Universität Bremen
FB 3 – Informatik
Prof. Dr. Rainer Koschke
TutorIn: Shi Hui

Software–Projekt 2 2013

VAK 03-BA-901.02

Architekturbeschreibung

Sylvia Kamche Tague	clar@tzi.de	2476985
Dario Treffenfeld-Mäder	dtm@tzi.de	2598686
Nils Sören Oja	nso@tzi.de	2725302
Sandor Herms	sanherms@tzi.de	2931655
Olga Miloevich	halfelv@tzi.de	2586817
Jannes Uken	ukenj@tzi.de	2787018

Abgabe: 22.12.2013 — Version 1.1

Inhaltsverzeichnis

Version und Änderungsgeschichte

Version	Datum	Änderungen
1.0	19.11.2013	Dokumentvorlage als initiale Fassung kopiert
1.1	21.11.2013	Sammlung erster Ideen
1.2	23.11.2013	Modulsicht eingefügt
1.3	27.11.2013	Zusammenhänge zwischen AF und Arch.
1.4	30.11.2013	Globale Analyse zusammengefügt
1.5	07.12.2013	Konzeptionelle Sicht
1.6	13.12.2013	Datensicht
1.7	17.12.2013	Konzeptionelle Sicht überarbeitet
1.8	20.12.2013	Evolution und Ausführungssicht
1.9	21.12.2013	Überarbeiten der Sichten
2.0	22.12.2013	Einführung

1 Einführung

bearbeitet von: Olga Miloevich und Jannes Uken

1.1 Zweck

Zweck dieses Dokuments ist die Kommunikation zwischen allen Interessenten. Das Dokument teilt die Arbeit in die Teile, die unabhängig von einander bearbeitet werden können. Es stellt einen hohen Abstraktionsgrad vor, damit ist es sowohl für die Projekt-Mitarbeiter als auch den Kunden verständlich ist. Frühere Entwurfsentscheidungen haben nachhaltige Auswirkungen und bieten die Möglichkeit, Probleme und Unklarheiten frühzeitig zu klären. Außerdem, dient dieses Dokument als Vorlage zum Erstellen der Architektur, Implementierung und Tests.

1.2 Status

Erster Entwurf der Bibliothekverwaltungssoftware wird in diesem Dokument vorgestellt. Spätere mögliche Reviews werden auch auf diesem Dokument basiert.

1.3 Definitionen, Akronyme und Abkürzungen

1.4 Referenzen

1. Koschke, Rainer: Architekturbeschreibung-Vorlage, SWP2, WS 2013/14
2. Koschke, Rainer: 3-Hinweise-Abgabe-Architektur SWP2, WS 2013/14

3. Koschke, Rainer: 10 Architektur, annotierte Folien SWP1, SS 2013
4. Gruppe ICC, Jannes Uken et all: Architekturbeschreibung, SWP1, SS 2013

1.5 Übersicht über das Dokument

Nach der Einleitung folgt die globale Analyse. Diese beschreibt die Einflüßfaktoren und Probleme mit möglichen Lösungsstategien. Dort wird betrachtet, welche Probleme auftreten können, wie schwer sie zu lösen sind, welche Auswirkungen sie auf das gesamte Projekt haben und wie man sie lösen könnte.
Kapitel drei beschreibt konzeptionelle Sicht, die mit von Hilfe UML-Diagrammen beschrieben wird. Sowohl die Kommunikation zwischen Client und Server wird hier genauer betrachtet, als auch die Komponente vom Client und vom Server.
Kapitel vier beschreibt die Modulsicht - die Beschreibung des Aufbaus des Systems und die Verbindung zwischen Modulen durch verschiedene Schnittstellen. Hier sind auch einige UML-Klassendiagrammen zu betrachten, die verschiedene Klassen von der zu liefernden Software beschreiben.
Die Datensicht in dem Kapitel fünf beschreibt die Erweiterungen des Datenmodells aus der vorherigen Anforderungsspezifikation mit implementierungsspezifischen Änderungen. Hier werden auch UML-Klassendiagramme verwendet, damit das Ganze übersichtlicher ist.
Kapitel sechs stellt die Ausführungssicht dar, in der man die Kommunikationen zwischen Client und Server mit den dazugehörigen Elementen sehen kann.
Zusammenhänge zwischen den Anwendungsfällen und der Architektur werden in dem nächsten, siebten Kapitel beschrieben. Hier werden die Anwendungsfälle aus der Anforderungsspezifikation aus technischer Sicht betrachtet.
Das letzter Kapitel, acht, beschreibt die mögliche Evolution des Produktes.

2 Globale Analyse

2.1 Einflussfaktoren

bearbeitet von Dario Treffenfeld - Mäder, Jannes Uken, Nils Sören Oja und Sylvia Kamche Tague

2.1.1 Organisatorische Faktoren

Einflussfaktor	Flexibilität	Veränderlichkeit	Auswirkungen
O1: Time-To-Market			
Die Endabgabe ist am 23. Februar 2014. Die erste lauffähige Basisversion ist bis zum 26. Januar 2014 einzureichen.	keine	Eine Änderungen des Abgabetermins durch den Veranstalter ist sehr unwahrscheinlich.	Die Arbeitszeit ist begrenzt, sodass unter Umständen nicht genug Zeit bleibt um für jedes Problem die beste Lösung zu finden.
O2: Anzahl Entwickler			
Die Gruppe hat 6 Mitglieder.	Jeder kann für sich selbst entscheiden die Gruppe zu verlassen. Es ist auch möglich, dass die Gruppe aufgeteilt wird.	Es könnten auch durch Krankheiten Entwicklungskräfte ausfallen.	Falls die Gruppengröße sinkt, kann über die Anpassung von Time-To-Market und Mindestanforderungen verhandelt werden.
O3: Erfahrung der Entwickler			
Erfahrung der Entwickler	keine	Das angeeignete Wissen könnten vergessen werden.	Der Erfahrungsstand hat wirkt sich auf die Entwicklungsgeschwindigkeit und -qualität aus.

2.1.2 Technische Faktoren

Einflussfaktor	Flexibilität	Veränderlichkeit	Auswirkungen
T1: Hardware			
Die Hardware des Kunden, insbesondere der verfügbare Speicher, stellt eine Beschränkung dar.	keine	In dem entscheidenden Zeitraum werden keine Veränderungen stattfinden.	Es muss darauf geachtet werden, dass die Software die Hardware nicht zu sehr belastet.
T2: Programmiersprache			
Die Programmiersprache Java bietet eine Feste Menge von Möglichkeiten betreffend Klassen, Interfaces und Vererbung.	keine	Es ist unwahrscheinlich, dass sich daran in absehbarer Zeit etwas ändert.	Die entworfene Architektur muss in Java implementierbar sein, da Java als Programmiersprache vorgegeben ist.
T3: Datenbank			
Der Gebrauch einer relationalen Datenbank ist gefordert.	Wir können uns für ein Datenbanksystem entscheiden.	Eine Änderung ist unwahrscheinlich.	Die Schnittstellen des Datenbanksystems müssen berücksichtigt werden.
T4: Testbarkeit			
Die Software muss auf Richtigkeit getestet werden.	geringer Einfluss auf Testumfang	Es sind keine Veränderungen zu erwarten.	Es muss darauf geachtet werden, dass die Blackboxtests implementierbar sind.

