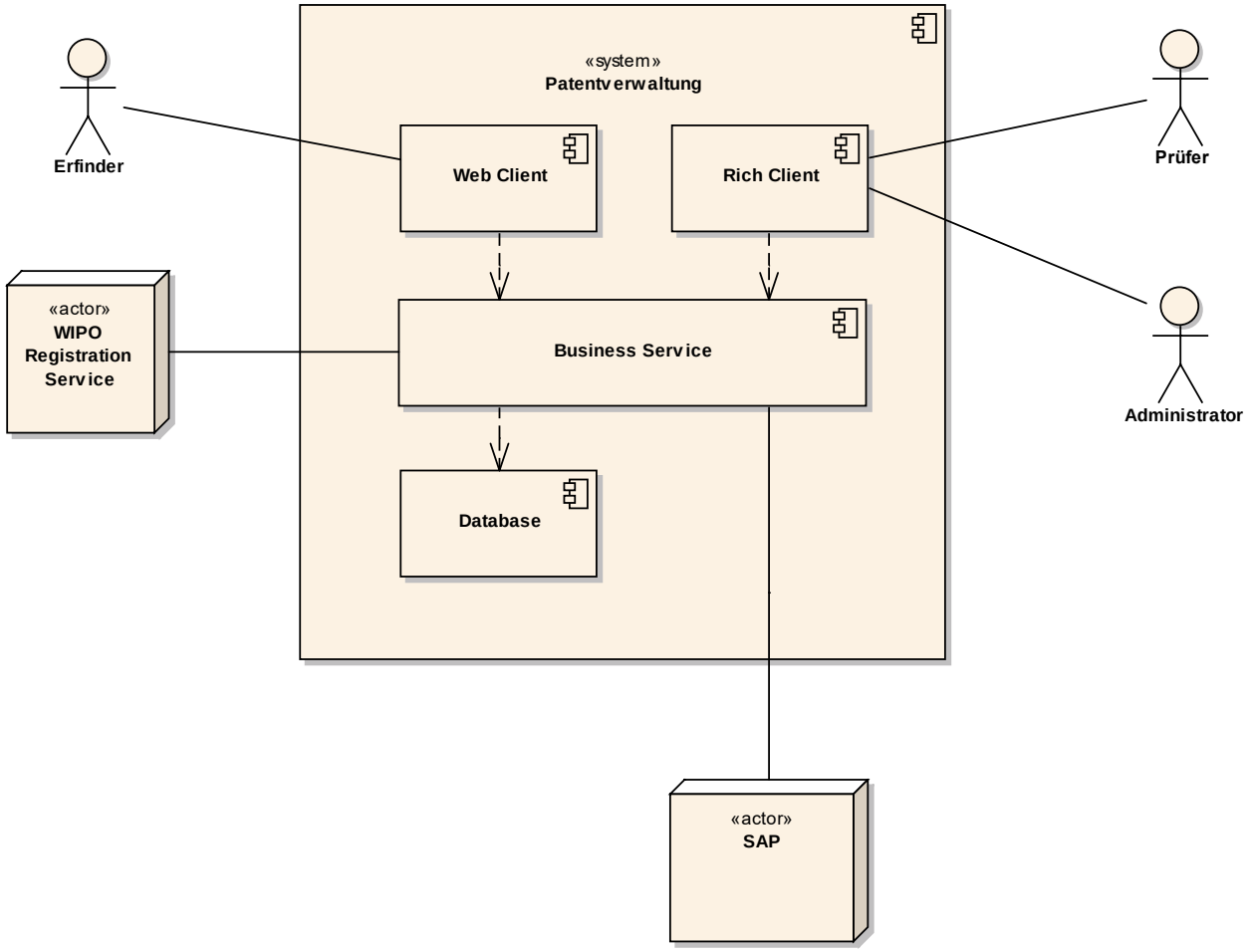
2.6. Coiffeur Salon (Anwendungsfall)

Lösungsvorschlag:



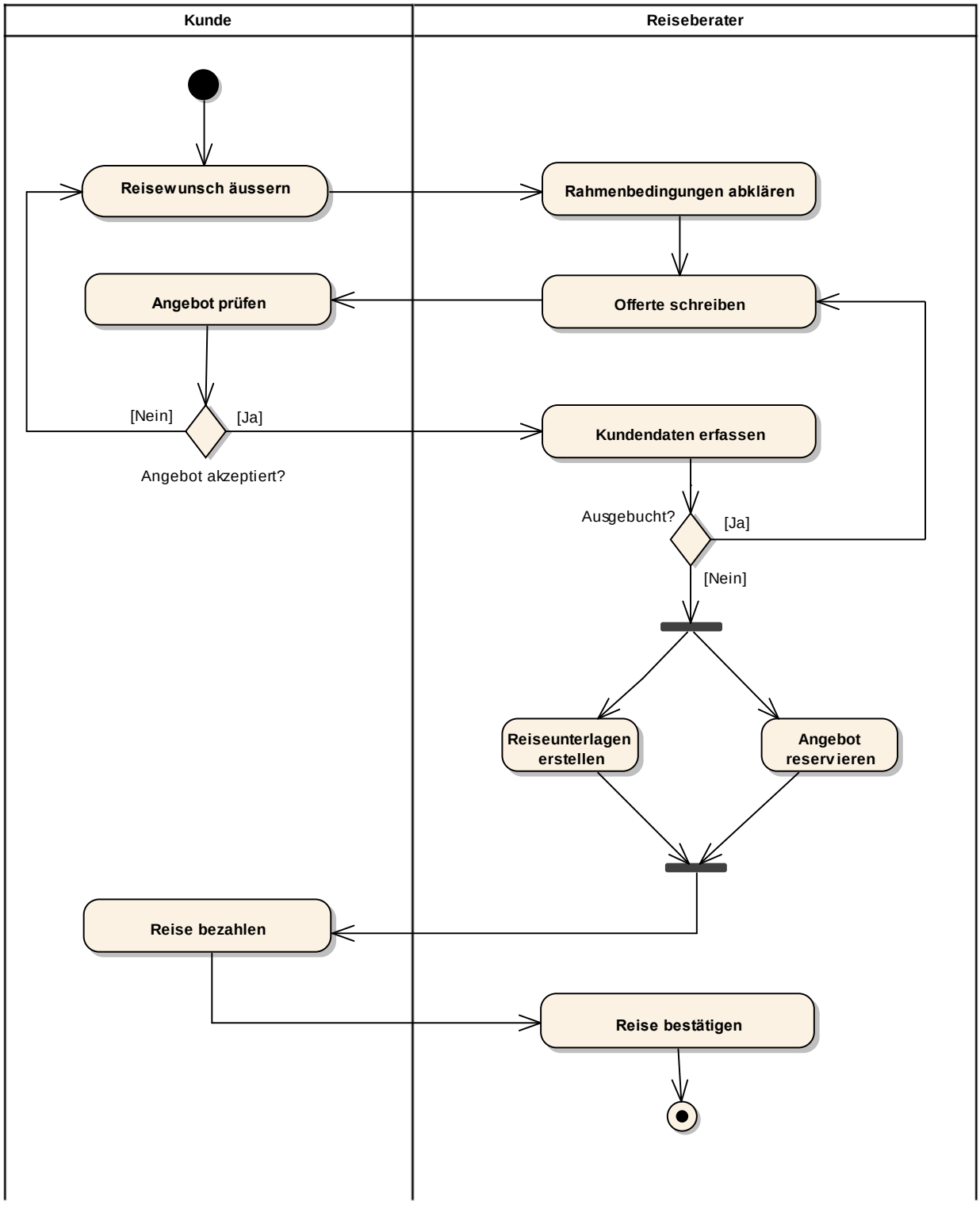
2.7. Patentverwaltung (Systemkontext)

Lösungsvorschlag:



