In diesem Kapitel:

17

ROP / RUP Analyse Case Study Bibliothek

Zur Illustration und besseren Vorbereitung auf das Vordiplom wollen wir ein Projekt von Anfang bis zu Ende mit der Methode (ROP / RUP) behandeln.

Als Hilfsmittel verwenden wir (die Links funktionieren leider nicht, da die Verzeichnisse pro Benutzer unterschiedlich sind):

Rational Rose und

Rational Unifed Process

1.1. Kurze Übersicht über die einzelnen Kapitel

1.1.1. Projektbeginn

In diesem Kapitel

- beschreiben wir das Projekt
- liefern etwas Hintergrundinformationen
- beschreiben einige Risiken des Projektes
- beschreiben die Aufgabenstellung

1.1.2. Beschreibung der Anwendungsfälle (Use Cases) :Anforderungsanalyse

In diesem Kapitel

- beschreiben wir das Systemverhalten
- die Beteiligten (Actors, Aktoren, Anwender)
- die Anwendungsfälle (Use Cases)
- die Beziehungen (Relationen) der Anwendungsfälle untereinander
- die Anwendungsfall-Diagramme (Use Case Diagrams)

1.1.3. Finden der Klassen : Domänenanalyse

In diesem Kapitel

- fassen wir kurz die Merkmale von Objekten zusammen

- beschreiben wir Zustand, Verhalten und die Identität von Objekten
- befassen wir uns mit generellen Merkmalen von Klassen
- beschreiben wir, wie man Klassen finden kann
- dokumentieren wir einige Klassen
- fassen mehrere Klassen zu Paketen (Packages) zusammen
- und fassen unsere Ergebnisse in einem Klassendiagramm zusammen

1.1.4. Aufzeigen von Objekt-Interaktionen

In diesem Kapitel

- zeigen wir, wie Anwendungsfälle umgesetzt werden (Use Case Realization)
- beschreiben wir Szenarios
- formalisieren wir Szenarios mit Hilfe von Sequenz Diagrammen (Sequence Diagrams)
- beschreiben wir Systemgrenzen mit Hilfe von Systemgrenzklassen (Boundary Classes)
- beschreiben wir Kollaborationsdiagramme und zeigen deren Zusammenhang mit den Sequenzdiagrammen

1.1.5. Festhalten von Beziehungen

In diesem Kapitel

- beschreiben wir unterschiedliche Formen der Beziehung:
  1. Assoziation
  2. Aggregation
- beschreiben wir Rollen und Mächtigkeit der Beziehungen
- zeigen wir auf, wie Beziehungen gefunden werden können

1.1.6. Beschreibung der Dynamik und der Struktur

In diesem Kapitel

- zeigen wir, wie das Verhalten und die Struktur von Systemen beschrieben werden kann
- beschreiben wir die Operationen (Methoden) und Attribute (Datenfelder) der Klassen und Objekte

1.1.7. Aufzeigen von Vererbung

Neben der Delegation ist die Vererbung ein zentrales Thema im Rahmen der OO Systementwicklung

- Vererbung
- Generalisierung
- Spezialisierung
- Vererbungsbaum
- Vererbung und Aggregation

1.1.8. Beschreibung des Objektverhaltens

Die innere Struktur der Objekte wird mit Hilfe von Zustanddiagrammen (und Petri-Netzen) beschrieben. Wir beschreiben

- wie Zustandsdiagramme mit Rose beschrieben werden können
- wie Zustände beschrieben werden

- wie Zustandsübergänge (Transitions) beschrieben werden

1.1.9. Überprüfung des Modells

Bevor wir die Architektur fixieren und mit der Implementation beginnen, überprüfen wir das Modell:

- Klassen werden zusammengefasst, eliminiert, aufgeteilt
- die Konsistenz der Modelle wird überprüft
- Szenarios werden durchgespielt

1.1.10. Architektur

Wir fassen die gefundene Architektur zusammen, in Form verschiedener Sichten:

- logische Sicht
- Komponentensicht
- Prozesssicht
- Anwendungsfallsicht
- Distributionssicht

1.1.11. Iterationsplanung und spezielle Themen

In diesem Kapitel fassen wir das Projekt zusammen und überlegen uns, wie wir die Weiterentwicklung sinnvoll planen und umsetzen können.

- Iterationsplanung
- Patterns
- Spezialthemen

Soweit die Gesamtübersicht! Da das Ganze als Wiederholung gedacht ist, werden wir uns zum Teil kurz fassen.

Alle Codebeispiele sind in Java!

1. Definition des richtigen Projektes

Bevor wir ein Projekt starten, müssen wir uns darüber Gedanken machen, ob das Projekt überhaupt Sinn macht!

Ideen zum Projekt erhalten wir von verschiedenen Quellen:
- (möglichen) Kunden
- Messen, Ausstellungen
- neuen Technologien
- Forschung und Entwicklung
- ...

1.2. Hintergrundinformationen

Wir möchten ein einfaches Bibliotheksystem entwickeln. Was in einer Bibliothek geschiet sollten wir alle wissen:
- wir können Bücher ausleihen
- wir können Recherchen in Auftrag geben
- wir können Fachartikel bestellen
- wir können Zeitschriften lesen und ausleihen
- ...

1.3. Risiken des Bibliotheksystems

Für den Benutzer ist es sehr wichtig, *aktuelle* Informationen über die vorhandenen oder ausgeliehenen Bücher und Zeitschriften zu erhalten. Es werden verschiedene Prototypen erstellt, um abzuklären, welche Werkzeuge angepasst sind:
- Datenbanksysteme
- Dateisysteme
- Kommunikationsmechanismen
- Technologien (Applets, HTML, XML, Server Pages, ...)

Das Ergebnis der Abklärungen hat gezeigt, dass das System in Java realisiert werden kann, ohne spezielle Datenbanksysteme oder spezielle Broker Technologien (CORBA, RMI, Jini,...).

Mögliche Risiken sind
- ungenügende Performance
- zu viele Werke (Datenbank wird nötig)
- Anwendung ist zu unstabil
- ...

1.4. Projektbeschreibung

Die Bibliothek verleiht Bücher und Zeitschriften sowie CDs und Videos an eingetragene Benutzer. Die Bücher, Zeitschriften, CDs und Videos sind ebenfalls registriert.

Die Bibliothek kauft auch periodisch neue Bücher oder abonniert neue Zeitschriften; populäre Werke werden mehrfach, in mehreren Kopien angeschafft.

Die Bibliotheksangestellten kooperieren mit den Bibliotheksbenutzer und sollen vom neuen Bibliothekensystem unterstützt werden.

Ein Bibliotheksbenutzer kann ein Buch oder eine Zeitschrift, welche temporär nicht verfügbar sind, reservieren. Die Person, welche die Reservation vornimmt, wird beim Verfügbarwerden des Werkes benachrichtigt. Reservationen können auch gelöscht werden.

Die Bibliothek muss die Informationen (Titel, Bibliotheksbenutzer, Gebühren, Reservationen) einfach eingeben, mutieren und auswerten können.

Das System muss auf unterschiedlichen Betriebsytemen lauffähig sein (Unix, Linux, Windows, OS/2, ...) und mit einer graphischen Oberfläche versehen sein.

Erweiterungen am System müssen möglich sein, ohne dass das ganze System neu entwickelt werden muss.

2. Analyse

Wir fassen die zwei Phasen "Anforderungsanalyse und Analyse der Domäne" zu einer Phase "Analyse" mit zwei Teilphasen (nach ROP) zusammen.

Analysetätigkeiten beschreiben das *WAS*, in beiden Teilphasen. Deswegen können wir sie auch in kleineren Projekten zusammen fassen.

1.1. Projektbeginn

Als erstes müssen wir einen Grob-Projektplan definieren und die Projektorganisation fixieren. Bei Projektbeginn geht es darum alle administrativen Abläufe zu fixieren :

- wer muss informiert werden, wer möchte informiert werden
- welche Ergebnisse erwartet der Steuerungsausschuss
- ...

1.2. Anforderungsanalyse (Requirement Analyse)

1.2.1. Anwender (Actors)

Als erstes müssen wir die Anwendungsfälle beschreiben.

Wir gelangen zur Beschreibung der Anwendungsfälle, indem wir

- die Projektspezifikation durchlesen und
- uns mit den Beteiligten unterhalten
- die Fachliteratur darüber lesen (Standardwerke)
- Tagungen zum Thema besuchen
- bestehende Systeme untersuchen
- ...

Die Beteiligten des Systems:

- Bibliothekar(in)
- Bibliotheksbenutzer
- Bibliotheksangehörige können gelegentlich auch Bücher, Magazine, Videos, CDs ... ausleihen
- der Bibliotheksbenutzer arbeitet nicht direkt mit dem System, er nutzt das System mit Hilfe der Bibliothekare/innen


Rose.lnk

Schauen wir uns mal an, wie Actors in Rose erfasst und beschrieben werden.
Ich setze voraus, dass Sie die Evaluationsversion installiert haben und diese Übungen auch praktisch durchführen!

1. Nach dem Starten von Rose sehen Sie links oben ein Fenster mit den unterschiedlichen Sichten:
   - *Use Case View*
   - *Logical View*
   - *Component View*
   - *Deployment View*
2. Wählen Sie die *Use Case View* und klicken Sie auf die rechte Maustaste
3. Sie sollten ein Menü sehen mit etwa folgenden Einträgen:
   - *Open Specification*
   - *New*
   - *Sort*
   - ...
   - *Refersh*
   - *Allow Docking*
   - *Hide*
4. wählen Sie *New* - Package, Use Case, *Actor* ...
5. das System kreiert einen neuen Actor *New Class*
6. benennen Sie den Actor (zum Beispiel *Bibliothekar*)

1.2.1.1.Dokumentation der Actors

Jeder Actor, jeder Anwender des Systems, sollte möglichst schnell kurz beschrieben werden.