2.1.3 Produktfaktoren

Einflussfaktor	Flexibilität	Veränderlichkeit	Auswirkungen
P1: Funktionalität			
Die Mindestanforderungen müssen erfüllt werden.	keine	Es besteht die Möglichkeit, dass weitere Anforderungen eingeschränkt oder ganz entfernt werden.	Hat allgemein großen Einluss auf die Architektur.

P2: Benutzerfreundlichkeit			
Das Produkt soll so einfach wie möglich und so schwer wie nötig zu bedienen sein.	Es gibt keine Vorschriften, sodass wir eigenständig entscheiden können.	keine	Wirkt sich auf den Bereich der Architektur aus, der direkt vom Benutzer verwendet wird.
P3: Performanz			
Die in den Anforderungsspezifikation festgelegten Ausführungszeiten sollen eingehalten werden.	Falls sich herausstellt, dass wir es nicht schaffen so zu programmieren, dass die Software schnell genug ist, könne die angaben nachträglich angepasst werden.	keine	Es muss darauf geachtet werden, dass die Software effizient arbeitet.
P4: Erweiterbarkeit			
Es ist wünschenswert, dass sich das Produkt leicht erweitern lässt.	Die Erweiterbarkeit ist optional. Um an Abstraktion zu sparen, kann auf sie verzichtet werden.	keine	Solange es kein zu großes Hindernis darstellt, kann beim Entwurf darauf geachtet werden, dass er erweiterbar bleibt.

2.2 Probleme und Strategien

Bearbeitet von Jannes Uken und Dario Treffenfeld-Mäder

2.2.1 Problemkarten 01 - Arbeitsaufwand und Zeitplan

Arbeitsaufwand und Zeitplan
Unser Abgabetermin ist fest und der Arbeitsaufwand ist groß. Das wirkt sich zwangsweise auch auf den Programmierstil aus. Hierfür müssen deshalb Strategien festgelegt werden. **Einflussfaktoren** O1: Time-To-Market O2: Anzahl Entwickler O3: Erfahrung Entwickler P1: Funktionalität
Lösungen **Strategie 1: Drei-Schichten-Architekturen** Da wir wenig Zeit haben, implementieren wir unser System in der bereits bekannten Drei-Schichten-Architekturen. Die drei Schichten sind Presentation, Business-Logik und Persistence. **Strategie 2: Vorgegebene Struktur beibehalten** Die Struktur der Vorgabe (Bibclient,Bibcommon und Bibjsf) wird beibehalten, da wir keine Android-App erstellen, sondern die Website für mobile Geräte anpassen, fällt der Bibclient aus unserem Modell raus. Bibcommon bleibt jedoch bestehen. **Strategie 3: Schnelle Lösung vor effizienter Lösung** Da die Zeit so knapp bemessen ist, ziehen wir eine schnelle einer effizienten Lösung vor, solange sie die Anforderungen erfüllt.

Die Strategien für diese Problemkarte schließen sich nicht gegenseitig aus. Die Strategien haben für sich keine großen Nachteile und können für unser Projekt zusammen eingesetzt werden.

2.2.2 Problemkarten 02 - Datensicherheit

Datensicherheit
Die Daten auf der Datenbank müssen sicher und einfach zu erreichen sein. **Einflussfaktoren** O1: Time-To-Market O2: Anzahl Entwickler O3: Erfahrung Entwickler P1: Funktionalität P3: Performanz
Lösungen **Strategie 1: Kapselung der Datenhaltung** Die Datenbank als eine eigene Komponente im System anlegen. **Strategie 2: Benutzung eines Datenbankmanagementsystems** **Strategie 3: Verwendung eines erweiterten Verschlüsselungssystems** Wir verwenden ein relationales Datenbankmanagementsystems.

Das System der relationalen Datenbank ist weit verbreitet. Wir werden auch auf diese Art unsere Daten speichern, da wir auf unsere Erfahrung mit SQL zurück greifen können und diese Datenstruktur flexibel und einfach zu handhaben ist.

Für den Aufbau unseres Systems ist die Kapselung der Daten auch von Vorteil. Da die Vorlage des Projekts schon auf diese Weise aufgebaut wurde, wäre eine andere Strategie anzuwenden, auf unnötig mehr Arbeit hinausgelaufen.

Auf Grund von Zeitmangel und dem nicht vorhandenen Wissen über die Entwicklung und Implementierung von Verschlüsselungssystemen, sehen wir davon ab, die dritte Strategie zu verfolgen.

2.2.3 Problemkarten 03 - Datenstruktur

Datenstruktur
Die Datenstruktur muss einheitlich und übersichtlich sein. **Einflussfaktoren** O1: Time-To-Market O2: Anzahl Entwickler O3: Erfahrung Entwickler P1: Funktionalität
Lösungen **Strategie 1: Businesshandler** Wir führen die Komponente Businesshandler ein, in der die Datenstrukturen implementiert sind. **Strategie 2: Common-System-Komponente** Die Common-Komponente wird so erweitert, dass sie die benötigten Datentypen (Buch/Zeitschrift etc.) implementiert und für das System eine einheitliche Datenstruktur bereit stellt.

Die Erweiterung der Common-Komponente erschien uns als sinnvoller, als den Business-Handler um diese Funktionen zu erweitern. Dieser sollte besser als Vermittler zwischen der Datenbank und der Webapp fungieren. Damit beugen wir einer Überladung des BusinessHandlers vor.

2.2.4 Problemkarten 04 - Benutzung von Java Server-Faces

Java Server-Faces
Das Benutzen von Java Server-Faces setzt eine bestimmte Komponentenstruktur voraus. **Einflussfaktoren** O1: Time-To-Market O2: Anzahl Entwickler O3: Erfahrung Entwickler P1: Funktionalität
Lösungen **Strategie 1: Kapselung der Java Server-Faces Komponenten** Wir implementieren die Komponenten XHTML, Bean und Facelet. **Strategie 2: Implementierung einer anderen Darstellungsmethode, statt der Webapp mit den Java Server-Faces** Die Implementierung einer anderen Darstellungsmethode könnte die Übertragung zwischen dem System(letztendlich der Datenbank) und dem Benutzer vereinfachen.

Wir werden die in der Vorlage bereits verwendeten Java Server-Faces benutzen, was den Arbeitsaufwand für diese Funktionen niedrig hält. Die Überlegungen des Projektteams haben keine einfacheren und ebenso benutzerfreundlichen Realisierungen finden können.