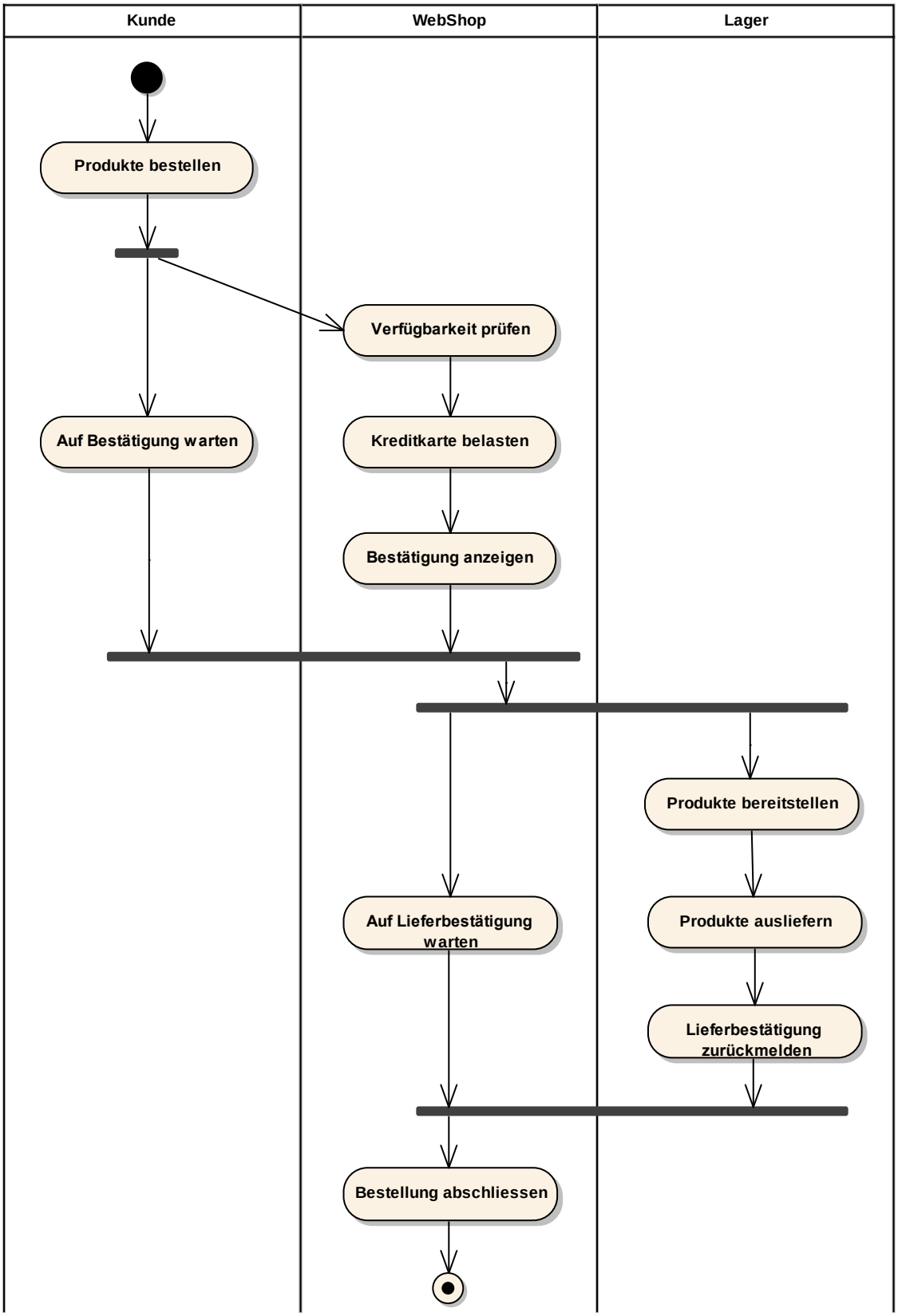
2.8. Reisebüro (Aktivitätsdiagramm)

Lösungsvorschlag:



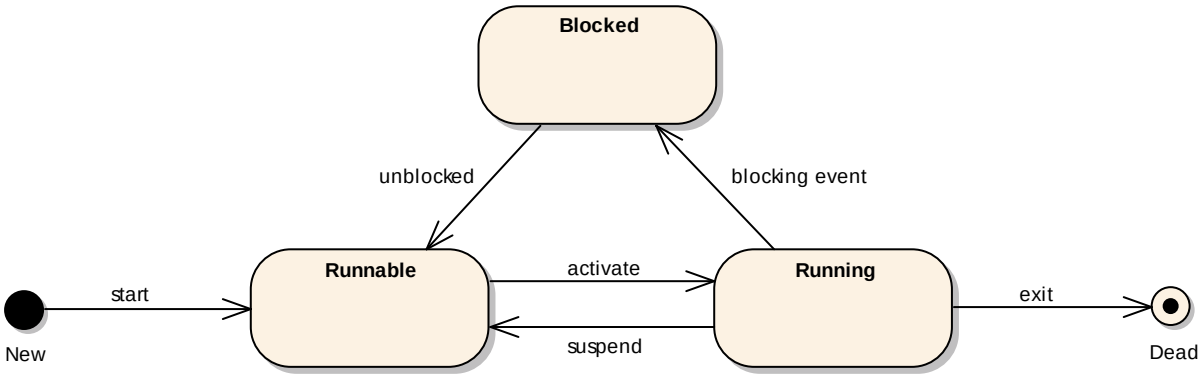
2.9. Bestellvorgang (Aktivitätsdiagramm)

Lösungsvorschlag:



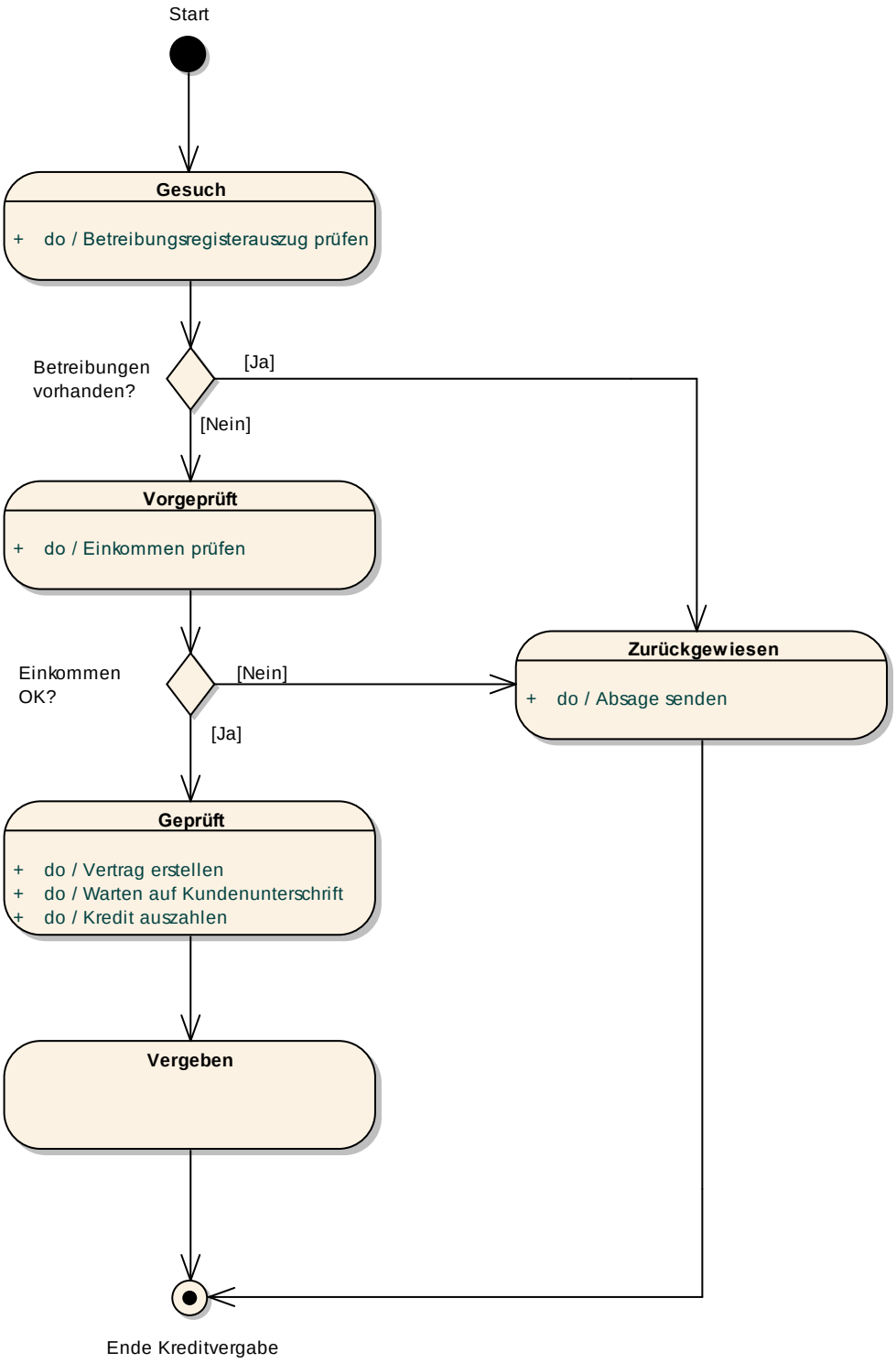
2.10. Thread (Zustandsdiagramm)

Lösungsvorschlag:



