# Einführung Owl / Owlet

Die OwlSuite ist ein Büchereisystem für die Verwaltung von Büchern, Magazinen und anderen Medien. Die OwlSuite bietet folgenden Funktionsumfang:

* Benutzer- und Rollensystem
  + Anbindung an ein LDAP
* Suchen von Medien über komplex kombinierbare Suchkriterien
* Ausleihen von Medien
* Reservieren von Medien
* Zurückgeben von Medien
* Verlängern von Medien
* Verwalten der Konfiguration
* Direkte Benachrichtigung über Events

Die OwlSuite besteht aus zwei Teilanwendungen

## Owl

Owl ist das Serversystem der Bibiliothekssoftware. Über eine Reihe von Schnittstellen APIs kann mit dem Service kommuniziert werden.

* RMI
* Corba
* EJB
* SOAP

## Owlet

Owlet ist eine grafische Benutzeroberfläche für die Verwendung des Owl Services. Er bietet die Möglichkeit alle Funktionen des Owl Services zu nutzen und bietet damit den Anwendern des Systems alle Möglichkeiten von Owl an.

# Timebox 1

## Datenmodell

Das Datenmodell von Owl wurde sehr komplex gestaltet. Der Ansatz dabei war es ein möglichst umfangreiches und erweiterbares Bibliothekssystem zu erstellen was mit diesem Modell eindeutig gelungen ist. Das Modell zeichnet sich dadurch aus dass es sehr dynamisch ist was das Erstellen von neuen Inhalten im Inventar oder das erweitern der möglichen Aktionen im System betrifft.

Zudem wählt zudem einen sehr stark objektorientierten Ansatz der Darstellung der Daten.

Die in Abbildung 1 gezeigte Version des Datenmodells zeigt war eine sehr umfangreich ausgearbeitet Variante welche so nicht komplett implementiert wurde. Sie zeigt Sie das finale Datenmodell erweitert werde kann um ein vollständiges und real einsetzbares Modell auf Basis von Owl aussehen könnte.

Das Modell gliedert sich in vier Teilbereiche.

### Personen und Usersystem

Dieses System bildet jene Teilbereiche ab welche für die Authentisierung im System benötigt werden. Die Klasse Person dient als Basisinformation für Systembenutzer und Autoren. Ein Systembenutzer hat die Möglichkeit sich am Serversystem Owl einzuloggen und bekommt über die Zuweisung von Rollen die Rechte Aktionen auf dem Server auszuführen.

Ein Systembenutzer hat zudem eine Historie seiner Status und eine Liste von durchgeführten Transaktionen. Der Status legt dabei fest ob der Benutzer aktiv ist oder aufgrund einer Ursache gesperrt. Über Transaktionen können Zahlungen des Benutzers archiviert werden. Bspw. Zahlungen für bspw. Mitgliedsbeiträge.

### Medien-Management

Wie der Name der Sektion vermuten lässt handelt es sich dabei um die Verwaltung der im Inventar verfügbaren Medien. Über eine komplexe Vererbungsstruktur können verschiedenste Medientypen wie Bücher, Magazine oder CDs eingepflegt werden.

Über ein Tagging System können zu einem Medium Schlüsselworte definiert werden eine Kategorisierung der Medien ermöglicht.

### Exemplar-Management

Im Medienmanagement können lediglich die Medien selbst angelegt werden. Welche oder wie viele Exemplare von diesem Medium nun zur Verfügung stehen finden sich im Exemplar Management. Im System können mehrere Exemplare zu einem Medium angelegt werden welche wiederum mit zusätzlichen Meta-Informationen wie etwa den Zustand des Mediums bestückt werden können.

Ein Medium hat einen Status der festlegt ob das Exemplar gerade ausgeliehen werden kann oder nicht. Alle Status des Exemplars werden in einer Historie abgelegt.

### Ausleih-System

Das Ausleih-System stellt eine Beziehung zwischen dem Exemplar-Management und dem Personensystem her. Über verschiedene Aktivitäten können Exemplaren interagieren. Eine Aktivität speichert sich die Historie über den Verlauf der Aktivität. Zu möglichen Aktivitäten zählen: Reservierungen, Ausleihen mit Verlängerungen oder Downloads.