- Bibliothekar(in):
  eine Person, die in der Bibliothek arbeitet und berechtigt ist, Ausleihen und Neuanschaffungen der Bibliothek zu erfassen und zu mutieren,
  sowie für die Betreuung der Bibliotheksbenutzers zuständig ist
- Bibliotheksbenutzer(in):
  eine Person, die Bücher, Zeitschriften, Videos, CDs,... ausleihen möchte, oder Fachartikelkopien bestellen möchte.


Rose.lnk

Schauen wir uns an, wie wir dies in Rose durchführen können:

- wählen Sie den Actor, den Sie beschreiben möchten, an und öffnen Sie die Spezifikation durch Doppelklick.
- Sie sehen eine *Class Specification* mit einem geöffneten Fenster *Documentation*
- in dieses Fenster können Sie nun die Spezifikation des Actors, des Anwenders eintragen (Freitext)

Es ist nun Ihre Aufgabe, die beiden Actors (Bibliothekar und Bibliotheksbenutzer) zu erfassen und das Modell zur Sicherung abzuspeichern.

Die UML Notation eines Actors ist, wie Sie in Rose sehen :


Bibliothekar
(from Use Case View)

1.2.2. Use Cases (Anwendungsfälle)

Unser Bibliothekssystem muss mindestens folgende Anwendungsfälle umfassen, um der Projektbeschreibung gerecht zu werden:

- einen Gegenstand ausleihen (Gegenstand : Buch, Video, CD, Zeitschrift, ...)
- einen Gegenstand zurück geben
- eine Reservation vornehmen
- eine Reservation löschen

- einen neuen Titel erfassen
- einen Titel mutieren oder löschen
- einen Gegenstand erfassen
- einen Gegenstand mutieren oder löschen

- einen Bibliotheksbenutzer erfassen
- einen Bibliotheksbenutzer mutieren oder löschen

Nicht speziell erwähnt haben wir allgemeine Wartungstätigkeiten, Unterhalt der verschiedenen Tabellen und Informationen.

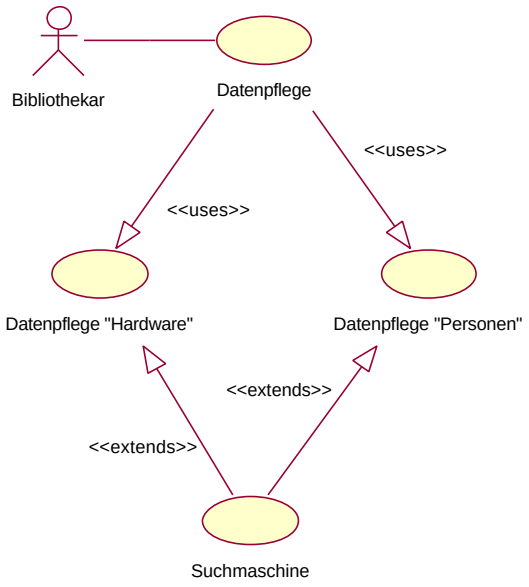
Die Unterscheidung zwischen Titel und Gegenstand ergibt sich daraus, dass verschiedene Titel mehrfach vorhanden sind, also eine Version, eine Kopie des Werkes ausgeliehen wird.

Verfeinerungen der Problemstellung:

1. Ein neuer Titel kann bereits vor dem physischen Vorhandensein des Werkes ins System eingegeben werden. Dadurch sind Reservationen bereits möglich.
2. die Datenpflege wird zusammengefasst:
   es wird ein Use Case *Datenpflege* zusammengefasst.
   Dieser Use Case verwendet die Datenpflege Use Cases (Titel erfassen, Titel mutieren, Gegenstand/Kopie erfassen, Gegenstand/Kopie mutieren, Bibliotheksbenutzer erfassen, Bibliotheksbenutzer mutieren) als "Unteranwendungsfälle" im Sinne von <<*uses*>> in UML.

   In UML zeichnen wir dies wie eine Generalisierung / Vererbung. Bei der <<uses>> Beziehung zeigt der Pfeil (unlogischerweise) auf die Verfeinerung; bei der <<extends>> Beziehung zeigt der Pfeil auf den Anwender des speziellen Use Cases (hier einer Suchmaschine, die von mehreren Use Cases gemeinsam genutzt werden).

   In UML bedeutet dieser Pfeil sonst, dass die Klasse bei der der Pfeil endet, eine Oberklasse (Generalisierung) ist.



Schauen wir uns jetzt an, wie wir die Use Cases in Rose erfassen und Beziehungen der Anwendungsfälle untereinander sowie Beziehungen der Actors untereinander sichtbar gemacht werden können.

1. In der *Use Case View* klicken wir wieder auf die rechte Maustaste und erstellen einen neuen Use Case.
2. Rose erstellt einen Anwendungsfall <<>> *NewUseCase*.
3. Benennen Sie den Use Case (gemäss einem Anwendungsfall), zum Beispiel *Resevierung*
4. mit Hilfe eines Doppelklicks öffnet sich die Spezifikation des Use Cases
5. *Use Case Specification for Reservierung - Documentation*:
   stellt Ihnen eine Freitextfläche zur Erfassung der Beschreibung zur Verfügung.
6. Beschreiben Sie den Use Case möglichst kurz und bündig:
   falls der Bibliotheksbenutzer keine Reservation hat:
   - *ein Titel wird identifiziert*
   - *ein verfügbares Exemplar wird identifiziert*
   - *der Bibliotheksbenutzer wird identifiziert / verifiziert*
   - *die Bibliothek leiht den Gegenstand an den Bibliotheksbenutzer aus*
   - *das Ausleihen wird registier*

   falls der Bibliotheksbenutzer das Werk reserviert hat
   - *der Titel wird identifiziert*
   - *das verfügbare, reservierte Exemplar wird identifiziert (geholt, ...)*
   - *der Bibliotheksbenutzer wird überprüft (ID Karte)*
   - *die Bibliothek leiht das Werk an den Benutzer aus*
   - *die Ausliehe wird registriert und die Reservation gelöscht (siehe Anwendungsfall Reservation löschen)*

Wenn Sie diesen oder einen ähnlichen Text erfasst haben und bestätigt haben (*Apply)*, dann erscheint der Text auf der linken Seite im unteren Fenster.

Dieser Anwendungsfall besteht aus einem primären und einem sekundären Szenario (Reservation nicht vorhanden /vorhanden).

In UML werden Anwendungsfälle mit Hilfe des folgenden Symbols beschreiben.

Mögliche Raster für die Anwendungsfallbeschreibung können Sie im Skript zum Thema "Use Cases" nachlesen.


RationalUnfiedProcess.lnk

Aus der RUP Beschreibung heraus können Sie Templates aufrufen, die Ihnen die Erstellung der Dokumentation erleichtern.
Diese Templates finden Sie im Verzeichnis .. wordtmpl\templates des RUP Verzeichnisses. Sie können diese Dokumente den Rose Symbolen "anhängen":

1. öffnen Sie einen Use Case durch Doppelklicken
2. Sie sehen oben mehrere "Fächer":
   - *General*
   - *Diagrams* (dort können Sie auch Diagramme anhängen)
   - *Relations*
   - *Files*:
     klicken Sie auf die rechte Maustaste und wählen Sie
     Insert File um eine Datei auszuwählen oder
     Insert URL um eine URL auszuwählen
     und dem Use Case "anzuheften"

1.2.3. Beziehungen zwischen Anwendungsfällen

Wir haben die zwei grundlegenden Beziehungen zwischen Anwendungsfällen kennen gelernt:

- uses
  mehrere Anwendungsfälle können diesen Anwendungsfall nutzen
- extends
  optionales Verhalten eines Anwendungsfalles


Rose.lnk

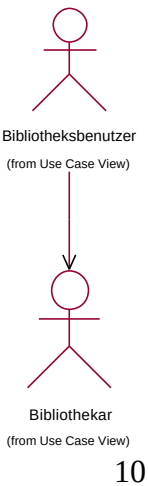
In Rose verwendet man zum Zeichnen beider Fälle das Verallgemeinerungssymbol, vergibt aber je nach Fall (extends, uses) einen andern Stereotyp in der Spezifikation. Zeichnen Sie einen Anwendungsfall, der einen andern erweitern, sowie einen Anwendungsfall der einen andern nutzt.

Sie finden im Skript und der Klausur Beispiele dazu.

Vervollständigen Sie nun die Anwendungsfälle des Bibliotheksystems, wie oben beschrieben.