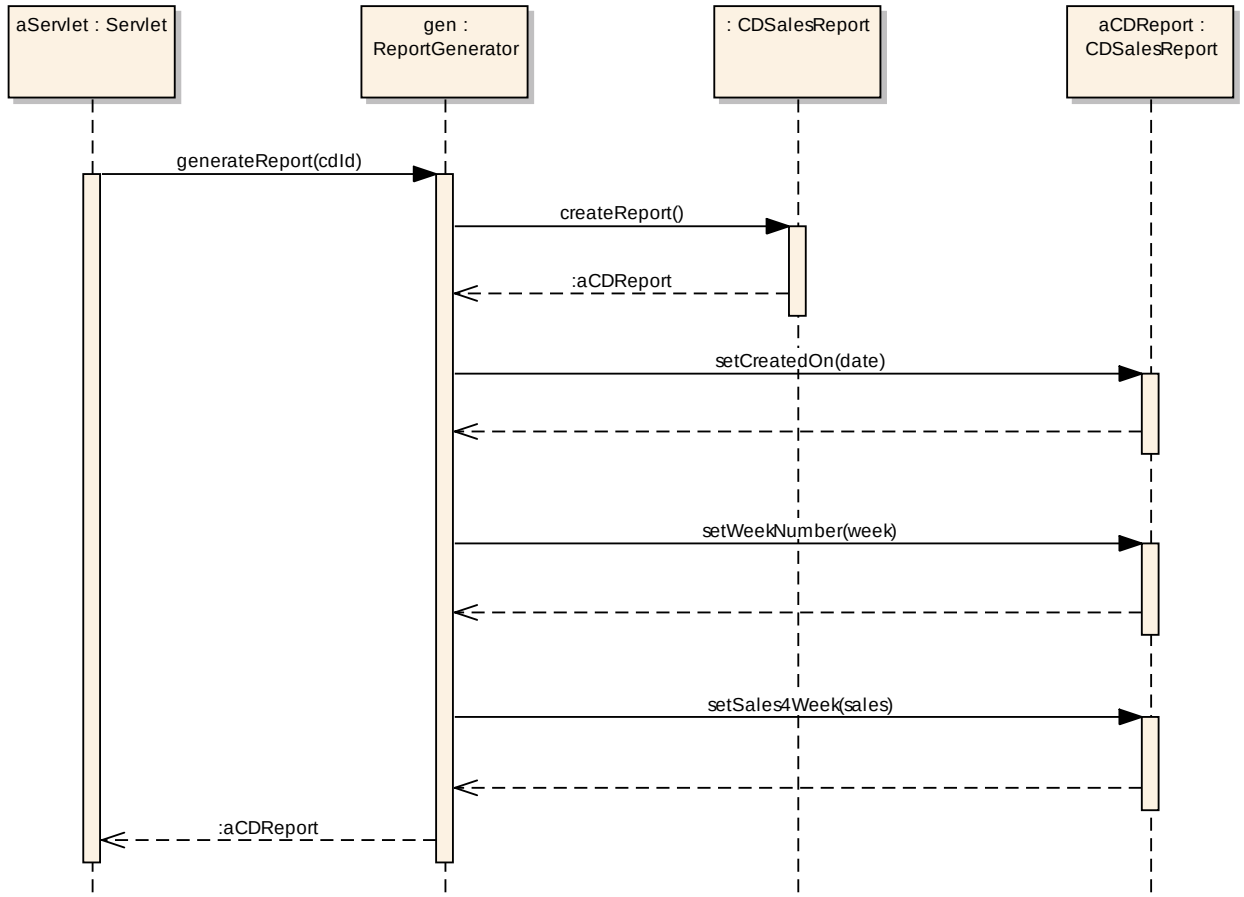
2.11. Kreditvergabe (Zustandsdiagramm)

Lösungsvorschlag:



2.12. CD Sales Report (Sequenzdiagramm)

Lösungsvorschlag:



Beachte:

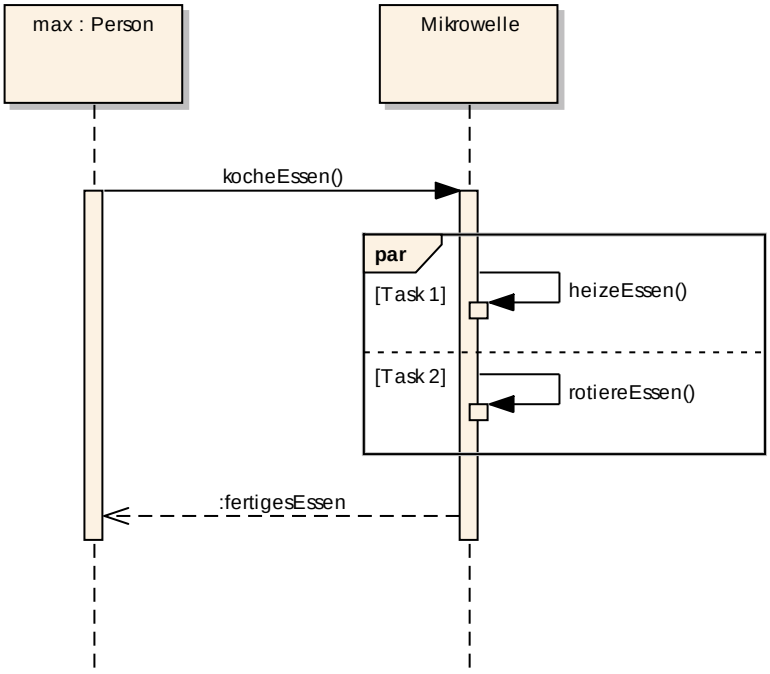
Zu beachten ist der Unterschied zwischen den beiden Elementen: CDSalesReport und aReport : CDSalesReport.

Beim ersten Element wird keine konkrete Instanz angesprochen. Diese wird zuerst mit der Factory Methode createReport()erzeugt.

Anschliessend wird mit der konkreten Instanz aCDReport (vom Typ CDSalesReport) kommuniziert.

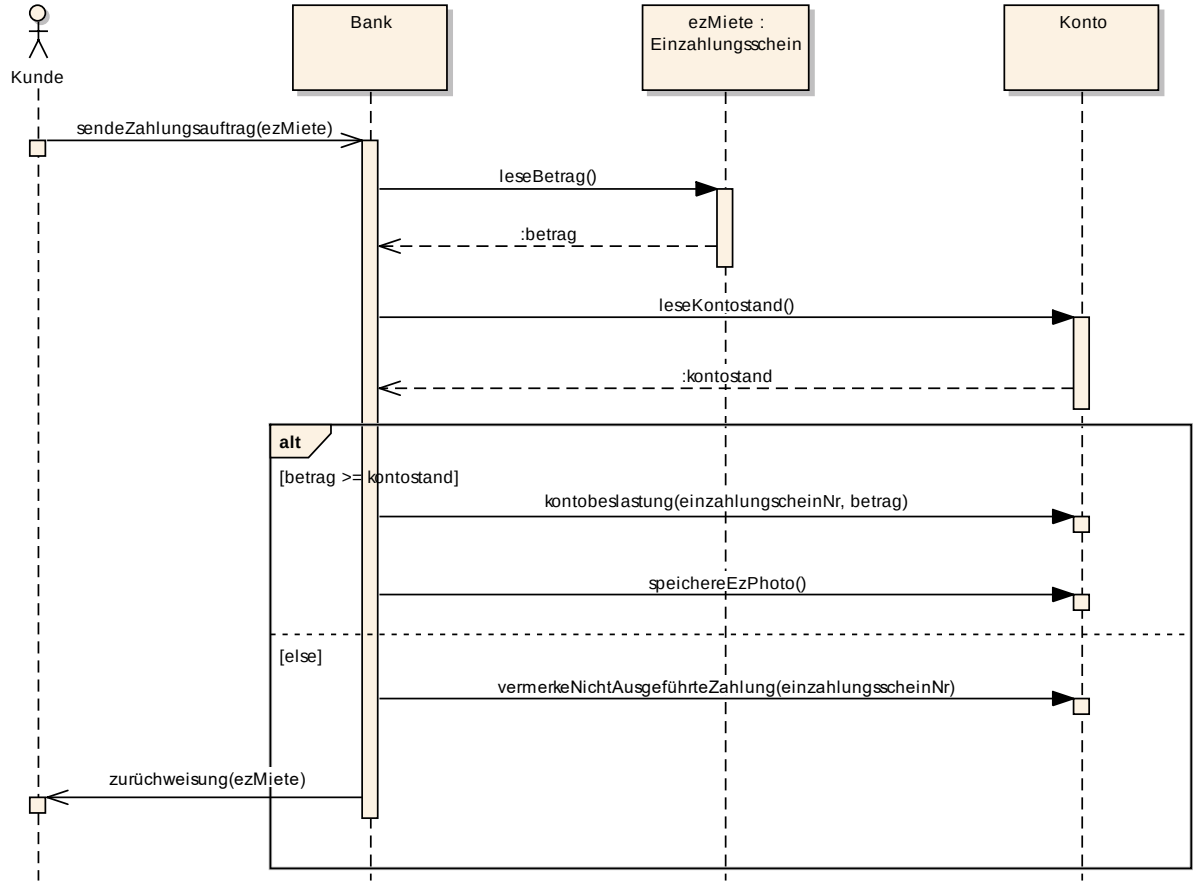
2.13. Mikrowelle (Sequenzdiagramm mit Nebenläufigkeit)

Lösungsvorschlag:

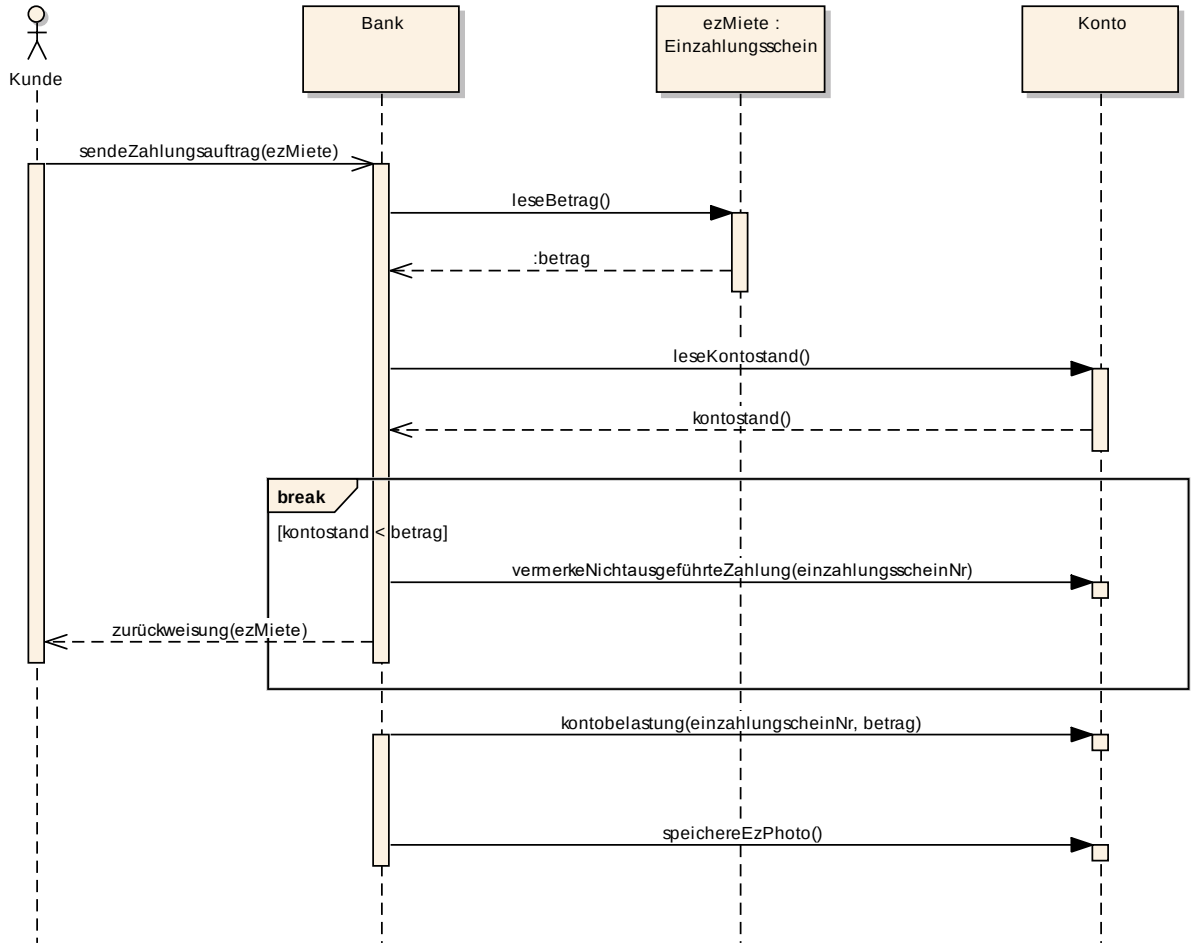


2.14. Zahlungsauftrag (Sequenzdiagramm mit alternativem Ablauf und Abbruch)

Lösungsvorschlag Aufgabe 1:

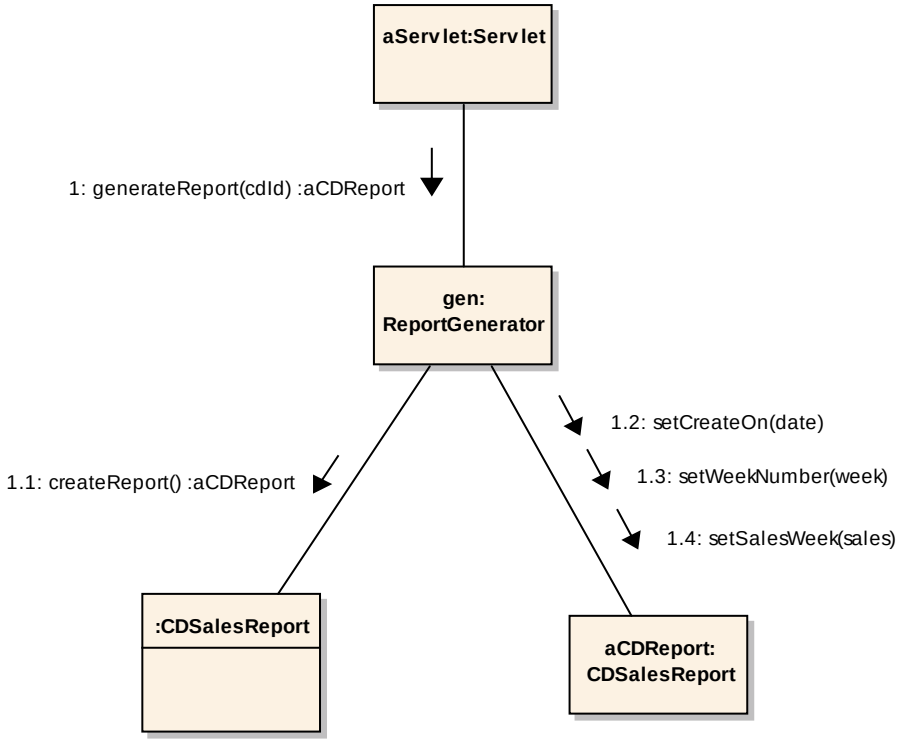


Lösungsvorschlag Aufgabe 2:



2.15. CD Sales Report (Kommunikationsdiagramm)

Lösungsvorschlag:



3. Lösungen Anaylse & Design

3.1. Fallstudie (Analyse und Design)

3.1.1. Vision

Einführung

Systeme zur Verwaltung von Kundenbeziehungen werden auch als Customer Relationship Management (CRM) Systeme bezeichnet. Sie haben zum Ziel die Kundenzufriedenheit und Bindung an das Unternehmen zu erhöhen. Generell unterscheidet man zwischen kommunikativen, analytischen und operativen CRM Systemen.

Geschäftsmöglichkeiten

Auf dem Markt gibt es bereits eine Vielzahl von etablierten CRM Produkten. Die meisten Systeme decken viele Aspekte des Marketings ab und sind für grosse Kundenstämme ausgelegt. Bei Systemen wie SAP oder Siebel ist das CRM sogar nur ein Bestandteil einer ganzen Modulpalette und voll integriert mit den anderen Komponenten. Kurz, die meisten CRM Systeme sind eher für mittlere und grosse Unternehmen ausgelegt.

In kleinen Firmen erfolgt die Kundenpflege oft manuell durch den Chef oder durch einige wenige leitende Angestellte. Für die Verwaltung der Kundendaten dient oft eine Excel Liste oder etwas Ähnliches. Eine gezielte Kontaktpflege, Protokollierung und Auswertung findet nicht statt. Dabei sind gerade für kleine Firmen gute Kontakte oft überlebenswichtig. Genau diese Marktlücke soll abgedeckt werden.

Mit Customer First soll ein smartes CRM System für kleine Unternehmen im Dienstleistungssektor erstellt werden.

Produkt

Das Produkt ermöglicht kleinen Unternehmen die gezielte Pflege von Kundenbeziehungen. Nachfolgende Tabelle zeigt die wichtigsten Funktionen und den Nutzen für die Kunden:

Funktion	*Kundennutzen*
Zentrale Verwaltung von Kundendaten	Einheitliche und vollständige Kundendaten. Kein Datenverlust bei einem Mitarbeiteraustritt.
Protokollierung von Kontakten	Nachvollziehbare Kundenbetreuung. Gezielter Aufbau von wichtigen Kundeninformationen
Auswertung der Kundenpflege	Gezielte Auswertung der aktuellen Kontaktpflege und Möglichkeit zum ergreifen von Massnahmen zur Förderung der Kontaktpflege basierend auf realen Auswertungen.
Einfache und schlanke Bedienung durch einen Webbrowser	Aktuelle Daten für alle Mitarbeiter innerhalb und ausserhalb der Firma.
Verwendung von Open Source Produkten	Geringe Kosten