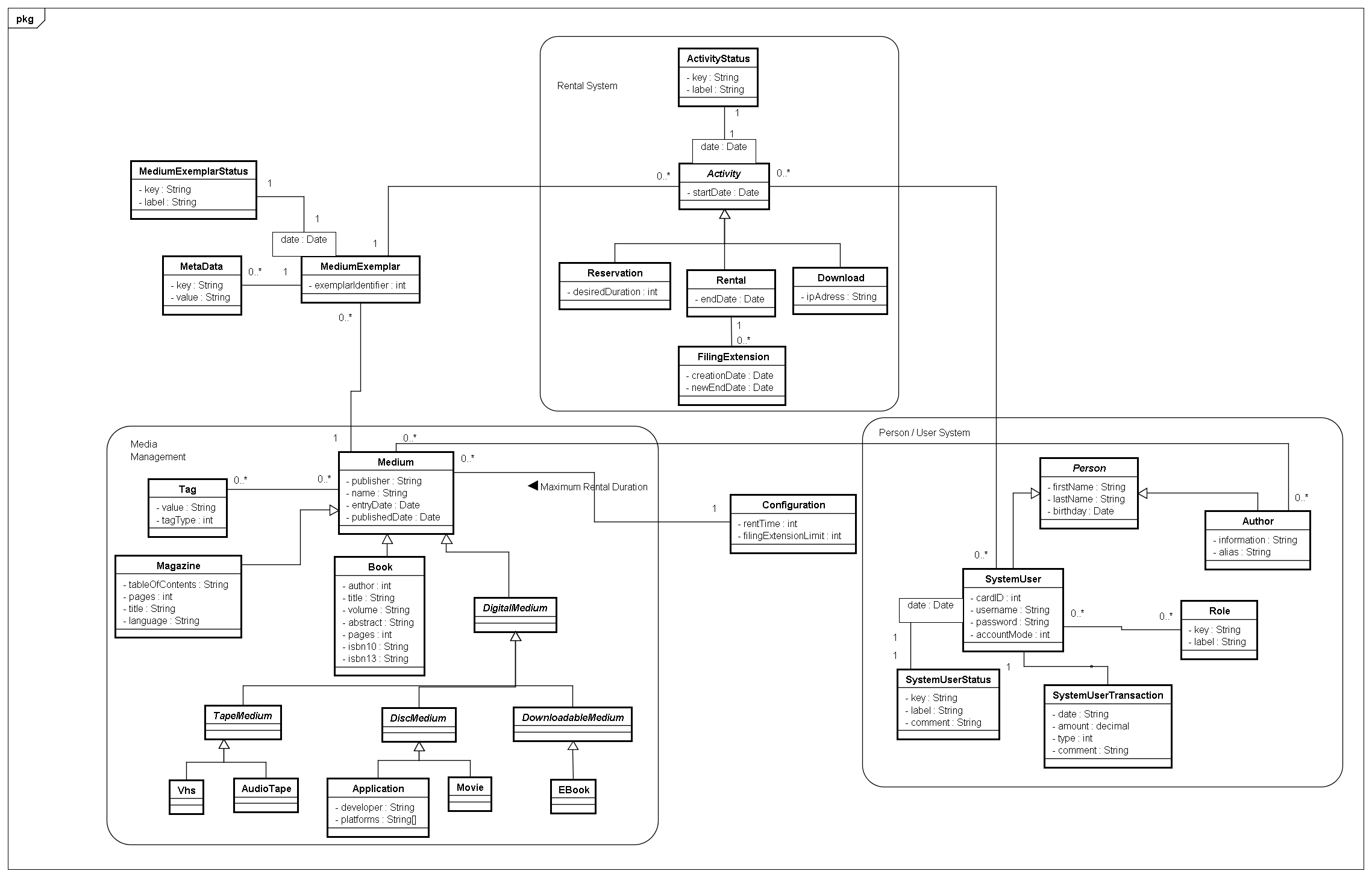


Abbildung 1 Erste Version des Datenmodells

Wie bereits erwähnt ist diese Variante des Datenmodells sehr umfangreich und hätte den Rahmen dieser Semesterarbeit gesprengt. Deshalb wurde eine vereinfachte Variante des Datenmodells angefertigt.