2.2.5 Problemkarten 05 - Überlastung des verwendeten Servers

Überlastung des Servers
Der Server könnte durch das Bibliothekssystem nur sehr langsam oder gar nicht funktionieren. Dies könnte nur zu bestimmten Tageszeiten auffallen, wenn zum Beispiel viele Anfragen gleichzeitig an den Server gestellt werden. **Einflussfaktoren** O1: Time-To-Market O2: Anzahl Entwickler O3: Erfahrung Entwickler P1: Funktionalität P2: Performanz T1: Hardware T2: Programmiersprache
Lösungen **Strategie 1: Effizienter Programmcode** Je effizienter der Programm-code und -stil ist, desto stabiler wird das System. Dies beugt erst nach der Auslieferung auftretenden Problemen, wie nicht ausreichend starke Server, vor. **Strategie 2: Belastungstests** Mit Hilfe von Belastungstests können die späteren Anforderungen während der Implementierung an dem System getestet werden und die Stabilität des Systems sicher gestellt werden.
Strategie 3: Kundeneinschränkungen Der Kunde gibt vor, wie stark die Server des Systems später sein werden. Daran muss sich das Entwicklungsteam orientieren.

Auf Grund der Time-To-Market können wir die Strategie 1 nicht wirklich anwenden. Die Entwicklung erfolgt natürlich so effizient wie möglich, wird jedoch durch die geringe Implementierungszeit eingeschränkt. Im Testplan sind für die zweite Strategie Belastungstests vorgesehen, die sicherstellen sollen, dass die Software auf den gegebenen Servern(Strategie 3) einwandfrei funktioniert.

3 Konzeptionelle Sicht

bearbeitet von Jannes Uken

3.1 Übersicht

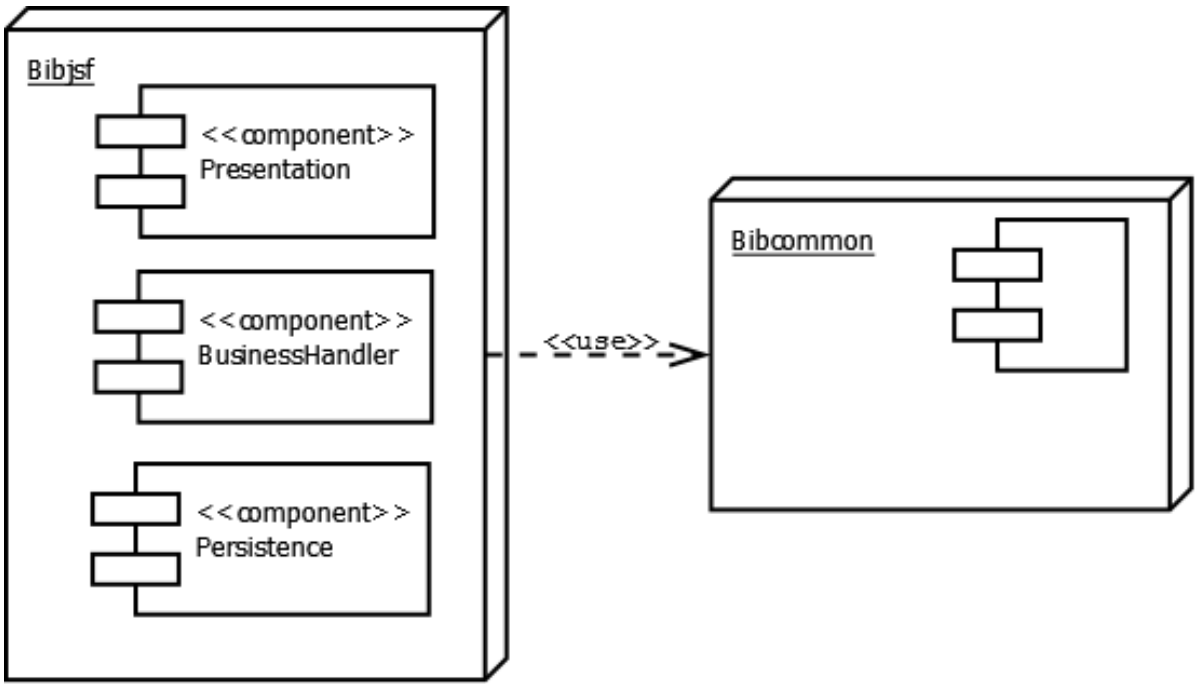

Abbildung 1: Überblick der Systemkomponente

Da die Webapp in der `Presentation`-Komponente des Servers liegt, benötigen wir keine `Client`-Komponente.

- **Server:**

  Der Server verbindet die `Presentation`-Komponente, die die Darstellung der Webapp übernimmt, mit der `Persistence`. Diese dient zur eigentlichen Kommunikation des Systems mit der Datenbank. Die Logik des Systems wird durch die `BusinessHandler` ausgeführt. Diese dienen als Verbindung zwischen der `Presentation` und der `Persistence`.

- **Common:**

  Die `common`-Komponente stellt die Klassen zur Verfügung, die innerhalb des Servers benötigt werden, um die Daten zwischen der Webapp und der Datenbank auszutauschen.