Technologien

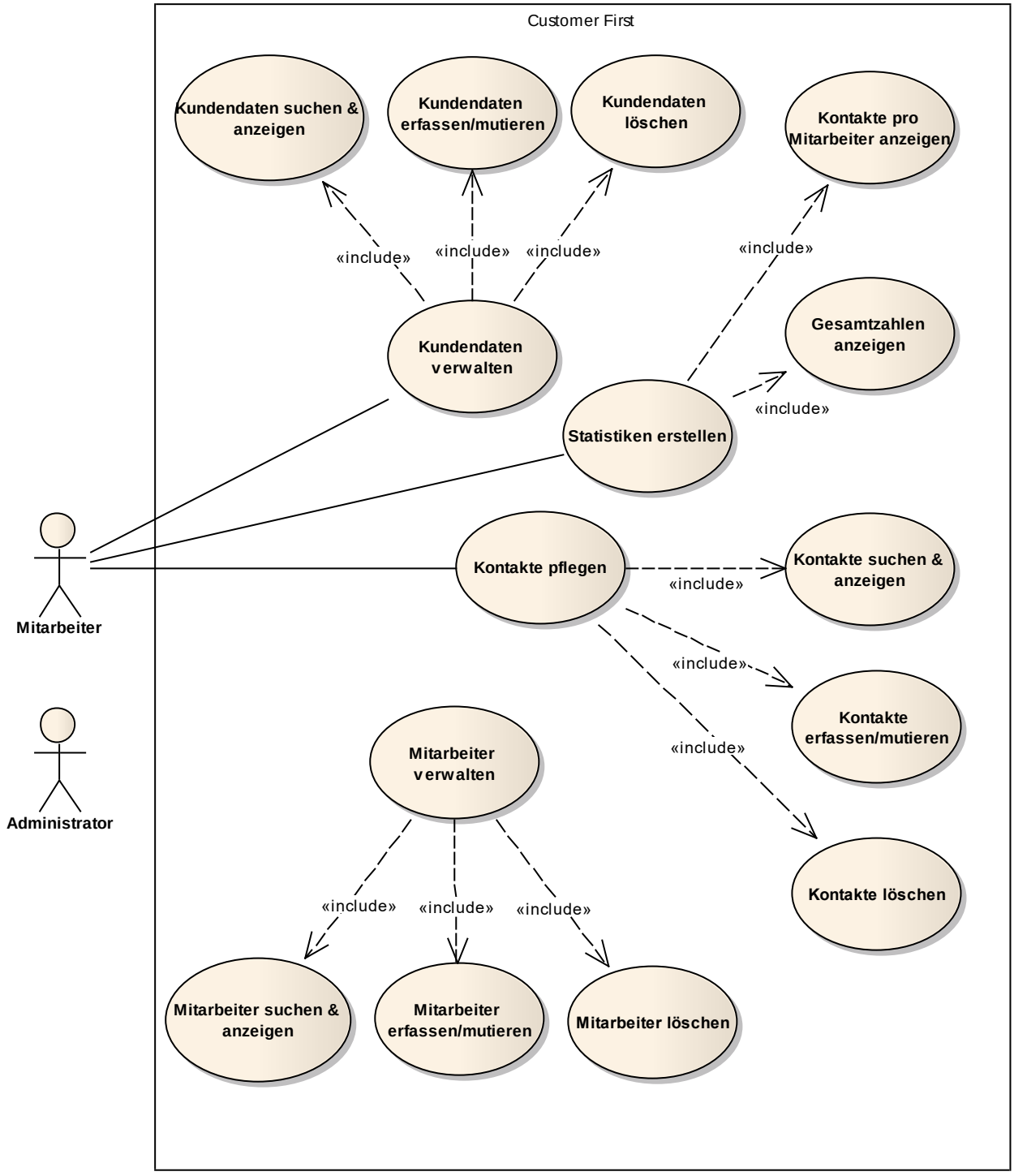
Customer First soll auf Basis einer vier Tier Architektur im Java EE Umfeld (mit Java Server Faces und Enterprise JavaBeans) erstellt werden. Als Produkte kommen Tomcat als Web Server, JBoss als Application Server und MySQL als Datenbank zum Einsatz.

3.1.2. Anwendungsfall Modell

Akteure:

Rolle	***Beschreibung***
Administrator	Der Administrator ist zuständig für die Verwaltung der Mitarbeiter im System.
Mitarbeiter	Der Mitarbeiter ist zuständig für die Verwaltung der Kunden und Kontakte. Zudem kann er Statistiken ausführen.

Anwendungsfall Diagramm:



Anwendungsfall Spezifikation:

Stellvertretend für alle Anwendungsfälle werden nachfolgend die drei Fälle aus dem Bereich „Kundendaten verwalten“ spezifiziert. Damit sind die für die Architektur relevanten Punkte abgedeckt.

Technische Fehler werden bei den alternativen Abläufen nicht behandelt, da Sie bei fast allen Anwendungsfälle auftreten können. Fehlt die Verbindung zum Server oder der Datenbank, so zeigt der Client eine Fehlermeldung an. Ist auch der Web Server nicht erreichbar, so zeigt der Browser an, dass die gewünschte Seite nicht erreicht werden kann.

Kundendaten suchen und anzeigen:

Beschreibung	Suche nach Kundendaten und Anzeige der Trefferliste und Detaildaten
Bereich	Kundendaten verwalten
Primäre Akteure	Mitarbeiter
Voraussetzungen	Keine
Normalverlauf	1) Der Mitarbeiter wählt die Funktion „Kundendaten anzeigen“ 2) Das System zeigt einen Suchdialog an 3) Der Mitarbeiter gibt die Suchkriterien ein und startet die Suche 4) Das System führt die Suche aus und zeigt die Resultate auf dem Bildschirm in einer Trefferliste an. *Werden keine Treffer gefunden: siehe Alternative Verläufe.* 5) Der Mitarbeiter wählt einen Treffer aus startet die Funktion „Detaildaten anzeigen“ 6) Das System lädt die Detaildaten zum ausgewählten Datensatz und zeigt diese auf dem Bildschirm an
Alternative Verläufe	Keine Treffer gefunden: 1) Das System gibt eine Meldung aus, dass keine Treffer gefunden wurden und zeigt den Suchdialog an. Anschliessend geht es mit Punkt 3 im Normalverlauf weiter Server oder Datenbank nicht vorhanden: 1) Das System gibt eine Fehlermeldung aus, dass der gewünschte Service nicht zur Verfügung steht.
Nachbedingungen	Keine
Spezielle Anforderungen	Keine

Kundendaten erfassen / mutieren:

Beschreibung	Neue Kundendaten erfassen oder bestehende mutieren
Bereich	Kundendaten verwalten
Primäre Akteure	Mitarbeiter
Voraussetzungen	Normalverlauf: keine Alternativer Ablauf – Bestehende Daten mutieren Es wurde eine Suche durchgeführt und auf der Trefferliste ein Datensatz selektiert. Alternativer Ablauf – Ungültige Eingaben keine
Normalverlauf	1) Der Mitarbeiter wählt die Funktion „Kundendaten erfassen“ 2) Das System zeigt eine leere Kundendaten Form an 3) Der Mitarbeiter gibt die Angaben zur Firma, die Adresse und mögliche Kontaktpersonen ein und drückt „Speichern“ 4) Das System prüft die Eingaben auf Ihre Gültigkeit und erstellt einen neuen Kundendatensatz. Dabei wird eine eindeutige ID generiert und angezeigt. Diese ID kann für weitere Funktionen verwendet werden. Sind die Eingaben ungültig: *siehe Alternative Verläufe*
Alternative Verläufe	Bestehende Daten mutieren: 1) Der Mitarbeiter wählt die Funktion „Kundendaten mutieren“ 2) Das System lädt die Daten und zeigt Sie in der Kundendaten Form 3) Der Mitarbeiter mutiert eines oder mehrere Felder und drückt „Speichern“ 4) Das System prüft die Eingaben auf Ihre Gültigkeit und aktualisiert die mutierten Daten. Sind die Eingaben ungültig: *siehe Alternative Verläufe* Ungültige Eingaben: 1) Das System zeigt eine Fehlermeldung am oberen Rand der Seite und darunter die Kundendaten Form. Anschliessend geht es mit dem vorherigen Punkt weiter.
Nachbedingungen	Keine
Spezielle Anforderungen	Die Kundendaten werden innerhalb einer Transaktion gespeichert oder aktualisiert, so dass keine Inkonsistenten Datensätze entstehen.

Kundendaten löschen:

Beschreibung	Besehende Kundendaten löschen
Bereich	Kundendaten verwalten
Primäre Akteure	Mitarbeiter
Voraussetzungen	Normalverlauf Es wurde eine Suche durchgeführt und eine Trefferliste angezeigt.
Normalverlauf:	1) Der Mitarbeiter wählt die Funktion „Kundendaten löschen“ 2) Das System zeigt die Kundendaten und die Frage, ob die Daten wirklich gelöscht werden sollen. 3) Der Mitarbeiter drückt „Löschen“ 4) Das System löscht die Kundendaten.
Alternative Verläufe	Keine
Nachbedingungen	Keine
Spezielle Anforderungen	Die Kundendaten werden innerhalb einer Transaktion gelöscht so dass keine Inkonsistenten Datensätze entstehen.

3.1.3. Anforderungen

Die Anforderungen werden mit einer eindeutigen Id gekennzeichnet. Diese wird gleichzeitig für die Gruppierung der Anforderungen genutzt, so dass zusammengehörende Funktionsblöcke gebildet werden können.