1.2.4. Vollständiges Use Case Diagramm

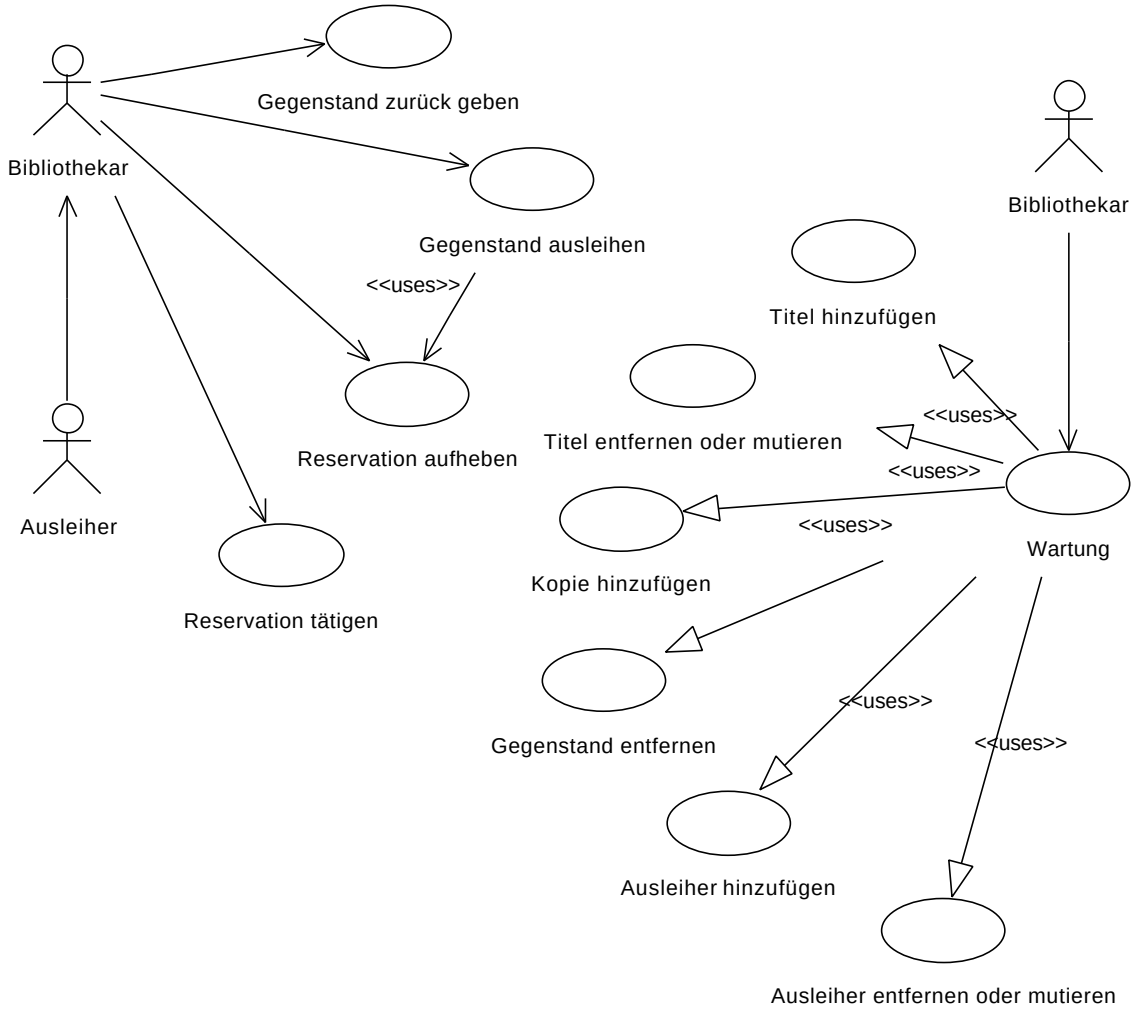
Als Spezialität unseres Systems haben wir noch definiert, dass der Bibliotheksbenutzer nicht direkt mit dem System arbeiten darf, weil wir die Sicherheitsaspekte weglassen wollen und etwas lauffähiges (in Java geschrieben) realisieren möchten.




Rose.lnk

Die Beziehung zwischen dem Bibliothekar und dem Bibliotheksbenutzer ist eine Assoziation; damit diese gerichtet wird, muss in der Spezifikation der Assoziation (doppelklicken auf die Verbindung) im Abschnitt *Role A Detail* bezw. *Role B Detail* das Feld *navigable* selektiert bezw. deselektiert werden. Damit wird die Richtung der Assoziativität festgelegt.

Als Ergebnis erhalten wir folgende Beschreibung unseres Systems:



1.3. Finden der Klassen : Domänenanalyse

Das Finden der Klassen ist eine nicht triviale Tätigkeit. Hinweise auf Kandidaten ergeben sich aus einer Analyse der Substantive (Klassenkandidaten) und der Verben (Methodenkandidaten) sowie aus einem eventuell vorhandenen Datenmodell (Entitäten sind Klassenkandidaten, mit den Standard Datenbankoperationen [insert, delete, select...] als Methodenkandidaten).

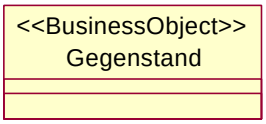
In unserem Falle haben wir keine grosse Auswahl:

- Bibliotheksbenutzer / Ausleiherinformation (als Unterscheidung zum Ausleiher in der Anwendungsfall-Modellierung).
- Titel
  - Buchtitel
  - Magazine Titel
- Gegenstand
- Reservation
- Ausleihe

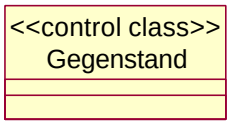
Erfassen von Klassen in Rose
Hier zuerst eine generelle Anleitung:

- Anwählen der Logischen Sicht des Projektes (*Logical View*)
- Öffnen Sie die logische Sicht durch Doppelklick und beschreiben Sie kurz das logische Analyse Modell:
  Die logische Sicht zeigt eine OO basierte Analyse des Bibliotheksystems.
  Alle Klassen und Packages stammen aus der Anwendungsdomäne, sind also nicht optimiert in Richtung Implementation.
  Diagramme umfassen :
  einfache Klassendiagramme, Zustandsdiagramme innerhalb der Klassen, wo überhaupt angebracht.
  Die Aufteilung in die zwei Packages GUI und Business entspricht einem vereinfachten Model-View-Controler Pattern (oder einfach einer 2-Tier Architektur; Client - Server)
- <<>> *NewClass* wird eingefügt und angezeigt (selektiert). Benennen Sie die Klasse (gemäss obiger Kandidatenliste)
- *Gegenstand*
- öffnen Sie die Klasse *Gegenstand* mit einem Doppelkick und spezifizieren Sie diese (Business oder Analyse) Klasse:
  Der Gegenstand stellt eine physikalische Grösse dar, einen Titel oder eine Kopie eines Titels.
  Der Gegenstand hat mehrere Zustände:
  - ausgeliehen
  - nicht ausgeliehen
  Jedem Gegenstand ist ein Titel zugeordnet (Titel, Author, ISBN, ...)
  Die Unterscheidung <Titel,Gegenstand> erlaubt eine bessere Verwaltung eines Werkes mit mehreren Kopien oder die Reservation eines Titels, eines Buches, welches bestellt, erfasst aber noch nicht geliefert wurde.

Die Klasse müsste jetzt in etwa wie folgt aussehen:
den Stereotyp (Business Objekt) haben sie definiert. Es gibt einige vordefinierte Stereotypen, wie Sie aus dem PopUp Menü erkennen können.



Ändern Sie den Stereotyp in der Klassenspezifikation auf "Control Class". Jetzt kann Ihre Klasse entweder so (Label):



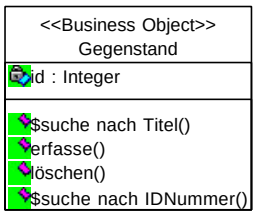
oder so (als Icon):

aussehen. Sie können das Aussehen steuern, indem Sie (rechte Maustaste) die "Option" "Stereotype Display" anwählen.

- erfassen Sie die Methoden und Attribute der Klasse *Gegenstand:*
  Methoden:
  **$***suche nach Titel() ($* soll bedeuten : generische Suche)
  $suche nachID()
  $suche nach Reservation()
  erfasse()
  löschen()
  Attribute:
  id : Integer

Je nach Startauswahl beim Starten von Rose stehen Ihnen die Java Datentypen und Java Klassen alle bereits zur Verfügung; sonst können Sie eigene Datentypen definieren.

Und hier das Ergebnis :
die kleinen Iconen vor den Methoden und Attributen zeigen den Typus an.

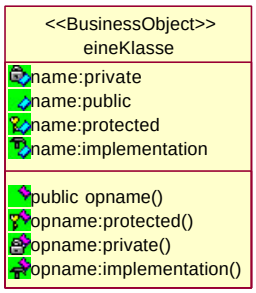


Im obersten Teil wird die Klasse beschrieben, inklusive dem Stereotyp. Dieser kann auch in Form eines Icons ausgedrückt werden.

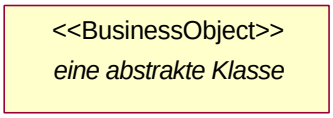
Im zweiten Teil stehen die Attribute, mit *Sichtbarkeitskennzeichen* (**public,private**,...) und Datentyp.

Im dritten Teil finden wir alle Methoden, inkl. Sichtbarkeit.

Die Boxen 2 und 3 (Attribute und Methoden) können auch ausgeblendet werden.