3.2 Server

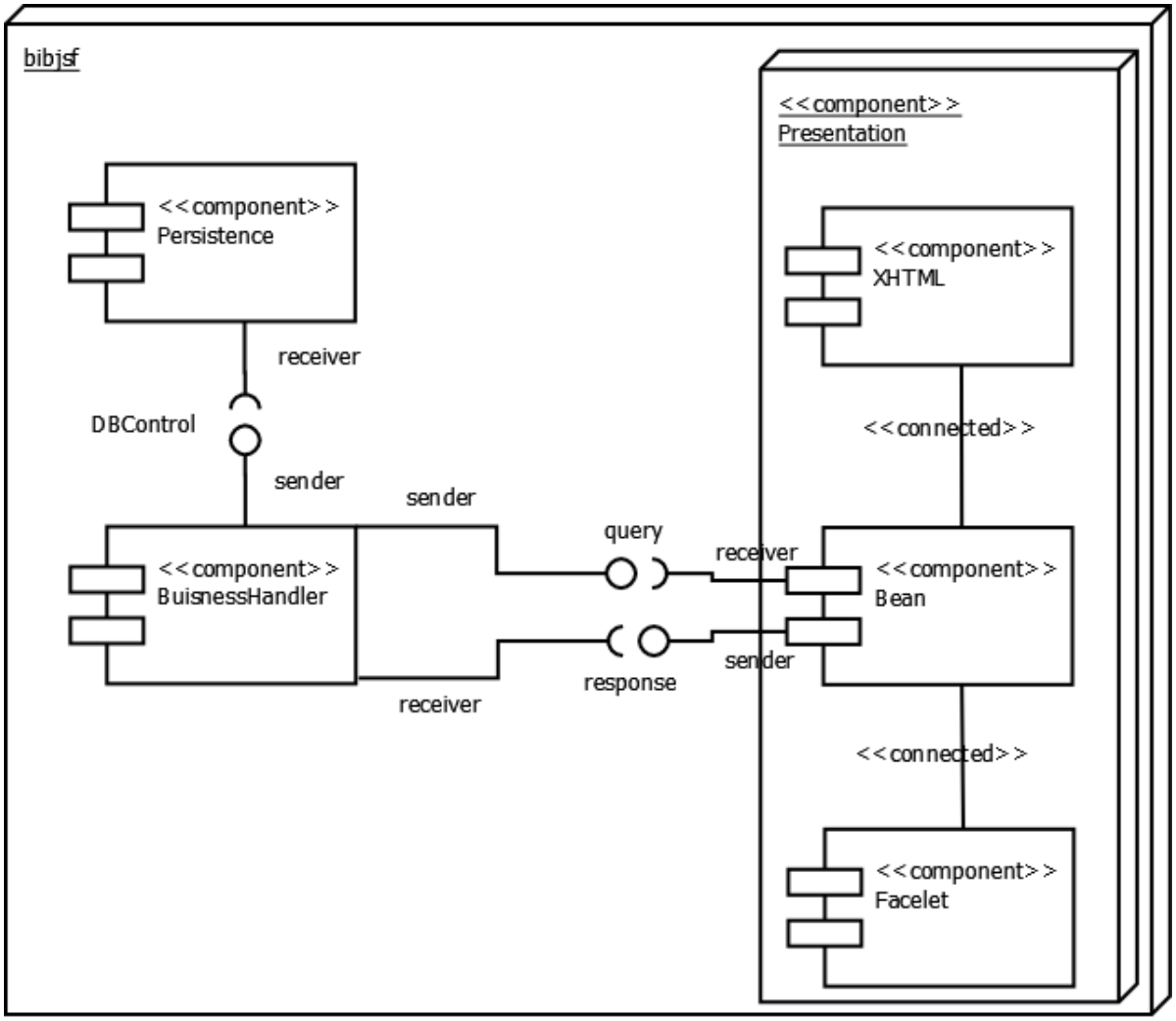

Abbildung 2: Überblick Server-Komponente

Die Darstellung wird in unserem System von den `Facelets` übernommen und die `XHTML`-Komponenten verarbeiten die Anfragen der Benutzer. Die Verbindung der beiden Komponenten mit den Beans ist durch <<connected>> gekennzeichnet. Die Anfragen werden an die `Beans` weitergegeben, die per Query und Response eine Verbindung mit den `BusinessHandlern` aufnehmen. Diese leiten die Änderungen oder Anfragen an die Datenbank weiter. Die `BusinessHandler` sind nicht direkt mit der Datenbank verbunden, sondern senden die Anfragen über die `DBControl` an die `Persistence` und entkoppelt damit die Logik von der Datenbank. Die Anfragen werden dort verarbeitet und sofern benötigt, werden Objekte erstellt und über den `BusinessHandler` an die `Beans` zurück gesendet.

3.3 Common

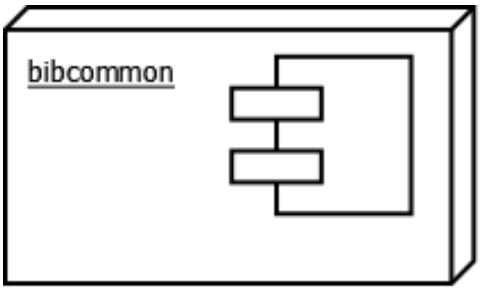

Abbildung 3: Überblick Common-Komponente

Die Kommunikation innerhalb des Systems wird größtenteils über das Versenden von Objekten realisiert.

Sowohl die `Persistence`, die `BusinessHandler` als auch die `Beans` haben Zugriff auf die Klassen der `bibcommon`, die die für die Kommunikation benötigten Klassen bereitstellt.

So wird zum Beispiel für das Erstellen eines Mediums in der `Presentation`, genauer in der `Bean` der Medium-hinzufügen-Seite, eine Instanz des Mediums erstellt und dann über den `BusinessHandler` an die `Persistence` geschickt. Die `Persistence` erweitert die Datenbank dann dementsprechend.

Analog werden bei einer Anfrage an die Datenbank die passenden Objekte erstellt, die später auch auf der `Webapp` zu sehen sein werden. Die Objekte werden über den gleichen Weg zurück an die `Beans` gesendet. Dort werden sie von den `Facelets` aufgegriffen und in der `Webapp` angezeigt.

Im Falle, dass ein Rating erstellt wird, greift auch der entsprechende `BusinessHandler` auf die `Common`-Komponente zu und legt das benötigte Objekt an.

4 Modulsicht

bearbeitet von Dario Treffenfeld - Mäder und Nils Sören Oja

Beim Entwurf der Module sind wir „bottom-up“ vorgegangen; das heißt wir haben uns zuerst überlegt, was wie in der Datenbank gespeichert werden soll, und haben anhand dessen die Datenklassen entworfen. Davon ausgehend haben wir uns in Richtung der GUI vorgearbeitet.

4.1 Businesslogic

Die Business Logic kann man als eine disjunkte Einteilung der Persistence verstehen. So wird sicher gestellt, dass Zugriffe auf ein und die selbe Tabelle der Datenbank nicht

miteinander interferieren. Dies wird erreicht, indem man die parallele Existenz mehrerer Instanzen dieser Handler-Klassen unterbindet (Singleton), und und die Methoden synchronisiert, sodass immer nur eine der Methoden eines Handlers zu jedem gegebenen Zeitpunkt aktiv sein kann.