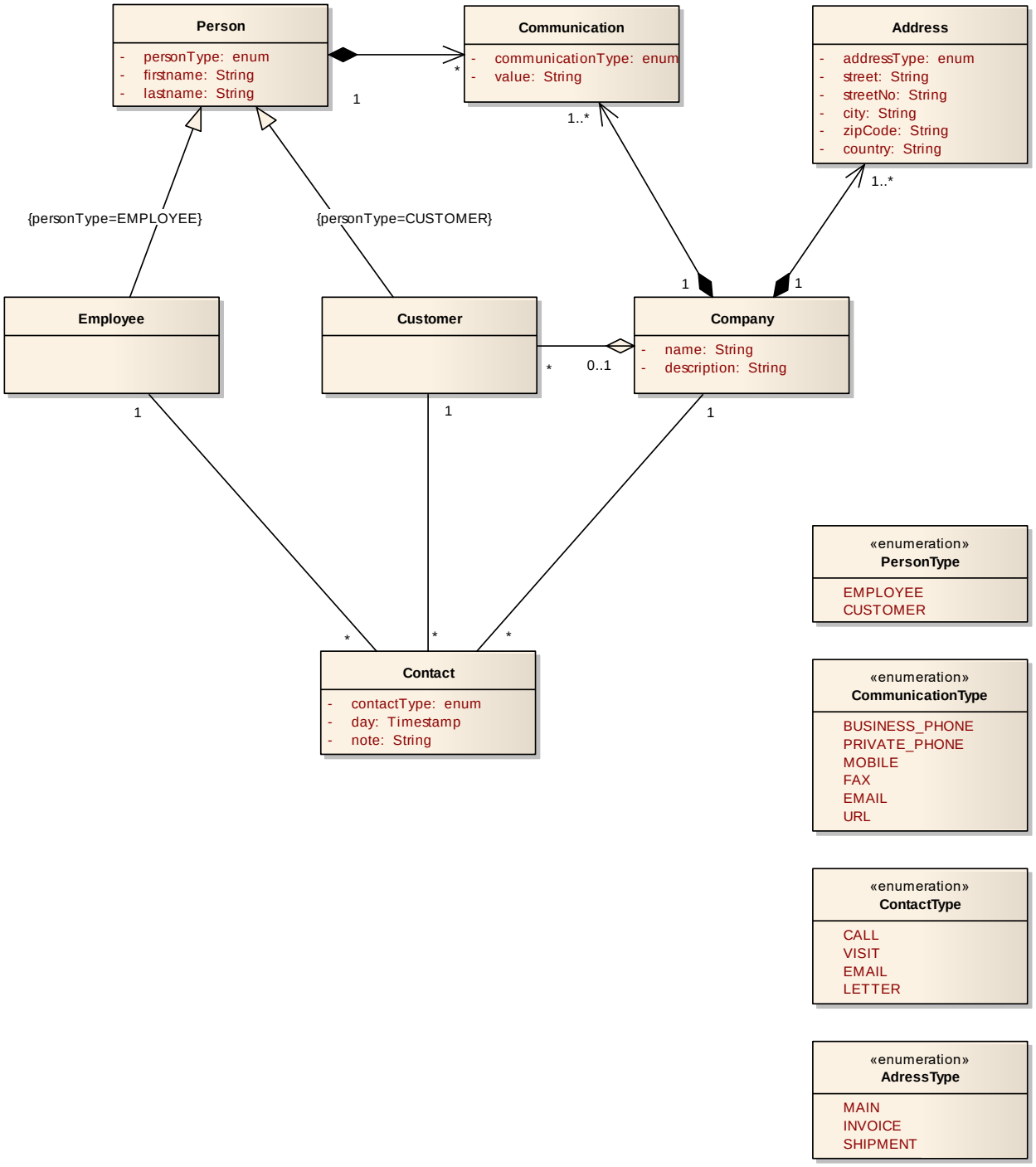
Jede Anforderung wird mit einer Priorität (Hoch, Mittel oder Tief) und Verbindlichkeit (Muss, Soll oder Kann) klassifiziert.

Die Anforderungen werden nur grob beschrieben. Die weiteren Details werden während der Design Phase definiert.

Id	*Prio.*	*Verb.*	*Anforderung*
A01 – Verwalten von Kundendaten			
A01.01	Hoch	Muss	Das System bietet die Möglichkeit bieten, Kundendaten zu suchen und anzuzeigen.
A01.02	Hoch	Muss	Das System erlaubt es dem Benutzer, neue Kundendaten zu erfassen. Die Kundendaten beinhalten Angaben zur Firma, Adresse und den Kontaktpersonen.
A01.03	Hoch	Muss	Das System ermöglicht, bestehende Kundendaten zu mutieren.
A01.04	Hoch	Soll	Falls bestehende Kundendaten nicht mehr relevant sind, bietet das System die Möglichkeit, diese zu löschen.
A02 – Pflege von Kontakten			
A02.01	Hoch	Muss	Das System bietet die Möglichkeit bieten, durchgeführte Kundenkontakte zu einer bestimmten Firma zu suchen und anzuzeigen.
A02.02	Hoch	Muss	Das System ist in der Lage sein, einen Kundenkontakt zu erfassen. Ein Kundenkontakt besteht dabei aus Firma, Kontaktperson, Datum, Kontaktart (Besuch, Anruf, Brief, Mail), Notiz und dem Namen des Mitarbeiters, welcher den Kontakt „durchgeführt“ hat.
A02.03	Hoch	Muss	Das System bietet dem Benutzer die Möglichkeit, bestehende Kundenkontakte zu mutieren.
A02.04	Hoch	Soll	Falls erfasste Kontaktdaten nicht mehr relevant sind, sollte das System dem Benutzer die Möglichkeit bieten, diese zu löschen.
A03 – Verwalten von Mitarbeiterdaten			
A03.01	Hoch	Muss	Das System muss die Möglichkeit bieten, Mitarbeiter der Firma zu suchen und anzuzeigen.
A03.02	Hoch	Muss	Das System erlaubt es dem Benutzer, neue Mitarbeiter zu erfassen. Die Mitarbeiterdaten beinhalten Vor- und Nachname sowie Kommunikationsdaten wie Telefon, Natel und Email Adresse.
A03.03	Hoch	Muss	Das System ermöglicht, bestehende Mitarbeiterdaten zu mutieren.
A03.04	Hoch	Soll	Das System ermöglicht, bestehende Mitarbeiterdaten zu löschen

Id	*Prio.*	*Verb.*	*Anforderung*
A04 – Erstellung von Statistiken			
A04.01	Tief	Soll	Das System kann dem Benutzer eine Auswertung mit dem Total der erfassten Firmen, Personen und Kontakte zur Verfügung stellen.
A04.02	Tief	Soll	Das System kann dem Benutzer eine Auswertung der Kontakte pro Mitarbeiter anzeigen.
A10 – Technologie und Architektur			
A10.01	n/a	Soll	Das System soll mit einer drei Tier Architektur auf Basis der Java EE Technologie realisiert werden.
A10.02	n/a	Muss	Auf dem Web Server wird Tomcat verwendet.
A10.03	n/a	Muss	Als Application Server wird JBoss verwendet.
A10.04	n/a	Kann	Als Datenbank kommt MySQL zum Einsatz.
A10.05	na	Soll	Die Aufteilung der Applikation in eine Präsentations-, Logik und Datenschicht soll klar ersichtlich sein.
A10.06	n/a	Kann	Die Applikation soll die im Java EE Umfeld bekannten und sinnvollen Design Patterns einsetzen.
A10.07	n/a	Muss	Die Business Logik soll mit Enterprise JavaBeans realisiert werden
A10.08	n/a	Soll	Für die Implementation der Präsentationslogik kann Java Server Faces verwendet werden.
A20 – Nichtfunktionale Anforderungen			
A20.01	Mittel	Muss	Das System soll von einem Benutzer mit Web Browser Kenntnissen ohne zusätzliche Schulung genutzt werden können
A20.02	Tief	Soll	Es soll die jeweils neuste Version vom Internet Explorer und Firefox Web Browser unterstützt werden
A20.03	Mittel	Soll	Die Antwortzeiten für die Benutzeraktionen sollen im Intranet maximal eine Sekunde betragen.
A20.04	Tief	Soll	Das System soll bis zu 20 simultane Benutzer unterstützen ohne die vorgegebenen Antwortzeiten zu überschreiten.
A20.05	Hoch	Soll	Die Applikation soll mit einem normalen Browser bedient werden können. Es sind keine zusätzlichen Installationen auf dem Client PC notwendig.
A20.06	Tief	Kann	Es sind die Java Code Conventions von Sun Microsystems anzuwenden.
A20.07	Hoch	Muss	Es werden ausschliesslich Open Source Produkte verwendet