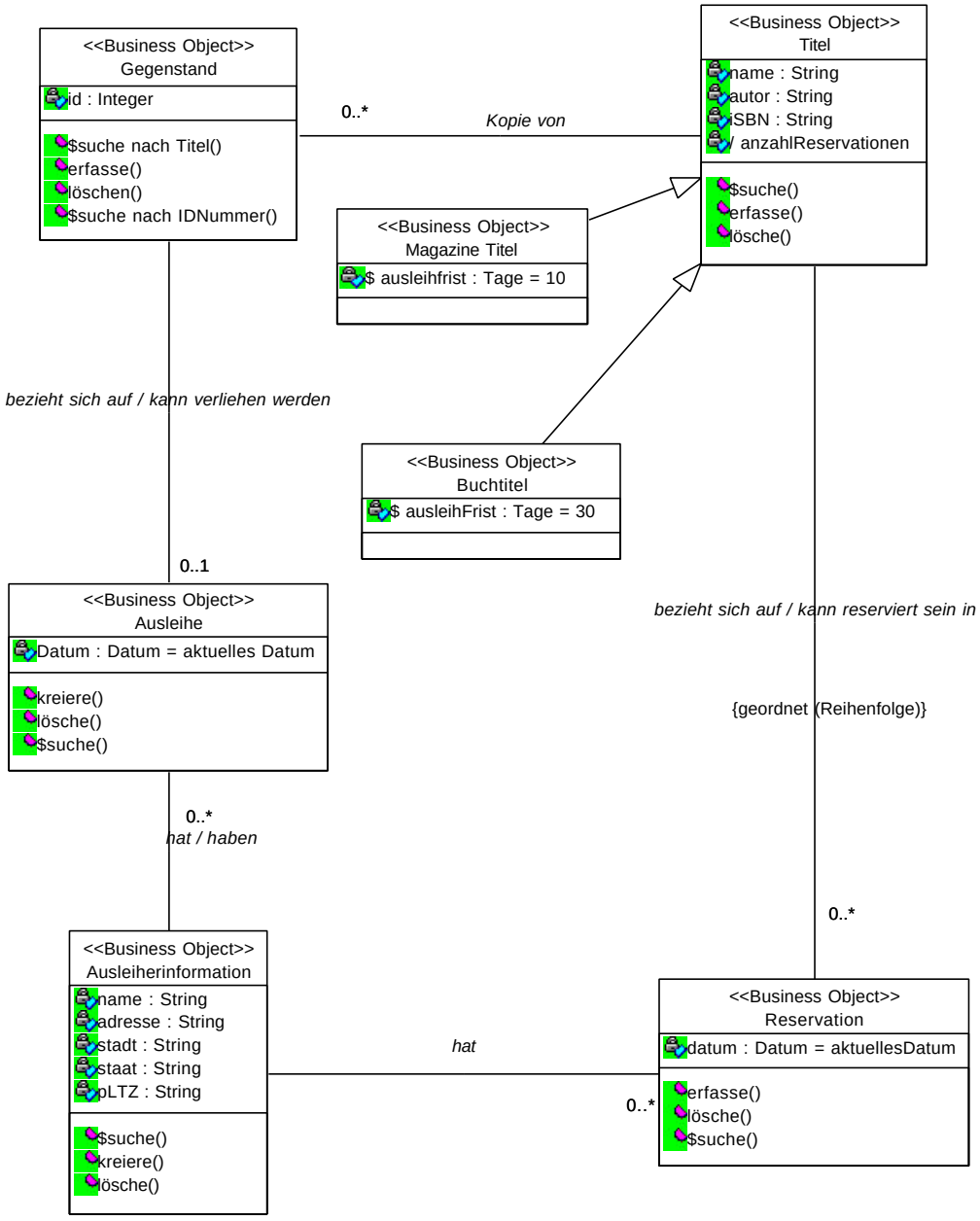
Falls es sich um eine abstrakte Klasse handelt, dann wir der Klassennamen kursiv geschrieben.



Die Unterscheidung in Entity, Boundary und Control Klassen wurde im Skript ausgiebig besprochen. Es handelt sich dabei um spezielle Stereotypen, die in der Praxis häufig auftreten.

Versuchen Sie die andern Klassen zu definieren, inklusive Methoden und Attributen.

Eine mögliche Lösung sieht wie folgt aus:



Wir haben im Wesentlichen ein übergeordnetes Business Objekt definiert, den Titel, der die unterschiedlichen Arten von Werken zusammen fassen soll. Der Einfachheit halber verzichten wir auf CDs, Videos, ...

Gemäss Skript würde man jetzt die Abläufe mit Hilfe von Aktivitätendiagrammen (weniger formell als die Zustandsgraphen) modellieren. Aktivity Diagrams fehlen in Rose zur Zeit noch. Wie im Softwaregeschäft üblich, sind diese für das nächste Release versprochen; ob Hauptrelease oder Wartungsrelease sei dahin gestellt. Die Symbole stehen als Add-In zur Verfügung.

Wie definiert man nun "Business Packages"?

Definition von (Analyse) Packages in Rose

Das Vorgehen lehnt sich an die andern Abläufe an:
wir gehen davon aus, bereits mehrere Klassen definiert zu haben und möchten diese nun probehalber in drei Packages zusammen fassen (wir kommen auf diese Zerlegung zurück und werden eine Aufteilung in zwei Packages definieren und vornehmen):

- Package 1 :
  Gegenstand
  Titel, inkl. Buchtitel und Magazine Titel
- Package 2 :
  Ausleihe
  Reservation
- Package 3 :
  Benutzerklassen (Ausleiherinformationen)

Vorgehen in Rose:

- anwählen der logischen Sicht (*logical View)*
- rechte Maustaste und Auswahl : *new - Package*
- es wird ein Package angelegt : <<>> *NewPackage* wir nennen es "Hardware" (Package1)
- analog verfahren wir zur Definition der weiteren Packages ("Personen", "Transaktionen")
  Die Aufteilung ist wie bereits gesagt wenig sinnvoll, da das Modell noch nicht komplex genug ist, um eine Unterteilung zu rechtfertigen!

Jetzt "versorgen" wir die Klassen in den Packages (drag & drop im Fenster auf der linken Seite).

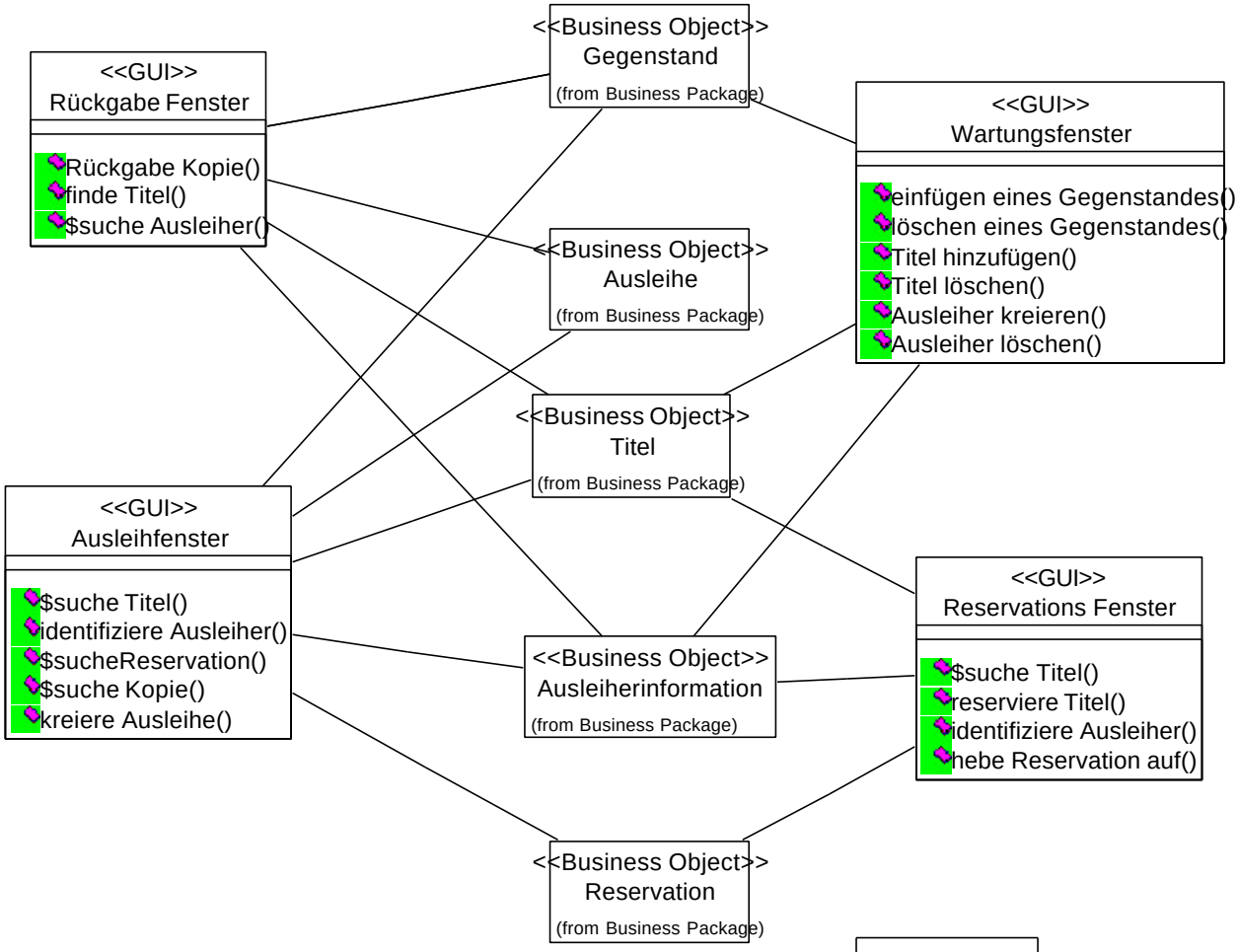
Die Verbindungen der Packages untereinander werden aber nicht automatisch nachgeführt. Dies wäre ja auf Grund der Verbindungen der Klassen untereinander bereits möglich.

Packages werden analog zu den Klassen spezifiziert und dokumentiert (Doppelklick im linken Fenster öffnet das Spezifikationsfenster; Doppelklick auf das Package selber "zoomed" in das Package).

Unsere Applikation ist noch recht "Business Logik" lastig. Was wir benötigen ist etwas mehr "Model-View-Controller" Pattern, also eine graphische Oberfläche. Bisher haben wir uns mit dem "Model" Teil begnügt.

Erweiterung der Bibliotheksapplikation durch eine graphische Oberfläche
Wir wollen, im Sinne eines Prototypen, bereits in der Analyse Phase eine Oberfläche entwickeln, die wir dann im Design implementieren werden.

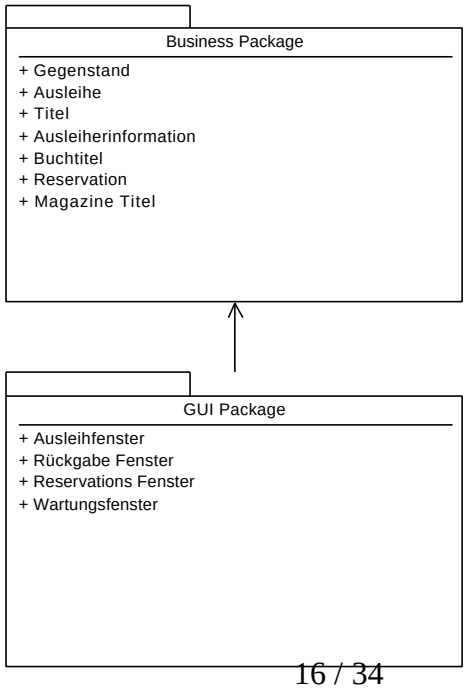
Das Klassendiagramm sieht ähnlich aus, wie jenes für die Business Logik.