### Personen und Usersystem

Das Usersystem verzichtet auf eigene Klassen wie Person oder Autor. Der Systembenutzer übernimmt dabei die Datenkomponenten der Person. Der Autor wird als Text direkt im Medientyp Buch gespeichert. Das Modell löst zudem attribuierte Relationen auf und führt dafür eigene Klassen ein.

### Medien-Management / Exemplar-Management

Das Medien- und Exemplar-Management wurden zusammengeführt und die Metadatenfunktion für Exemplare wurde entfernt. Zudem wurde die Vererbungshierarchie der einzelnen Medientypen vereinfacht da diese einfach bei Bedarf erweitert werden kann und zu Entwicklungszwecken weniger Medientypen ausreichend sind.

### Ausleih-System

Im Ausleihsystem selbst wurden kaum Änderungen vorgenommen. Die Aktivität für einen *Download* wurde entfernt da kein Medientyp für diesen Aktivitätstyp benötigt wird. Eine weitere Änderung ist, dass nun auch Medien Aktivitäten speichern. Dadurch können Reservierungen auf Medienebene durchgeführt werden.

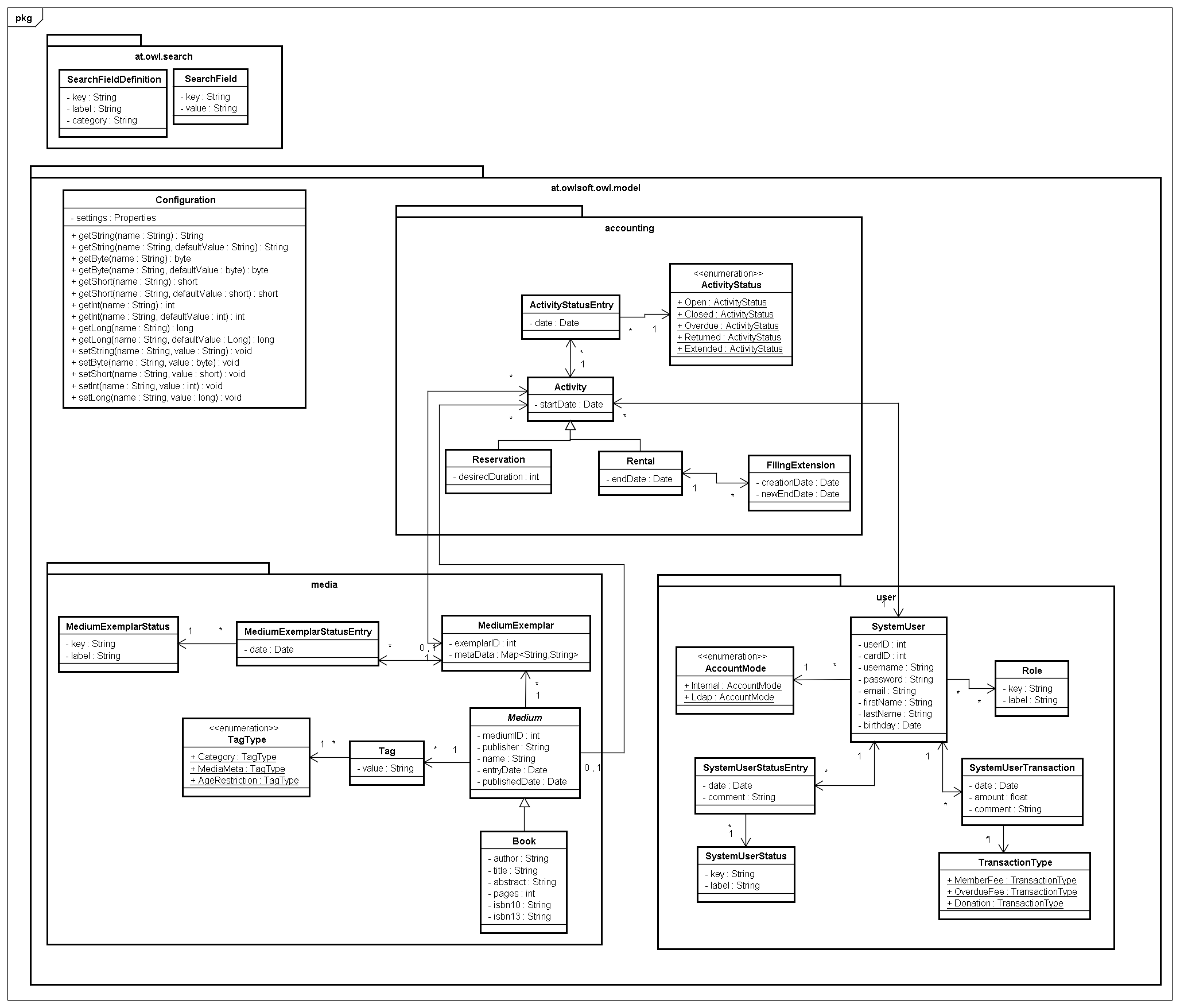
[](Classmodel.png)