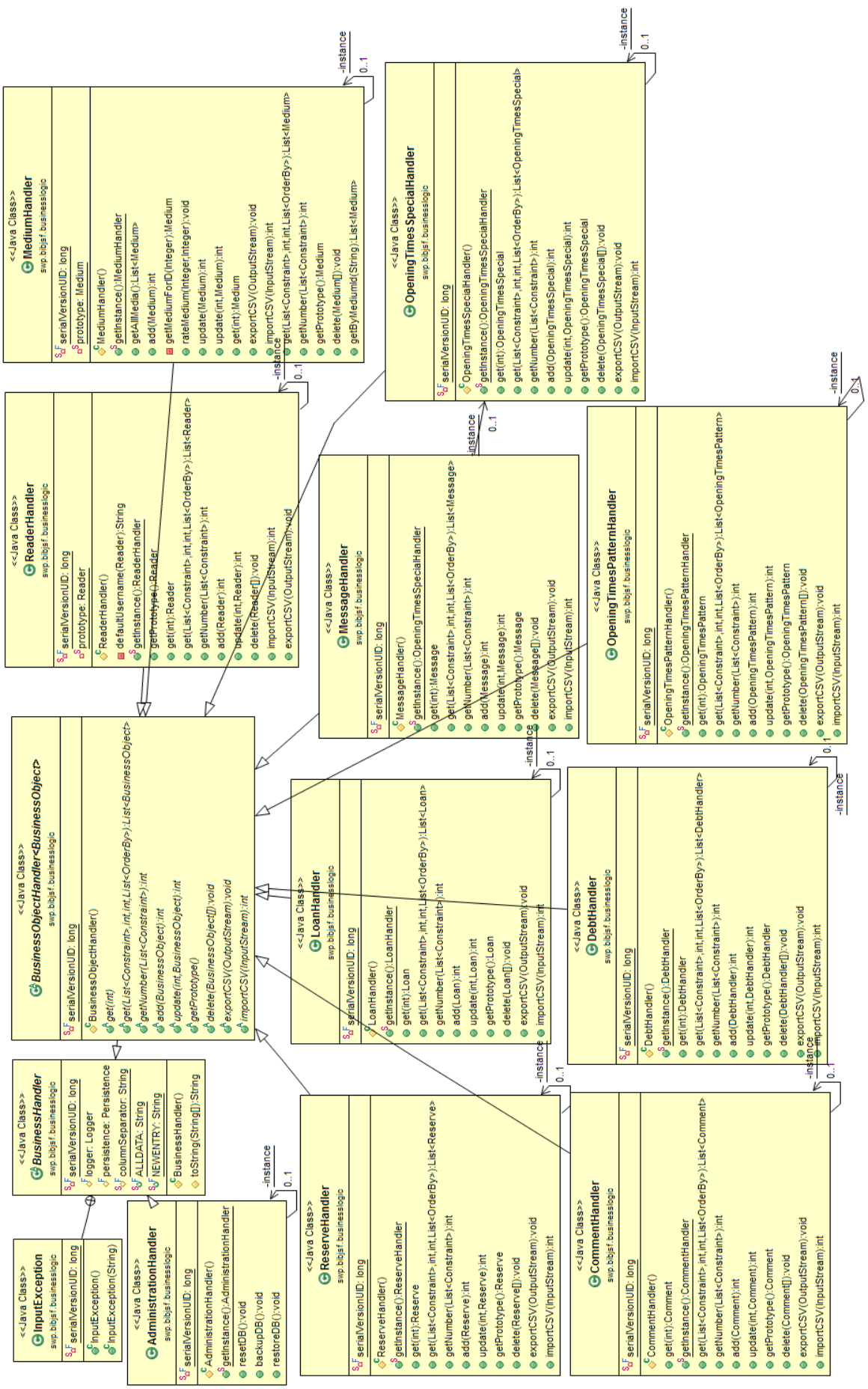

Abbildung 4: Modulsicht Businesslogic

4.2 Persistence

Über das Interface `Persistence`, welches von `Data` implementiert wird, finden die Zugriffe auf die Datenbank statt. Für jede der Datenklasse sind Methoden fürs Erstellen und Abfragen und je nach Bedarf auch fürs Aktualisieren und Löschen. Bei Klassen, die eine Relation der Leser- und Medienmenge repräsentieren, gibt es auch die Möglichkeit sich eine Liste der zu einem Leser in Relation stehenden Medien anzeigen zu lassen und umgekehrt.

Abbildung 5: Modulsicht Persistence

4.3 Presentation

In der Presentation befinden sich die Beans, für die Datenweiterleitung an die Business Logic dienen, wobei die Datenmanipulation nicht ausschließlich dort stattfindet. Für jede Tabellenart und Ansammlung von Eingabefeldern gibt es eine Klasse (Table bzw. Form) und für jeden Knopf und und jede Aktion eine Funktion in den Klassen.