Zusammenfassen der Logik und der GUI Klassen in je ein Package
Wir definieren zwei Packlages (unter *Logical View* mit der rechten Maustaste anlegen):

- *GUI Package*
- *Business Package*

Jetzt verschieben wir alle GUI Klassen in das GUI Package; alle andern Klassen in das Business Package.



1.4. Aufzeigen von Objekt-Interaktionen

Dynamik wird auf mehreren Ebenen in unterschiedlicher Detaillierung beschrieben:
auf der Benutzerebene (Analyse) mit Hilfe von

- Aktivitätsdiagrammen (Abläufe, in der Regel innerhalb eines Business Objektes [so funktioniert das Business Objekt])
- Kollaborationsdiagramme (Work Flows, in der Regel zwischen Business Objekten [so kommunizieren die Business Objekte miteinander])

Diesen Darstellungen entsprechen im Design und der Implementierung die Darstellungen:

- Zustandsdiagramme (Prozessbeschreibung einer Klasse)
- Sequenzdiagramme (Wechselwirkungsbeschreibung mehrerer Klassen)

1.4.1. Beschreibung der Dynamik der Anwendungsfälle

Die Dynamik bezieht sich auf konkrete Anwendungsfälle. Daher "heften" wir das Kollaborationsdiagramm bezw. das Sequenzdiagramm dem Anwendungsfall an.

In Rose werden Aktivitäten- und Kollaborationsdiagramme noch nicht sauber unterstützt. Wir verwenden deswegen nur Zustandsdiagramme (für die Klassendynamik) und Sequenzdiagramme (für die Anwendungsfälle).

Die Evaluationsversion ist auch bereits so erweitert worden, dass wir das Kollaborationsdiagramm aus dem in der Regel (als Entwickler, nicht als Benutzer) leichter zu zeichnenden Sequenzdiagramm generieren können!

Betrachten wir zuerst den Anwendungsfall "Gegenstand ausleihen".
Weitere Diagramme lassen sich wie folgt anhängen:

- Auswahl des Anwendungsfalles im Fenster auf der linken Seite (dem "Browser")
- rechte Maustaste anklicken : *new - Sequence Diagram* liefert *NewDiagram*

Legen Sie folgende Diagramme an:

Anwendungsfall	**Sequenzdiagramm**
Gegenstand ausleihen	Gegenstand ausleihen
	reservierten Gegenstand ausleihen
Gegenstand zurückbringen	Gegenstand zurück geben
Reservation tätigen	Titel reservieren
Titel entfernen oder mutieren	Titel entfernen
Titel hinzufügen	Titel hinzufügen
Kopie hinzufügen	Kopie hinzufügen
Gegenstand entfernen	Gegenstand entfernen
Ausleiher hinzufügen	Ausleiher hinzufügen
Ausleiher entfernen oder mutieren	Ausleiher entfernen
Reservation aufheben	Reservation löschen

Jetzt müssen wir die Abläufe noch im einzelnen beschreiben.

1.4.1.1.Gegenstand ausleihen

Beschreiben wir zuerst schematisch den Ablauf. Die Erfassung in Rose wird dann etwas leichter.

Wir wollen einen Gegenstand wie folgt ausleihen:

1. der Bibliothekar sucht einen Titel mit Hilfe des Ausleihfensters, konkret der Methode "$suche Titel"
2. diese ruft die Methode "$suche(String)" der Klasse Titel auf (und zeigt alle verfügbaren Kopien an)

3. der Bibliothekar muss nun eine Kopie auswählen. Er tut dies mit Hilfe der Methode "$suche Kopie" im Ausleihfenster
4. diese Methode ruft die Methode "$sucheNachTitel(Titel)" der Klasse Gegenstand

5. als nächstes muss der Bibliothekar den Ausleiher identifizieren. Er kann dies mit Hilfe der Methode "identifiziere Ausleiher" des Ausleihfensters
6. diese sucht die Ausleiherinformation mit der Methode "$suche(String)"
7. und schliesslich wird ein Eintrag in der Ausleihe kreiert "kreiere Ausleihe(Ausleiherinformation, Gegenstand)".


Rose.lnk

Jetzt erfassen wir diesen Ablauf in Rose. In der Praxis wird man eher umgekehrt vorgehen. Aber wenn der Ablauf unklar ist, dann lässt er sich schlecht modellieren!

- als erstes kopieren wir die relevanten Klassen und Aktoren ind Arbeitsfenster (drag & drop der folgenden Klassen bezw. Anwender):
  Bibliothekar
  Ausleihfenster
  Titel
  Ausleiherinformation
  Ausleihe
  Gegenstand
- **jetzt verbinden wir die Objekte**
  - **Pfeil** vom Bibliothekar zum Ausleihfenster
  - **Doppelklick** auf die Verbindung: dadurch werden alle verfügbaren Methoden (des - Zielobjektes) sichtbar
  - wir **wählen** die passende Methode aus ("$suche Titel() ")

Vervollständigen Sie das Sequenzdiagramm gemäss der obigen Vorgabe!

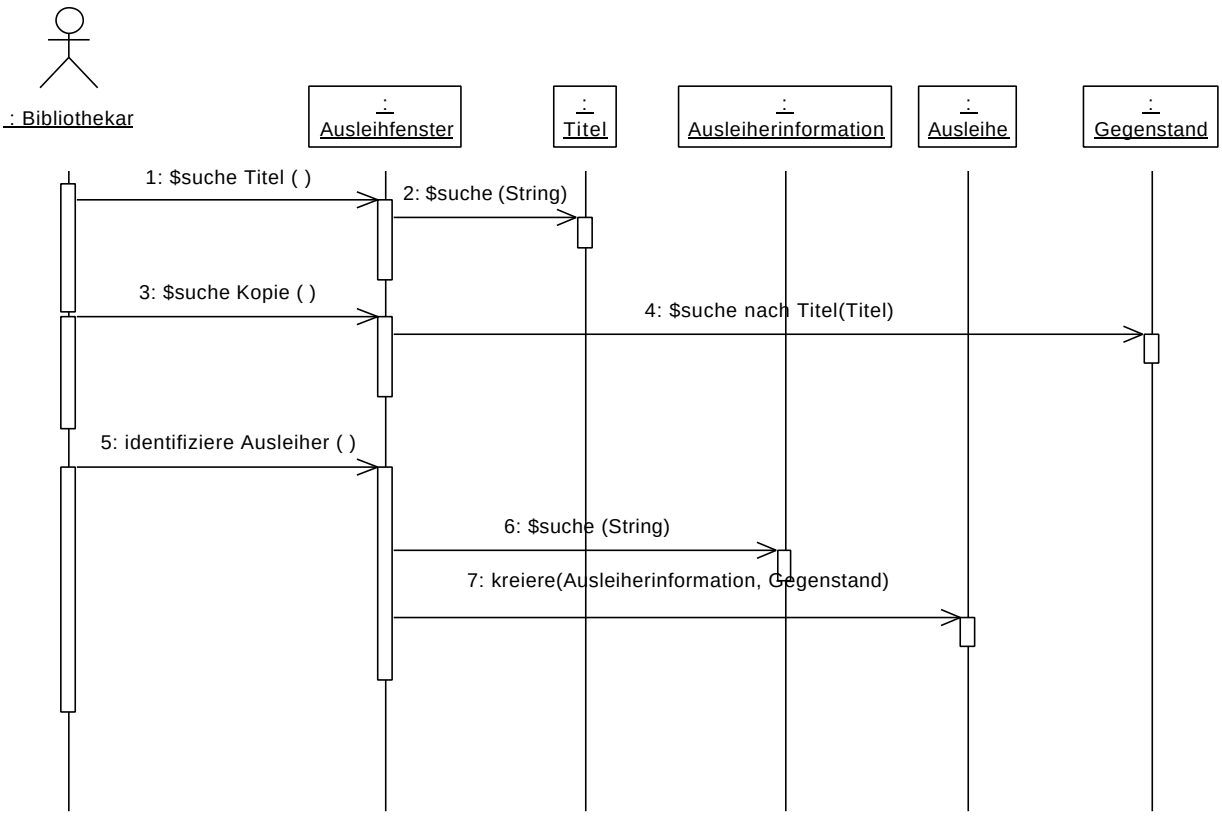
Sie sehen:
falls Sie die Objekte und (Analyse- oder Business-) Klassen sauber definiert haben, ergibt sich der Rest recht einfach und der obige Umweg über eine Textbeschreibung kann eigentlich entfallen.
In der Praxis wird der Benutzer den Ablauf beschreiben und Sie das Sequenzdiagramm oder ein Kollaborationsdiagramm zur Formalisierung verwenden : Methoden zeigen WER was tut.
Der Vollständigkeit halber: wie gelangt man nun zu einem Kollaborationsdiagramm oder eigentlich umgekehrt (in der Analyse startet man, als Benutzer, mit einem Kollaborationsdiagramm; Sie als Entwickler oder Analytiker machen daraus ein Sequenzdiagramm).