Abbildung 2 Zweite Version des Datenmodells

## [C:\Users\Manuel\Documents\uni\semester 5\workspace\OwlSoft\model\Architektur.png](Architektur.png)Architektur

In der Architektur wurde der Fokus auf eine streng getrennte Schichtenarchitektur gelegt. Das Serversystem setzt auf eine 5 Schichten Architektur:

### Kommunikation

Die oberste Schicht ist essentiell für die Interaktion mit der Anwendung. Über verschiedene Services können Client-Anwendungen Aktionen auf dem Owl Service auslösen. Diese Schicht ist komplett austauschbar und alle Services können Autonom arbeiten.

Implementierte Service-Systeme sind

* Remote Method Invokation (RMI)
* Corba
* Java Messaging Service (JMS)
* Enterprise Java Beans (EJB)
* Simple Object Access Protokoll (SOAP)

### Anwendungslogik

Diese Schicht implementiert die Anwendungsfälle von Owl wie etwa das Suchen von Medien oder das Ausleihen von Exemplaren. Diese Schicht stellt mit dem OwlApplicationContext eine Kapselung aller verfügbaren Controller an. Dieser Context repräsentiert eine einzelne Sitzung auf dem System und kann von höheren Schichten dazu verwendet werden um den Zugriff auf die Anwendungsfälle zu ermöglichen.

### Businesslogik

Die Business-Schicht implementiert Anwendungsfallübergreifende Logik. So wird auf dieser Schicht bspw. Die Benutzerauthentifizierung und –Autorisierung durchgeführt.

### Datenabstraktion

Die Datenabstraktionsschicht ist für höhere Schichten stark abstrahiert und kann vollständig ausgetauscht werden. Neben dem Datenmodell selbst wird hier auch der Zugriff auf die Datenbank über Data Access Objects abstrahiert.

## Objektorientierte Datenbank

Eine zentrale Entscheidung für die Entwicklung des Systems war die Wahl des Datenbank Management Systems. Owl setzt dabei auf die objektorientierte Datenbank DB4O[[1]](#footnote-1)

### Vorteile einer objektorientierten Datenbank

Ein objektrelationales Mapping entfällt komplett da direkt die Objekte serialisiert werden. Dadurch können komplexe Objektstrukturen und –Modelle gespeichert werden.

DB4O unterstützt neben einem Datenbank-Server System auch eine Dateidatenbank. Für die Entwicklung eines solchen Projektes wie Owl und die gemeinsame Arbeit an jenem, erleichtert dies die Entwicklung enorm.

### Nachteile einer objektorientierten Datenbank

Für die richtige Verwendung von objektorientierten Datenbanken sind noch wenig „Best-Practices“ definiert.

Bei relationalen Datenbanken ist es allgemein üblich einzelne Datenzeilen mit eindeutigen Identifikationsnummern (IDs) zu versehen. Bei objektorientierten Datenbanken sollte diese Aufgabe unnötig sein, doch wird dadurch die Handhabung der Objekte enorm erschwert.

Die Datenbank speichert sich verschiedene Objektversionen was eine inkonsistente Datenbank zur Folge hat.

Testdaten können nur umständlich über Code-Stücke erzeugt werden und können nicht in üblicher Tabellenform festgelegt werden.

Es steht keine standardisierte SQL für die Abfrage von Daten zur Verfügung.