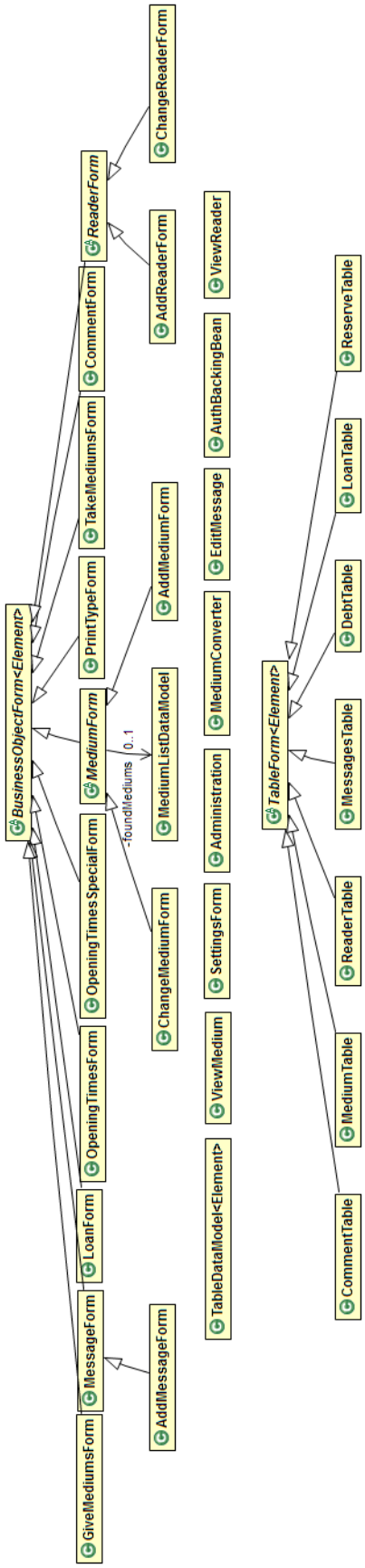

Abbildung 6: Modulsicht Presentation Übersicht

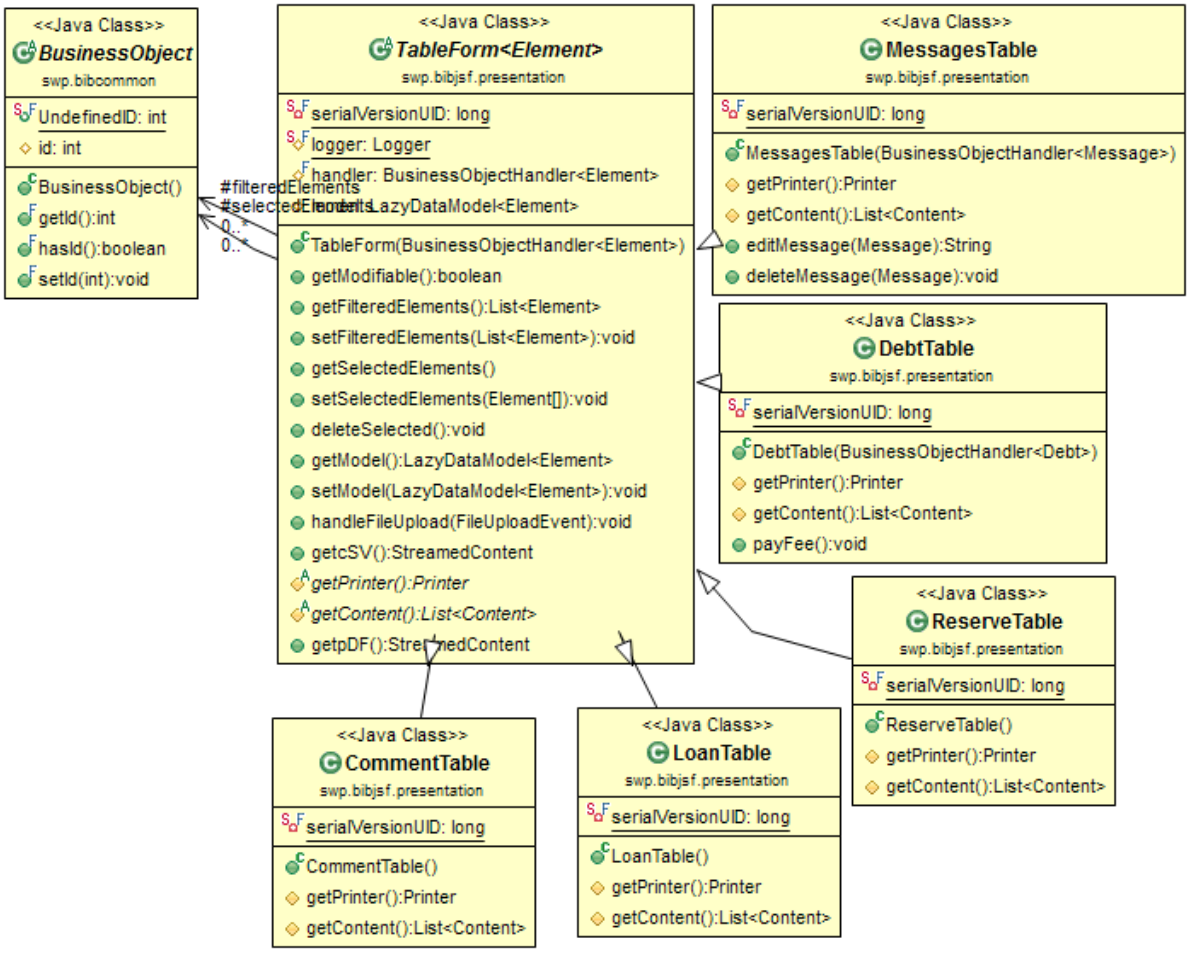

Abbildung 7: Modulsicht Presentation Tabellen

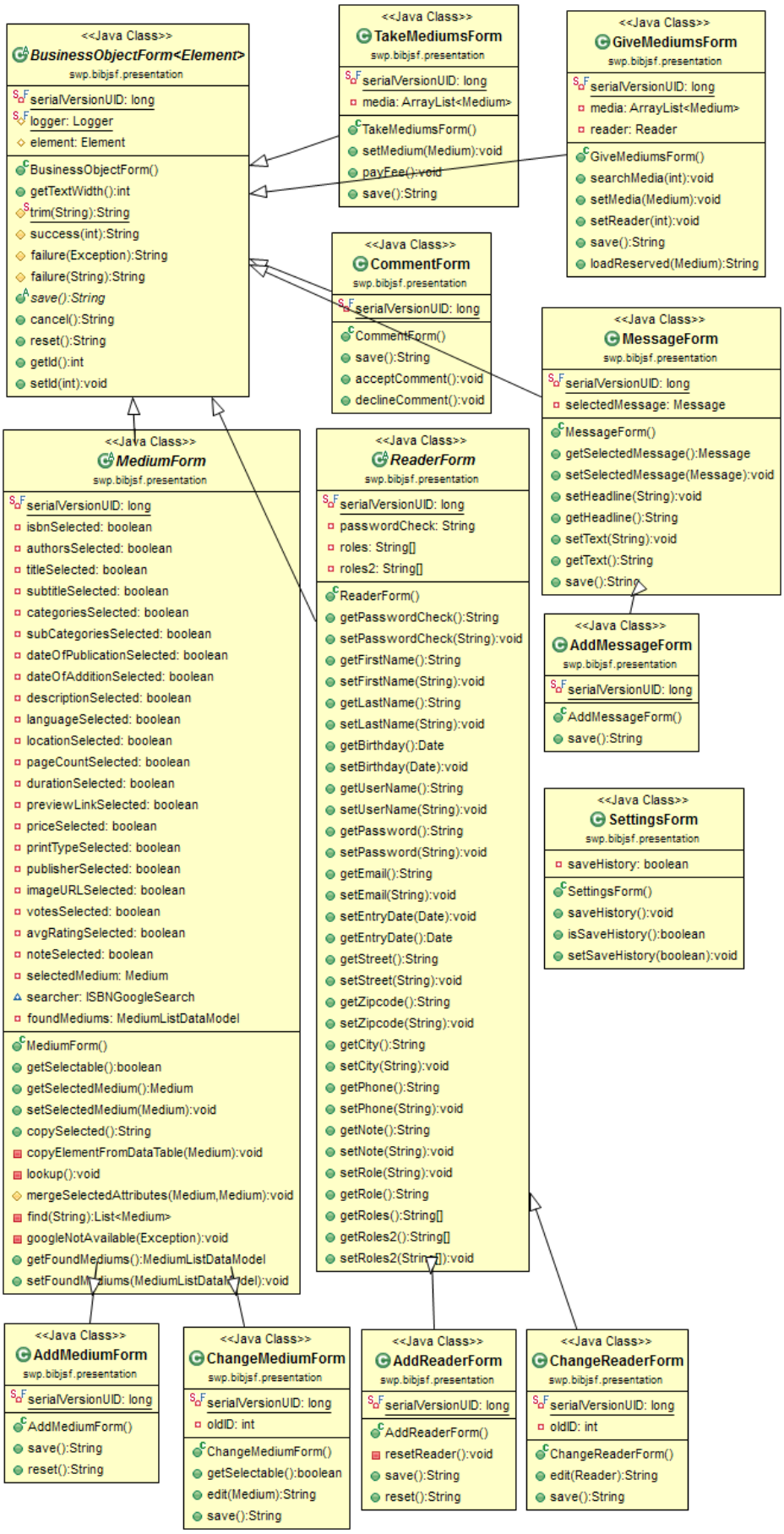

Abbildung 8: Modulsicht Presentation Formen

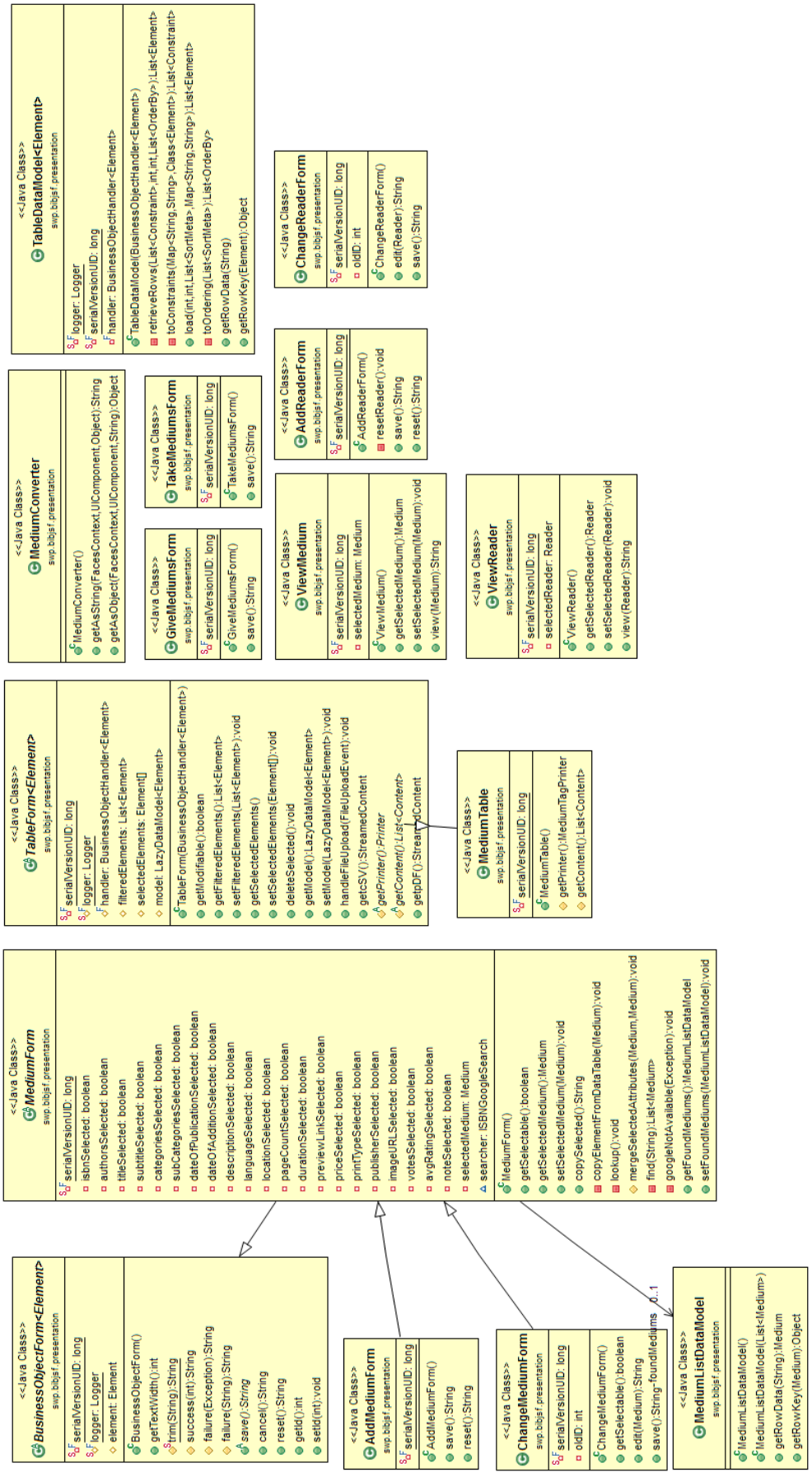

Abbildung 9: Modulsicht Presentation - Reader Medium Relation

5 Datensicht

bearbeitet von Dario Treffenfeld - Mäder , Nils Sören Oja und Sylvia Kamche Tague

Unsere Datensicht bildet den kompletten Datensatz der Architektur in einem Klassendiagramm ab (Abb. 10). Die Implementierung dieser Datenobjekte ist im Projekt Bibcommon zu finden. Die Klasse Medium repräsentiert alle Medien unseres Projektes (Buch, Zeitschriften, Software usw.). Medium hat unter anderem das Attribut PrintType. Mit ihm kann man verschiedene Medien Typen unterscheiden. Medium hat des Weiteren das Attribut mediumId. MediumId ist ein eindeutiger Identifier für Medien die es doppelt gibt, im Fall eines Buches ist dies die ISBN. Reader vertritt alle Benutzer (Normaler Ausleiher, Bibliothekar, Admin). Der Reader hat das Attribut groupid, diese beinhaltet die Art des Readers (USER, BIB, ADMIN). Außerdem gibt es noch das Attribut debt, dass die Schulden des Reader beinhaltet, sowie das Attribut unreturnedDebt, dass die Schulden von noch nicht zurück gegebenen Medien beinhaltet. Die Klasse Message stellt die vom Admin oder vom Bibliothekar versendeten Nachrichten da. Medium, PrintType, Reader und Message erben von der Klasse BusinessObject. BusinessObject gibt den Unterklassen eine ID. ReaderMediumRelation verbindet ein Medium mit einem Reader, außerdem hat sie noch ein Erstellungsdatum. Die Klassen Loan, Reserve und Comment erben von ReaderMediumRelation. Loan ist das Objekt fürs Ausleihen von Medien. Es verfügt über das Attribut extensonsTaken. ExtensonsTaken ist die Anzahl an von Verlängerungen die der Benutzer bereits bekommen hat. Reserve ist zum reservierten von Medien. Comment repräsentiert das Kommentieren der Benutzer von Medien. Comment verfügt über einen comment String der den Kommentar beinhaltet. Die Klasse Debt erbt von der Klasse Loan, sie steht für die Schulden eine Benutzers. OpeningTimesPattern setzt die normalen Öffnungszeiten der Bibliothek. OpeningTimesSpecial setzt spezielle Öffnungszeiten für Feiertage und ähnliches.