Erstellen eines Kollaborationsdiagramms aus einem Sequenzdiagramm
dies geschieht automatisch in Rose

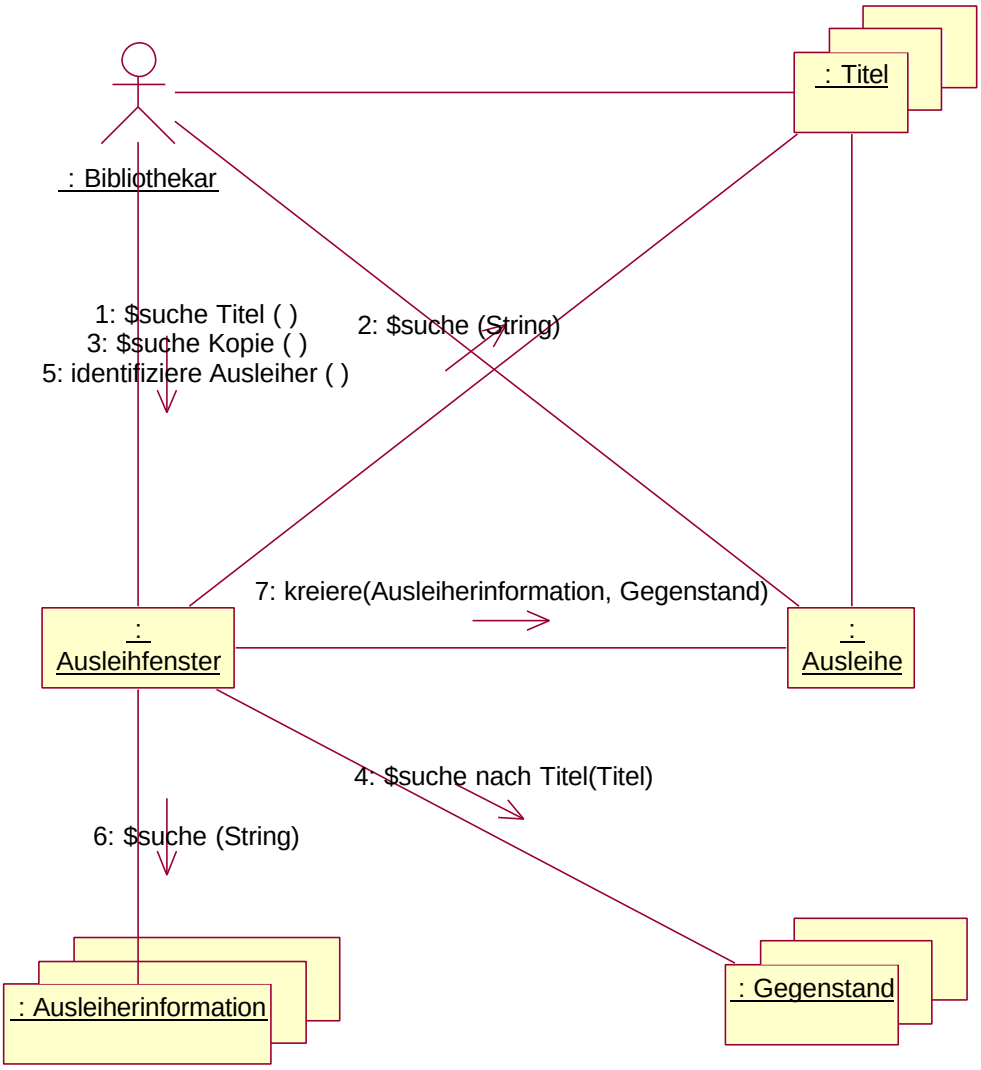
- wählen Sie ein Sequenzdiagramm aus (Doppelklick auf das Icon im Browser Fenster)
- PopUp Menu *Browse - Create Collaboration Diagram F5*
- Rose legt ein Kollaborationsdiagramm an, basierend auf dem Sequenzdiagramm. Analog kann man in der umgekehrten Richtung verfahren!
- Sie müssen nun nur noch das Layout optimieren

Und hier das Ergebnis:

Sequenzdiagramm



Kollaborationsdiagramm



Die weiteren Sequenz- oder Kollaborationsdiagramme können Sie als Übungsaufgabe erstellen. Versuchen Sie auch einmal der Weg "Kollaborationsdiagramm -> Sequenzdiagramm".

Die Musterlösung finden Sie in der Modellbeschreibung
(Rose Datei : "Bibliotheksanalyse.mdl").

1.4.1.1.Warum gibt es unterschiedliche Diagramme?

Wenn Sie die Sequenzdiagramme und die Kollaborationsdiagramme gesehen haben, werden Sie sich sicher fragen, warum man zwei Diagrammtypen definiert hat.

Die Begründung ist folgende:

- **Kollaborationsdiagramme:**
  legen das Hauptaugenmerk auf die Objekte und beschreiben das Zusammenspiel der Objekte auf einer abstrakten, allgemeinen Ebene.
  Ziel:
  der Anwender soll sein gewohntes Umfeld erkennen;

  Eine zu starke Formalisierung ist daher eher unerwünscht!

- **Sequenzdiagramme:**
  beschreiben recht präzise den zeitlichen Ablauf! Dieser ist für den Entwickler wicchtig; dem Anwender ist der Ablauf wahrscheinlich sowieso bekannt.
  Ziel:
  der Entwickler erkennt die zeitliche Abfolge der Methodenaufrufe;

  Eine Formalisierung ist daher erwünscht und nötig!

Analog ist die Unterscheidung zwischen den Aktivitätendiagrammen (Anwendersicht) und den Zustandsdiagrammen (Entwicklersicht). Falls nötig müssen Zustandsdiagramme (Automaten) auch noch in Bezug auf "Concurrency" in Richtung Petri Netze erweitert werden ("kommunizierende" Automaten).

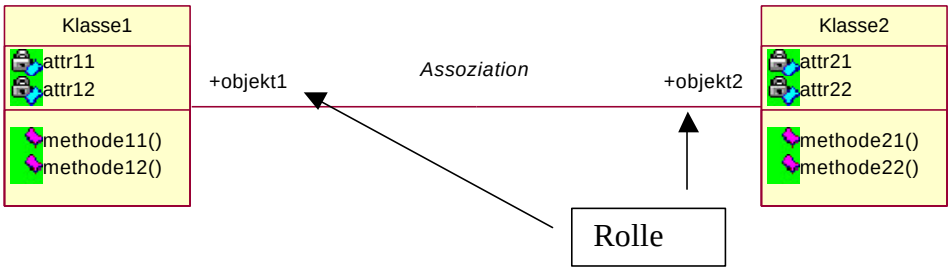
1.5. Festhalten von Beziehungen

Grundsätzlich kennt UML zwei Arten der Beziehung in der Analyse

- *Assoziationen*
- *Aggregationen*

1.5.1. Assoziative Beziehung : Assoziation

Darstellung in UML:



und hier in Java, generiert mit Rose (man sieht's am Code) :

```
// Source file: Klasse1.java

public class Klasse1 {
   private int attr11;
   private int attr12;
   public Klasse2 objekt2;              // Assoziation : Rolle =  objekt2

   Klasse1() {
   }
   /**
     @roseuid 37790ED5006E
    */
   public void methode11() {
   }

   /**
     @roseuid 37790EDC00AA
    */
   public void methode12() {
   }
}
```

und hier folgt Klasse 2:

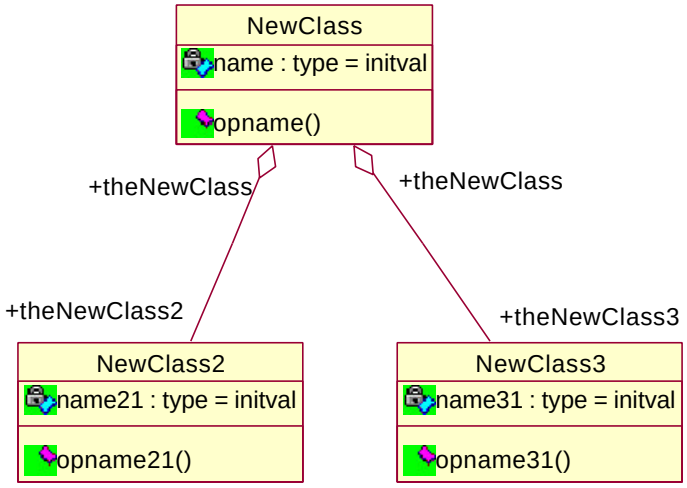
```
// Source file: Klasse2.java

public class Klasse2 {
   private int attr21;
   private int attr22;
   public Klasse1 objekt1;    //Assoziation: Rolle = objekt1

   Klasse2() {
   }
   /**
     @roseuid 37790EFC03C7
    */
   public void methode21() {
   }

   /**
     @roseuid 37790F010266
    */
   public void methode22() {
   }
}
```

Bemerkungen:

- bei einer Assoziation besteht eine Beziehung zwischen zwei Klassen darin, dass die eine Klasse in der einen, die andere Klasse *eventuell* in der andern Klasse instanziert wird (mindestens eine Rolle muss definiert werden können, sonst macht ja die Assoziation keinen Sinn)
- die Rolle muss in Rose spezifiziert werden, sonst wird von Rose automatisch ein Rollennamen definiert und in den Java Code eingefügt
- Rose generiert eine allfällig spezifizierte Mächtigkeit der Beziehung nicht korrekt:
  eine n:1 und eine 1:1 Beziehung generieren den selben Code.
  Im Falle der n:1 Beziehung wäre die Generierung eines Vektors von Objekten besser.

1.5.2. Aggregation

Darstellung in UML:



und in Java, wobei der Code auch hier automatisch generiert wurde

```
// Source file: NewClass.java

public class NewClass {
   private type name = initval;
   public NewClass2 theNewClass2;
   public NewClass3 theNewClass3;

   NewClass() {
   }
   /**
      @roseuid 37791BFF02FE
    */
   public return opname( argname) {
   }
}

// Source file: NewClass2.java

public class NewClass2 {
   private type name21 = initval;
   public NewClass theNewClass;

   NewClass2() {
   }
   /**
      @roseuid 37791BF60229
    */
   public return opname21( argname) {
   }
}
```

```
// Source file: NewClass3.java

public class NewClass3 {
   private type name31 = initval;
   public NewClass theNewClass;

   NewClass3() {
   }
   /**
      @roseuid 37791C080275
    */
   public return opname31( argname) {
   }
}
```

Bemerkungen:

- semantisch, das heisst von der Bedeutung her, bindet die Aggregation stärker als die Assoziation
- mit einer Aggregation will man ausdrücken, dass die Teile, die aggregiert werden, das zusammen das Ganze ausmachen, also eine *Teil-Ganzes Beziehung*, eine Beziehung eines Teiles zum Ganzen.
- in Java sieht eine Aggregation wie eine Assoziation aus.