Suche von Datensätzen problematisch (Kein SQL, Kriterien)

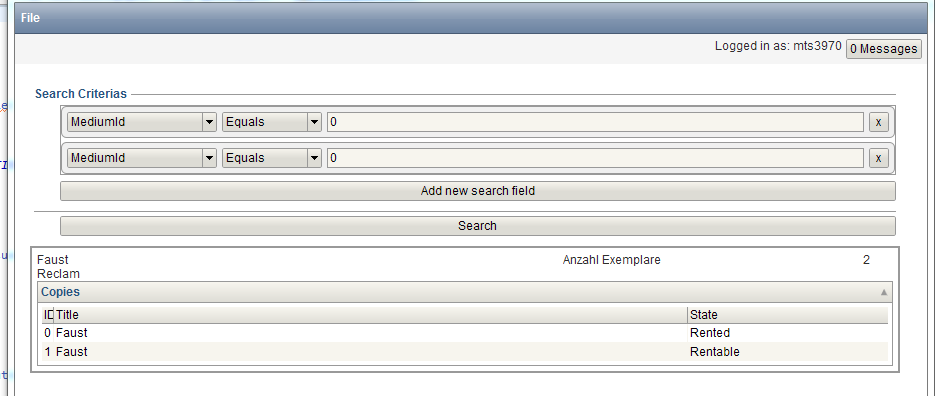
# Timebox 2 - RMI

RMI ist eine Technik, um über das Netzwerk auf Java-Objekte zuzugreifen. In dieser Timebox bestand die Aufgabe darin, für das bestehende System einen Communication Layer für RMI zu entwickeln, sowie einen Client dafür. Spezifisch sollten die Use Cases Suchen, Ausleihen, Verlängern, Reservieren, sowie Rückgabe realisiert werden.

Architektur: In dem Kommunikation-Layer für RMI sind die benötigten Use Case Controller implementiert. Diese basieren auf einer ApiBase Klasse, welche UnicastRemoteObjekt implementiert. Dadurch ist es möglich, auf diese Controller von entfernten Systemen aus zuzugreifen.

### Use Case Suchen:

Für den UseCase Suchen gibt es eine Eingabemaske im Client. Im Bereich Search Criteria[1] lassen sich Such-Kriterien wie die ID des gewünschten Mediums, den Titel, Publisher, etc. auswählen. Durch den Button Add new search field[2] lassen sich bei Bedarf weitere Kriterien hinzufügen. Bei einem Klick auf Search[3] werden die gefundenen Ergebnisse (falls vorhanden) angezeigt, sowie deren Exemplare, und die Verfügbarkeit der Exemplare.



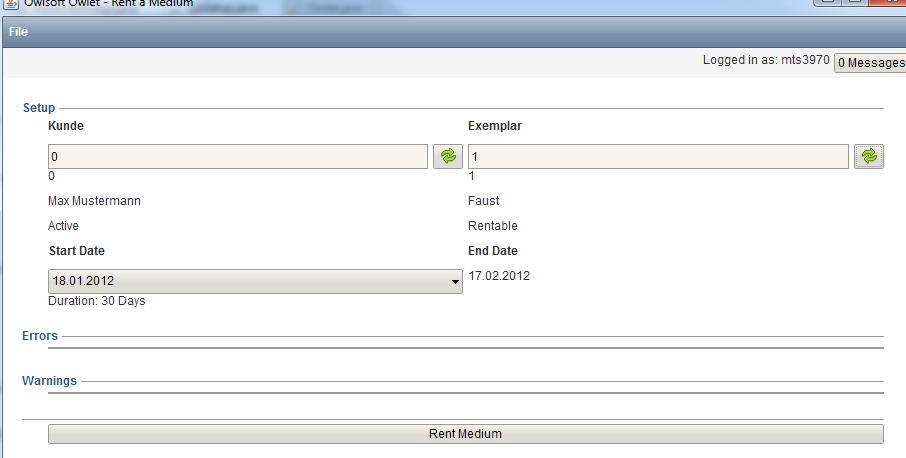
[3]

[2]

[1]

### Use Case Ausleihen

Zum Ausleihen gibt es die Eingabemaske Rental. Nach Eingabe einer Kunden-ID[1] werden die Daten des Kunden angezeigt, sowie nach Eingabe einer Exemplar-ID[2] die Daten des Exemplars. Sollten beide Datensätze in Ordnung sein (Benutzer darf ausleihen, Exemplar ist Rentable) lässt sich das Exemplar mittels des Buttons Rent Medium ausleihen.