3.1.4. Fachklassen

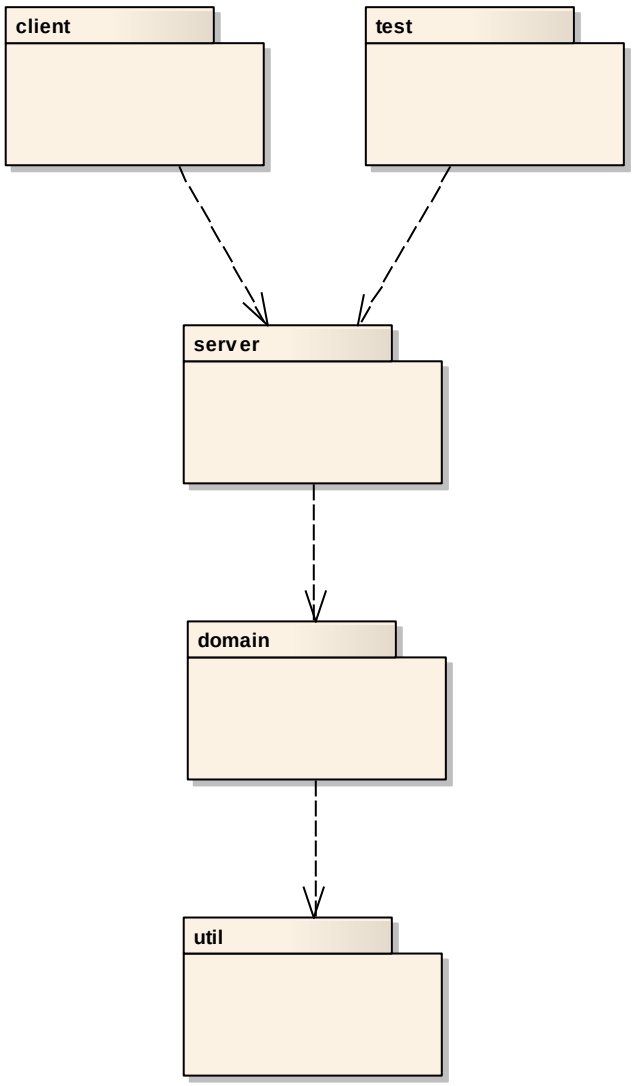


Bemerkung:

- Eine Firma kann pro AdressType eine Adressen Instanz besitzen.
- Eine Firma oder eine Person kann pro CommunicationType eine Communication Instanz besitzen.

3.1.5. Pakete

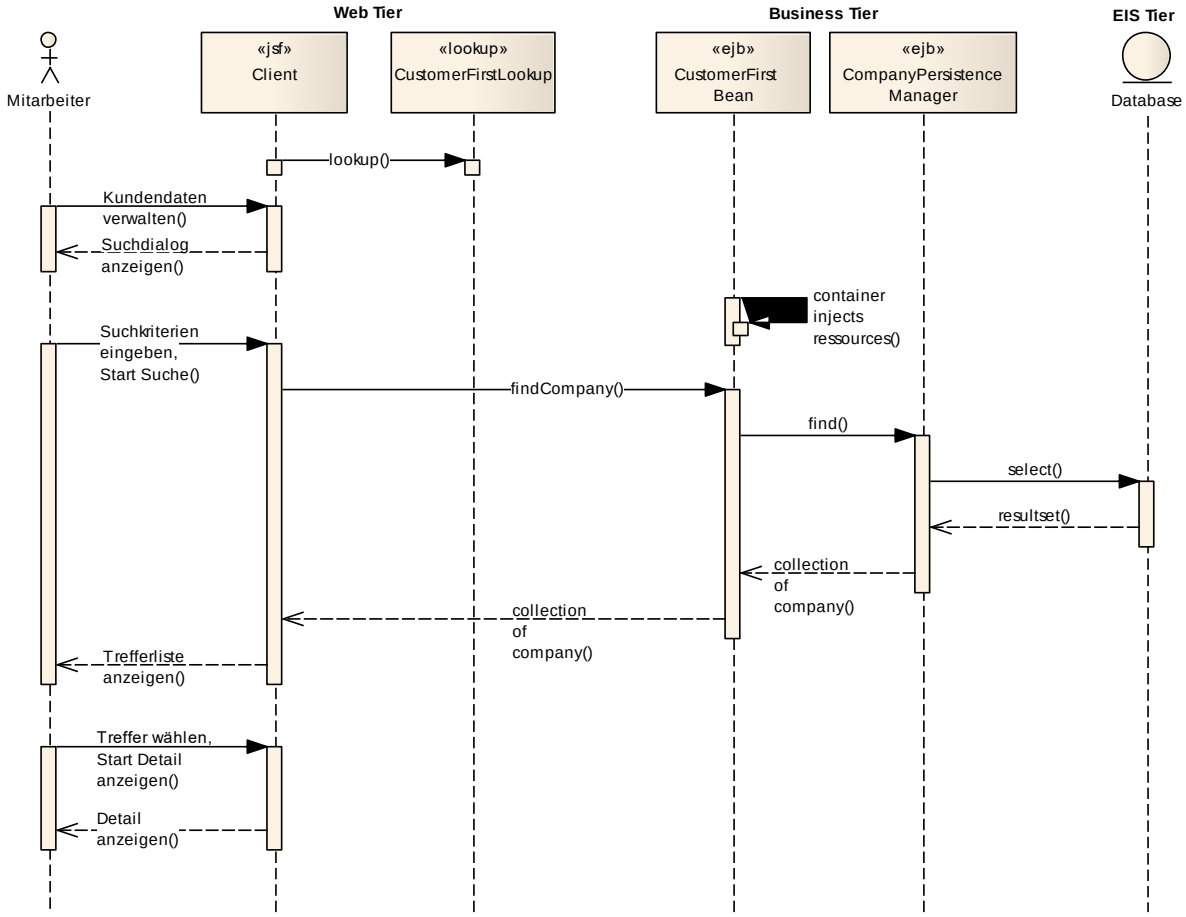
Die Software wird in folgende Top Level Packages unterteilt:



Paket	*Beschreibung*
util	Enthält die Klassen der Präsentationslogik für den Web Client, welcher mit Java Server Faces realisiert wird.
domain	Enthält JUnit Testklassen für Unit und Integrationstest des Servers, sowie ein Ladeprogramm für die Testdaten.
server	Enthält die Klassen zum Verwalten der Business Objekte und steuern der Anwendung auf den Application Server. Stellte eine Delegate Klasse mit allen Business Funktionen für den Client zur Verfügung.
client	Domänen Package der Applikation. Beinhaltet die Klassen der Business Objekte.
test	Utilitiy Package mit allgemeinen Library und Helper Klassen. Kann von allen Packages verwendet werden.

3.1.6. Aktivitäten

Die verwendeten Java EE Pattern werden anhand des Use Case „Kundendaten suchen und anzeigen“ mit Hilfe eines Sequenzdiagrames aufgezeigt. Alle anderen Use Case funktionieren vom Ablauf gleich oder sehr ähnlich. Das Diagramm zeigt in erster Linie den Ablauf des Server Teils, der Ablauf auf dem Client wird nicht detailliert aufgezeigt.



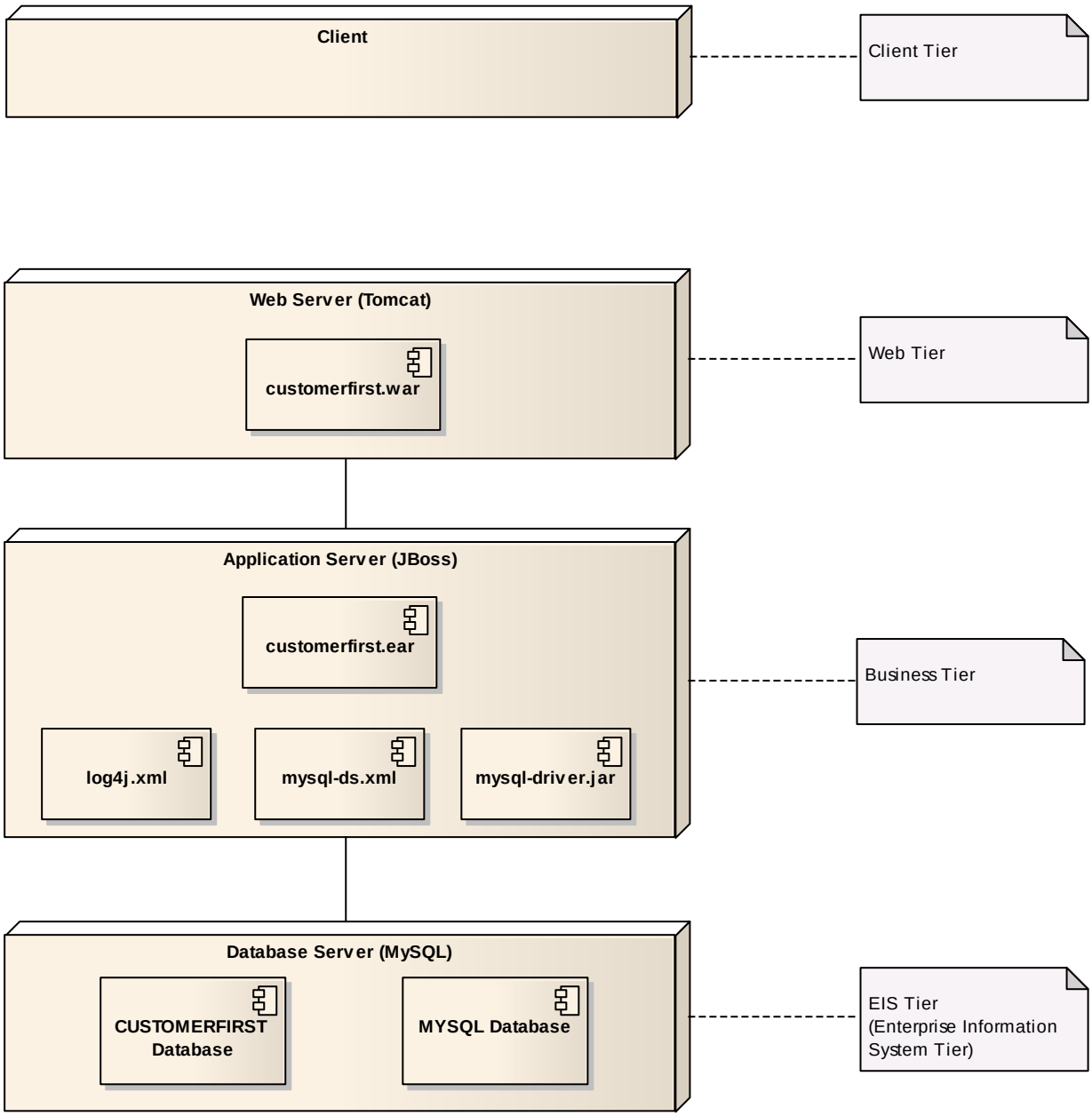
Bemerkung:

- Die Lookup Klasse liefert dem Client eine Instanz des CustomerFirst Remote Interfaces (via JNDI Lookup wird direkt das Remote Interface abgerufen).
- Das Customer First Bean fungiert als Facade. Die benötigte Local Interface Referenz auf den Company Persistence Manager wird durch den Container beim Erstellen des Bean "eingesprizt".
- Die Company Klasse ist POJO und Entity in einem und wird vom Company Persistence Manager für alle Persistenz relevanten Aktionen verwaltet.
- Die Aufbereitung der Detaildaten erfolgt nur noch auf dem Web Tier, da die Collection der Company Entitäten die vollständigen Daten enthält. Alternativ könnten die Detaildaten mit einem weiteren Aufruf auf den Business Tier selektiert werden. Dadurch würde die Trefferliste schlanker.

3.1.7. Verteilungsmodell

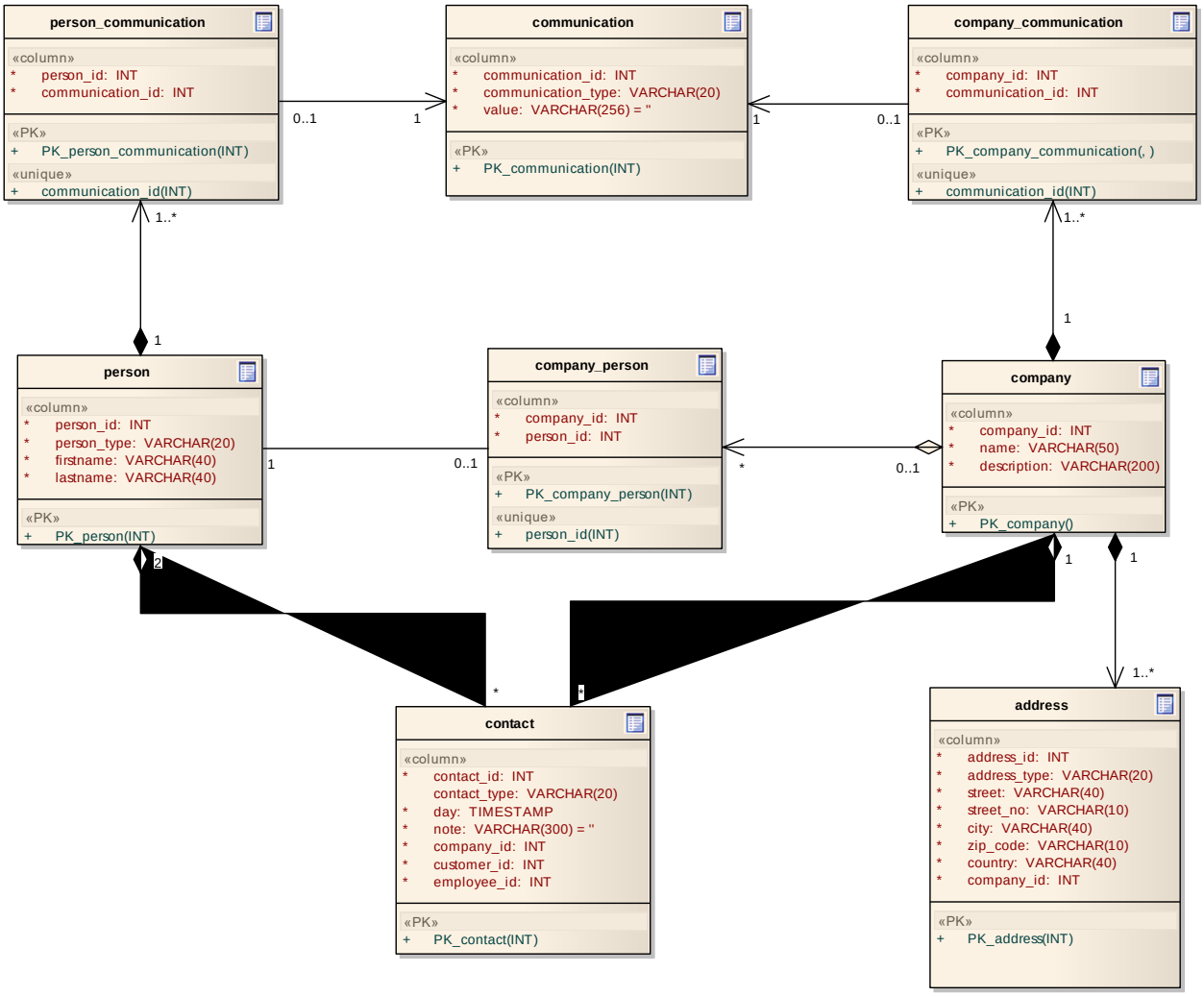
Das Einsatz-/Verteilungsdiagram zeigt die Hardwarekonfiguration von Customer First sowie die Verteilung der ausgelieferten Softwarekomponenten auf die verschiedenen Einheiten.

Die Komponenten des Client Tier werden nicht näher beschreiben, da hier, je nach Benutzer, eine beliebige Konfiguration möglich ist.



3.1.8. Datenmodell

Nachfolgend das ERD der MySQL Datenbank.



Bemerkung:

- Folgende One To Many Beziehungen werden mit einer Join Tabelle realisiert: Company --> Communication, Company --> Person, Person --> Communication
- Die One To Many Beziehung Company --> Address wird über den Foreign Key company_id in der Tabelle Address realisiert.
- Die Abbildung von Employee und Customer aus dem Domain Model erfolgt in der Tabelle Person über das Attribut person_type, welches in Java als Enum implementiert ist.
- Die Tabelle Contact hat eine Referenz auf die Tabelle Company und zwei Referenzen auf die Tabelle Person (Employee und Customer).

3.2. FIRST (Test und Implementation)

Die Abkürzung FIRST steht für die folgenden fünf Test Prinzipien:

- **F**ast

  "Many hundreds or thousands per second"

  Tests must be fast. If you hesitate to run the tests after a simple one-liner change, your tests are far too slow. Make the tests so fast you don't have to consider them.

- **I**solates

  „Failure reasons become obvious“

  Tests isolate failures. A developer should never have to reverse-engineer tests or the code being tested to know what went wrong. Each test class name and test method name with the text of the assertion should state exactly what is wrong and where. If a test does not isolate failures, it is best to replace that test with smaller, more-specific tests.

- **R**epeatable

  “Run repeatable in any order, any time”

  Tests must be able to be run repeatedly without intervention. They must not depend upon any assumed initial state, they must not leave any residue behind that would prevent them from being re-run. This is particularly important when one considers resources outside of the program's memory, like databases and files and shared memory segments.

- **S**elf validating

  “No manual evaluation required”

  Tests are pass-fail. No agency must examine the results to determine if they are valid and reasonable. Authors avoid over-specification so that peripheral changes do not affect the ability of assertions to determine whether tests pass or fail.

- **T**imely

  “Written before the code”

  Tests are written at the right time, immediately before the code that makes the tests pass. While it seems reasonable to take a more existential stance, that it does not matter when they're written as long as they are written, but this is wrong. Writing the test first makes a difference.

Für weitere Details siehe zum Beispiel http://agileinaflash.blogspot.com/2009/02/first.html oder „Clean Code“ [17].