1.5.3. Assoziation oder Aggregation oder Komposition?

Es ist eher schwierig zu entscheiden, ob eine Beziehung eine Aggregation oder eine normale Assoziation ist.

In Java spielt das eher keine Rolle, noch!

Aber im Modell selber, und darum geht es, verdeutlicht eine Aggregation schon etwas anderes als eine Assoziation.

In UML kennt man zusätzlich eine verschärfte Version der Aggregation, die *Komposition.*

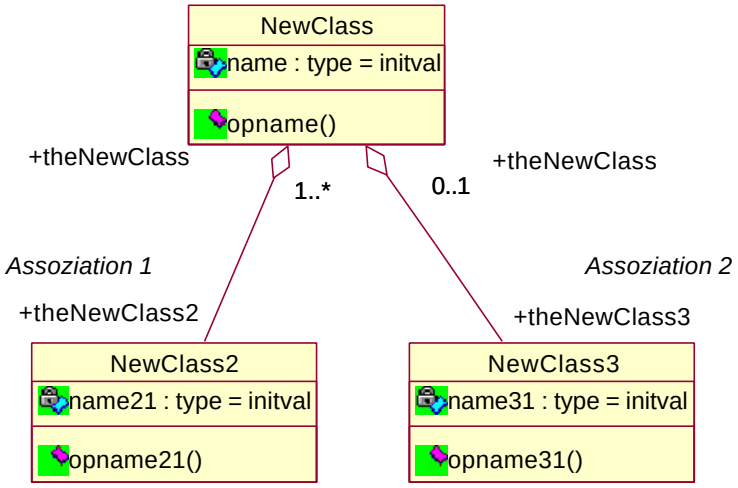
Eine Komposition ist eine Aggregation, bei der die Teile (Aggregation = Teil-Ganzes Beziehung) *nur* in dieser Klasse vorkommen.

Beispiel für eine Komposition:
Herr Frei hat ein Knie; sein Knie gehört ihm und nur ihm. Kein andere will es, kein anderer kriegt es, kein anderer kann es brauchen.

1.5.4. Multiplizitätsindikator

Die Beziehungen können noch mit einer Kardinalität versehen werden, welche anzeigt, wie oft eine Klasse in der andern Klasse auftreten kann. Wir kennen dies von den Datenmodellen.

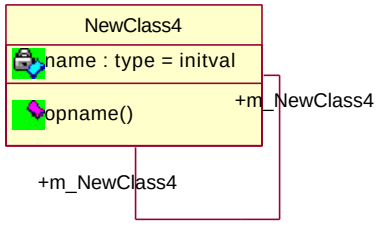
UML Darstellung:



In Java sieht man keinen Unterschied, obschon einer besteht: der generierte Code ist nicht korrekt. Besser wäre die Generierung eines Vektors von Objekten, falls die Beziehung n:1 ist; falls man UML für die Datenmodellierung einsetzt, dann spielt die Mächtigkeit natürlich die entscheidende Rolle.

1.5.5. Reflexive Beziehungen

UML Darstellung:

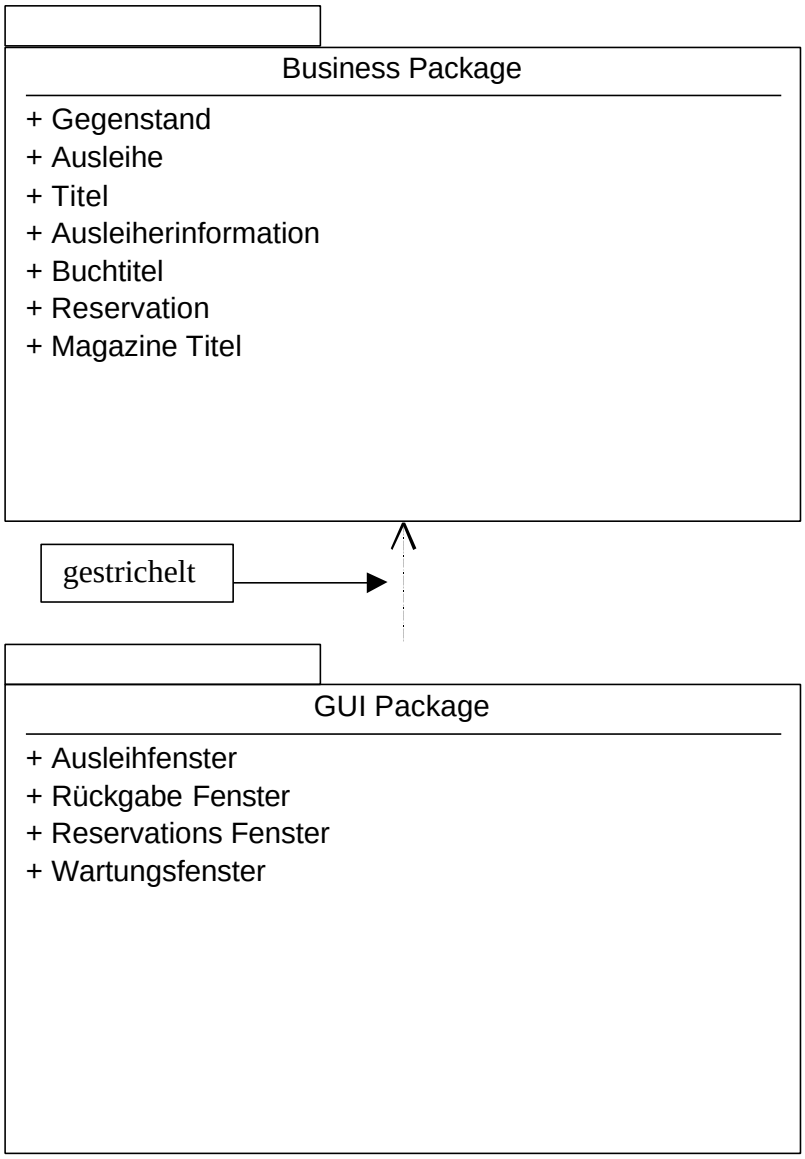


und in Java:

```
// Source file: NewClass4.java

public class NewClass4 {
   private type name = initval;
   public NewClass4 m_NewClass4;
   public NewClass4 m_NewClass4;

   NewClass4() {
   }
   /**
     @roseuid 377923DF01F9
    */
   public return opname( argname) {
   }
}
```

1.5.6. Beziehung zwischen Packages

UML Notation:



Bemerkungen:

- ob eine Beziehung zwischen Paketen besteht oder nicht, kann auf Grund der Beziehungen der Klassen in den Paketen bestimmt werden.
- in Rose verwendet man die *Dependency or instantiates* Beziehung

1.6. Beschreibung der Dynamik und der Struktur

Bei der Beschreibung der Dynamik können wir die Sequenzdiagramme oder die Kollaborationsdiagramme einsetzen.

Dabei ist folgendes Vorgehen angebracht:

- zuerst werden die Klassen beschrieben
- dann werden die "Innereien" der Klassen beschrieben, also die Methoden und die Attribute
- schliesslich zeichnet man die Sequenzdiagramme (am Anwendungsfall angehängt). Dabei geht man wie folgt vor:

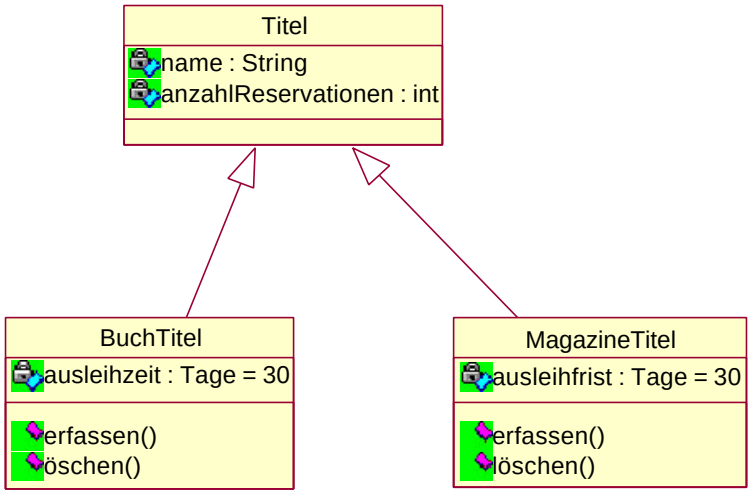
1. Klassen / Objekte mit drag und drop aus dem Browser Fenster auf die Arbeitsfläche "ziehen"; als erstes den Actor.
2. Verbindungen zeichnen und beschriften:
   beim Beschriften muss man auf die rechte Maustaste drücken und erhält eine Liste der verfügbaren Methoden.

1.7. Aufzeichnung von Vererbung

In unserem Beispiel haben wir kaum die Möglichkeit noch viel Vererbungen, Abstraktionen oder sogar mehrfach Vererbungen zu finden.

Die einzige Vererbung, die wir definiert haben, beschreibt die Zusammenfassung von Buchtitel und Magazinen zu einem allgemeinen Titel:

Darstellung in UML:



und in Java:

```
// Source file: Titel.java

public class Titel {
   private String name;
   private int anzahlReservationen;

   Titel() {
   }
}

// Source file: BuchTitel.java

public class BuchTitel extends Titel {
   private Tage ausleihzeit = 30;

   BuchTitel() {
   }
   /**
     @roseuid 37792B7400E1
    */
   public void erfassen() {
   }

   /**
     @roseuid 37792B86001F
    */
   public void öschen() {
   }
}

// Source file: MagazineTitel.java

public class MagazineTitel extends Titel {
   private Tage ausleihfrist = 30;

   MagazineTitel() {
   }
   /**
     @roseuid 37792BB70355
    */
   public void erfassen() {
   }

   /**
     @roseuid 37792BBF03C4
    */
   public void löschen() {
   }
}
```

1.8. Beschreibung des Objektverhaltens

Da in Rose die Aktivitätsdiagramme noch nicht implementiert sind, beschreiben wir die präziseren Zustandagraphen.