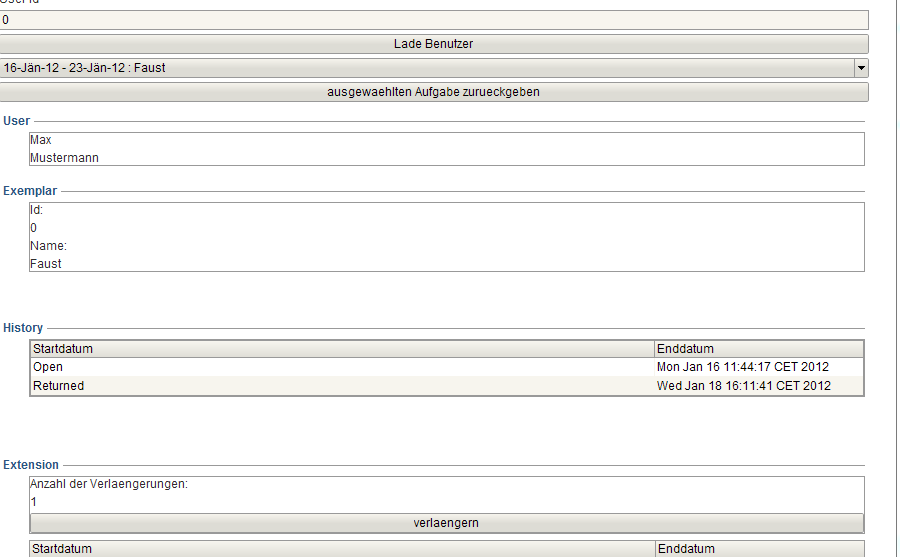
[3]

[1]

[2]

### Use Cases Verlängern/Zurückgeben

Für diese Use Cases gibt es eine gemeinsame Eingabemaske. Unter dem Menüpunkt Show A Rental ist diese erreichbar. Nach Auswahl des Benutzers[1] und der entsprechenden Leihe[2] werden weitere Daten dazu angezeigt. Das Exemplar kann nun zurückgegeben[3] oder verlängert[4] werden.



[4]

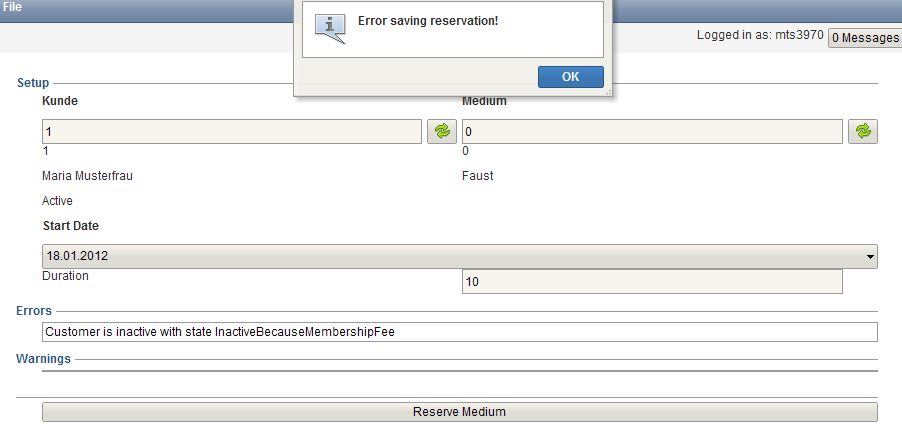
[3]

[2]

[1]

### Use Case Reservieren

Die Eingabemaske zum Reservieren ist identisch mit der Eingabemaske zum Ausleihen. Nach dem spezifizieren der Kunden-ID[1] sowie der Medium-ID[2] wird ein Button zum Reservieren[3] angezeigt. In dem unteren Bild wird eine Fehlermeldung[4] demonstriert, da der Benutzer 1 inaktiv ist und somit weder ausleihen noch reservieren kann.



[4]

[4]

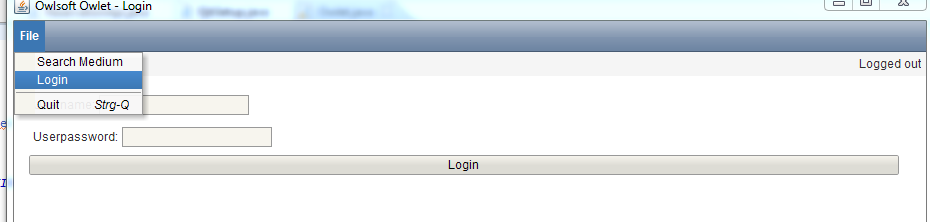
[3]

[2]

[1]

# Timebox 3 LDAP

Nach dem Einloggen in der Owlet Maske überprüft der Server ob der User in der Datenbank existiert. Falls dies nicht der Fall ist wird auf die LDAP Authentifizierung zurückgegriffen.



Da keine Timebox eine Userverwaltung vorgesehen hat, bekommen alle User alle Rollen zugewiesen. Um die JMS Timebox zu testen bekommen User mit einem d oder t im Namen die Operator Rolle zugewiesen.

Bei dieser Timebox traten speziell beim Testen ein Probleme auf. Bei mehrfachen Anfragen an das LDAP wurde die Verbindung abgelehnt und so konnte sich der User nicht anmelden.

# Timebox 4 Corba

Diese Timebox war sehr Aufwendig, da wir eine zu Komplexe Klassenstruktur für Corba haben. Dadurch mussten wir sehr viele IDL Files erstellt werden. Die automatisch generierten Files konnten nicht verwendet werden aufgrund unserer Klassenstruktur.

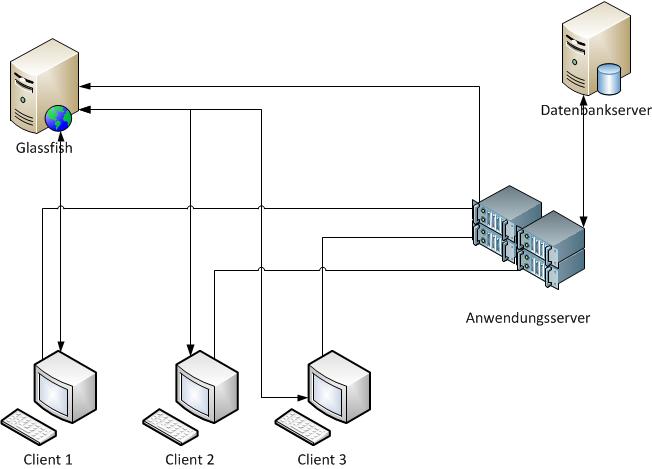
Der Client wurde während der Timebox als Unit Test entwickelt. Für die Abgabe steht aber eine Konsolenapplikation zur Verfügung.

Während der Entwicklung traten viele Fehler auf, weil unsere Interfaces nicht übermittelt werden konnten, um dieses Problem zu umgehen wurden eigene Corba Wrapper entwickelt.

# Timebox 5 Messaging

Die Abbildung zeigt die Abhängigkeiten von der Messaging API. Wenn eine Message auftritt wird diese vom Anwendungsserver an das Topic vom Webcontainer in diesem Fall der Glassfish geschickt. Der Anwendungsserver speichert die detaillierten Informationen.

Die Clients erhalten von dem Webservice eine einfache Textnachricht und wissen so, dass auf dem Server etwas passiert ist. Nach Erhalt dieser Nachricht laden die Clients die in der Datenbank gespeicherten Objekte über den Anwendungsserver



# Timebox 6 EJB

EJB wurde auf den vorhandenen RMI Client aufgebaut. Es konnten die EJB spezifischen Tags gesetzt werden und die Grundlegende Kommunikation funktionierte. Serverseitig hatten wir das Problem, dass intern RMI verwendet wurde, dadurch verlief die Handhabung der Klassen falsch. Dies wurde durch die Local Annotation gelöst, nach setzen dieser Annotation verwendete der Server die richtigen Objekte und nicht die RMI Kommunikation.

Ein ungelöstes Problem ist immer noch. Wenn der Server eine Exception an den Client übermittelt beendet der Server den Socket und somit kann der Client nicht mehr mit dem Server kommunizieren.

1. <http://www.db4o.com/> [↑](#footnote-ref-1)