Wenn man versucht, in Rose Zustandsdiagramme zu erfassen, versucht man als erstes mit *Browse - State Diagram.*

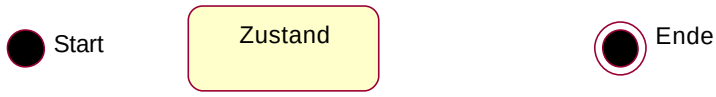
Aber diese Option ist nicht aktiviert!

Was machen wir falsch?

In Rose und UML sind Zustandsgraphen für die Beschreibung der Objekte und Klassen vorgesehen. Wir müssen also erst eine Klasse auswählen; dann wird auch die Menüoption aktiviert.

Vorgehen in Rose:

- im Browser Fenster eine Klasse anklicken
- rechte Maustaste drücken : *Open State Diagram*
  Jetzt wird ein neues Zustandagraphen Menü sichtbar.
- jeder Zustand hat einen Startzustand, einen Endzustand und normale Zustände:



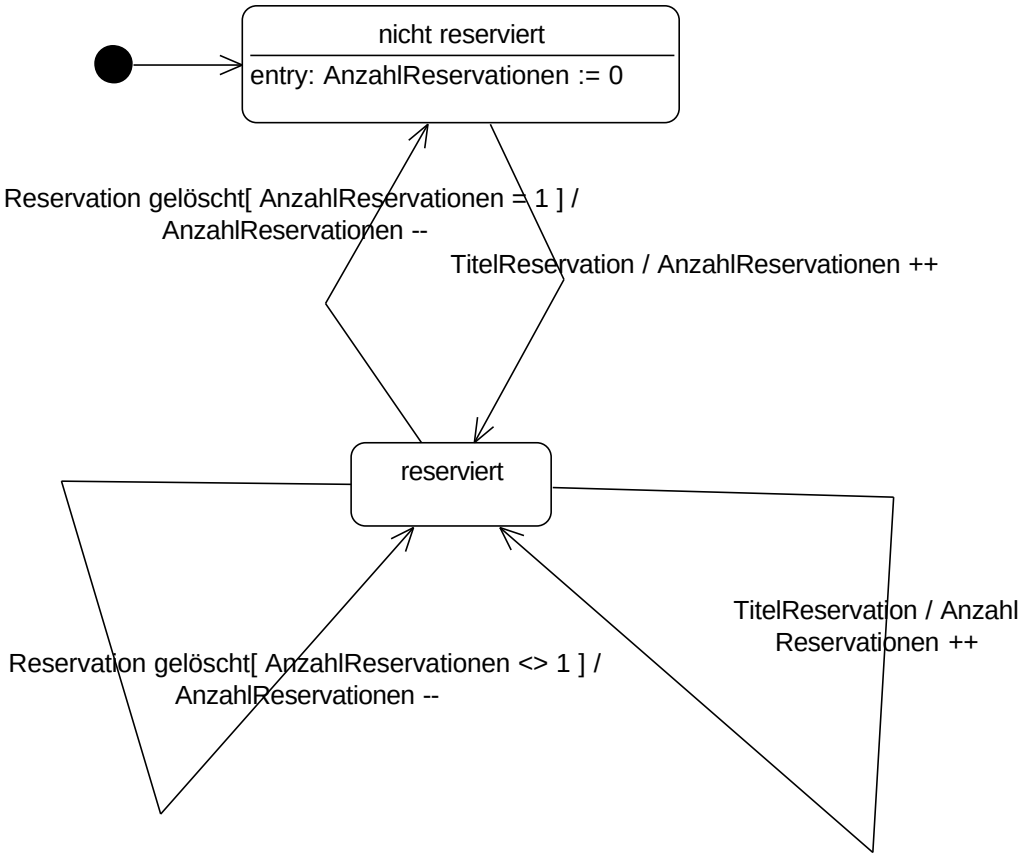
Versuchen Sie folgende Prozesse zu modellieren:
in der Klasse *Gegenstand* soll ein Zustandsgraph hinzugefügt werden, der folgenden Übergang beschreibt:

- Zustand : Start
- Zustand : *nicht ausgeliehen*
- Zustand : *ausgeliehen*
- Zustand : Endzustand

Der Übergang aus dem Zustand *nicht ausgeliehen* in den Zustand *ausgeliehen* erfolgt immer dann, wenn ein Gegenstand ausgeliehen wird. In der andern Richtung erfolgt der Übergang immer dann, wenn ein Gegenstand zurück gegeben wird.

Zudem besteht die Möglichkeit aus den beiden Zuständen *ausgeliehen, nicht ausgeliehen* jeweils in den Endzustand zu wechseln.

Zur Klasse Titel beschreiben wir die Dynamik mit folgendem Zustandsgraphen:



Wie lässt sich dieser Zustandsgraph interpretieren?

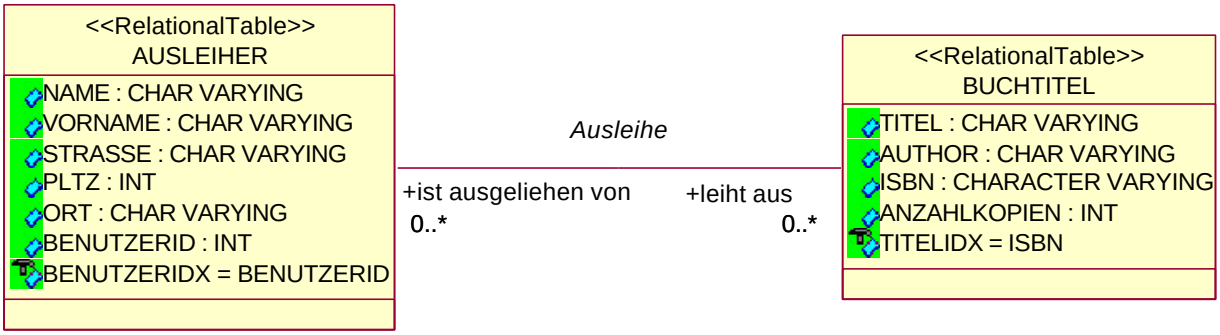
1.9. Aufgaben (Analyse)

Starten Sie Rose und wählen Sie als Basismodell ORACLE oder ein anderes Datenbanksystem, also *nicht* Java.

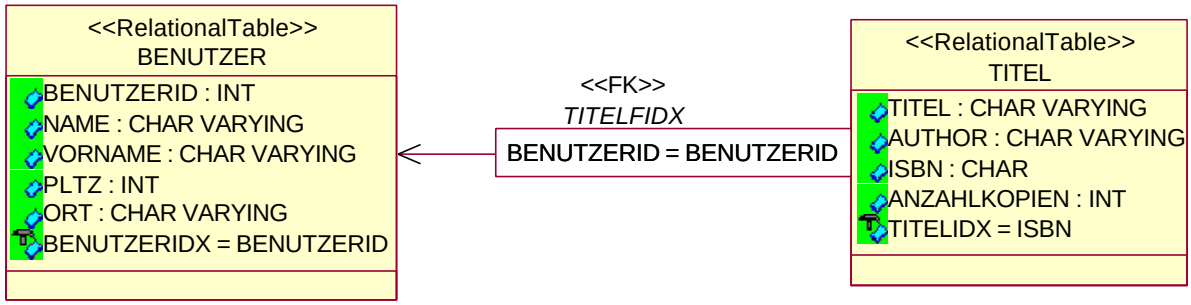
Prüfen Sie jetzt die Implikationen der Klassenverbindungen (Assoziation, Aggregation, Vererbung) auf der Ebene von Tabellen, an Stelle von Objekten und Klassen.

Hinweis:
Damit Sie DDL (Data Definition Language : konkret SQL) generieren können, müssen Sie die Klassen mit einem Wizzard (zum Beispiel im Menü unter *Tools - Oracle8 - Data Types Creation Wizzard*) kreieren, sonst können Sie anschliessend kein SQL generieren!

UML Darstellung:



oder mit Fremdschlüssel (Foreign Key)



DDL (SQL) für Oracle:

```
CREATE TABLE BENUTZERSCHEMA.BENUTZER (
    BENUTZERID INT NOT NULL UNIQUE,
    NAME CHAR VARYING(25) NOT NULL,
    VORNAME CHAR VARYING(25) NOT NULL,
    PLTZ INT NOT NULL UNIQUE,
    ORT CHAR VARYING(25) NOT NULL,
    CONSTRAINT BENUTZERIDX PRIMARY KEY (BENUTZERID));

CREATE TABLE BENUTZERSCHEMA.TITEL (
    TITEL CHAR VARYING(25) NOT NULL,
    AUTHOR CHAR VARYING(25) NOT NULL,
    ISBN CHAR(12) NOT NULL UNIQUE,
    ANZAHLKOPIEN INT,
    BENUTZERID INT,
    CONSTRAINT TITELIDX PRIMARY KEY (ISBN),
    CONSTRAINT TITELFIDX FOREIGN KEY(BENUTZERID) REFERENCES
BENUTZERSCHEMA.BENUTZER(BENUTZERID));
```

Bemerkungen:

- die Tatsache, dass es sich bei der Beziehung <Titel,Benutzer> um eine <n,m> Beziehung handelt, erkennt man auf der Ebene DDL nicht; erst die Inhalte der Tabelle, der Kolonnen, zeigen dies
- die (unidirektionale) Beziehung (Assoziation) zwischen Benutzer und Titel erkennt man an der Konstruktion der Schlüssel bezw. Fremdschlüssel