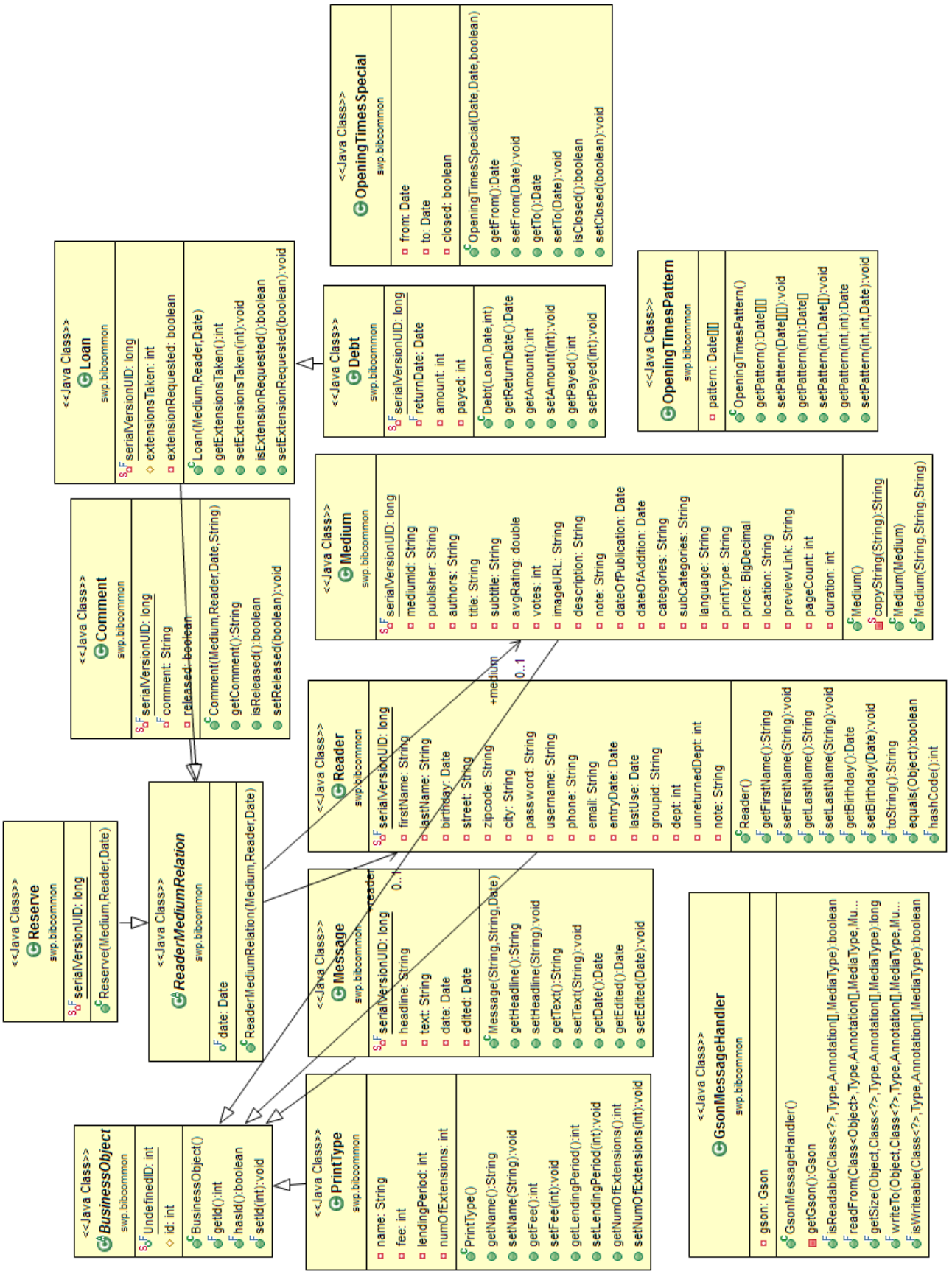

Abbildung 10: Datensicht

6 Ausführungssicht

bearbeitet von Sylvia Kamche Tague

In der Ausführungssicht (Abb. 11) wird eine Kommunikation von Elementen zwischen Client und Server dargestellt. Bei uns ist der Client der Browser. Die Kommunikation von Client zum Server erfolgt über das HTTP Protokoll, es gibt einen Server, mit dem sich mehrere Clients verbinden können. In unserer Fall haben wir im Client nur den Browser, von wo die Webseite aufgerufen wird. Da wir uns entschieden haben eine für mobile Geräte optimierte Webseite statt einer Android-App zu implementieren.

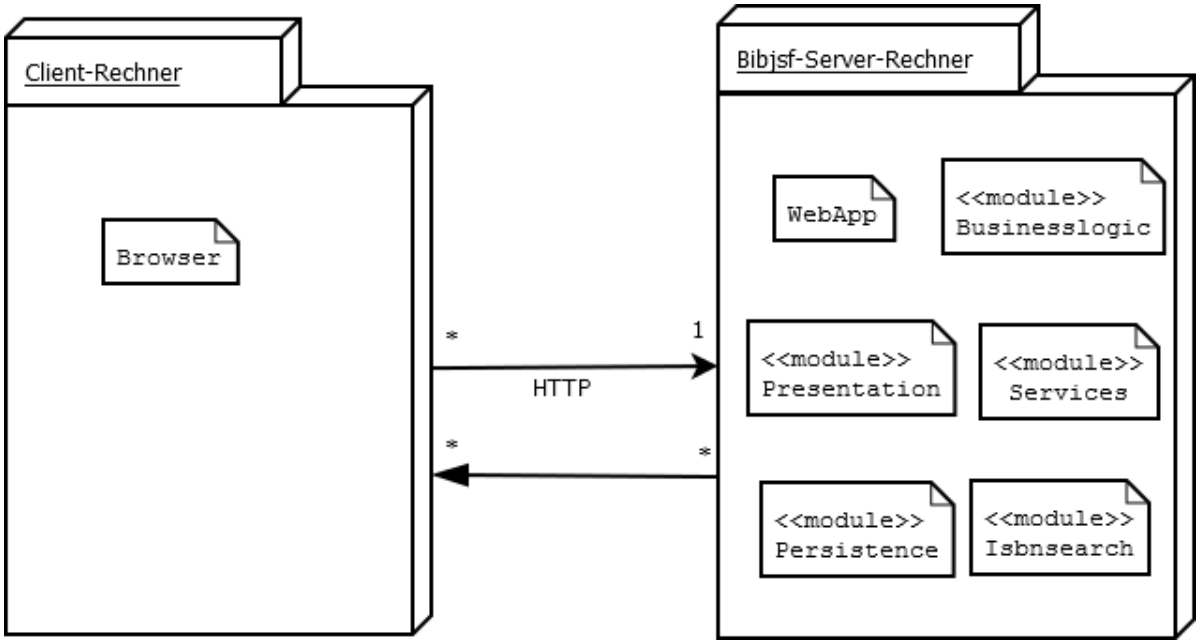

Abbildung 11: Ausführungssicht

7 Zusammenhänge zwischen Anwendungsfällen und Architektur

bearbeitet von Dario Treffenfeld - Mäder und Sylvia Kamche Tague

Anwendungsfall: Medium hinzufügen

Akteure: Berta Bibliothekar

Vorbedingung: Berta ist eingeloggt, die Startseite wird angezeigt und es liegt ein unregistriertes Medium vor.

Ablauf:

1. Berta klickt auf die Schaltfläche „Publikationen“.
2. Es öffnet sich eine Dropdown-Liste, in der Berta „Hinzufügen“ auswählt.
3. Es öffnet sich die Seite, in der die Daten des Mediums eingetragen werden müssen.
4. Berta tut dies und klickt auf den Knopf mit der Aufschrift „Speichern“.
5. Die Aktion `addBook` wird ausgeführt.

Fehler-/Ausnahmefälle: Nicht alle Felder wurden ordnungsgemäß ausgefüllt.

1. Das System zeigt eine Fehlermeldung an. (`errorConfirm`)
2. Berta überarbeitet die Eingaben und schickt eine neue Anfrage an das System.

Nachbedingung: Es existiert ein korrespondierender Eintrag für das Medium in der Datenbank.

Sequenzdiagramm: Medium hinzufügen

Bei dem Diagramm (Abb. 12) setzen wir Voraus, dass der Benutzer als Bibliothekar eingeloggt ist und sich auf der Startseite befindet.
Beim Drücken auf den ”Publikation” und dann „Hinzufügen“ wird die Seite „Medium Hinzufügen“ geöffnet, in dem der Bibliothekar die Daten des Buches eintragen kann. Wenn er diese Daten mit dem ”Speichern” Button bestätigt, wird die AddMediumForm Methode save() aufgerufen, die Wiederrum die MediumHandler Methode add(Medium) aufruft. In der Persistence ruft nun die Methode addMedium(Medium) die Methode insertByID(elment, table, minID, toIgnore, replace), das Medium ist nun in der Datenbank gespeichert.

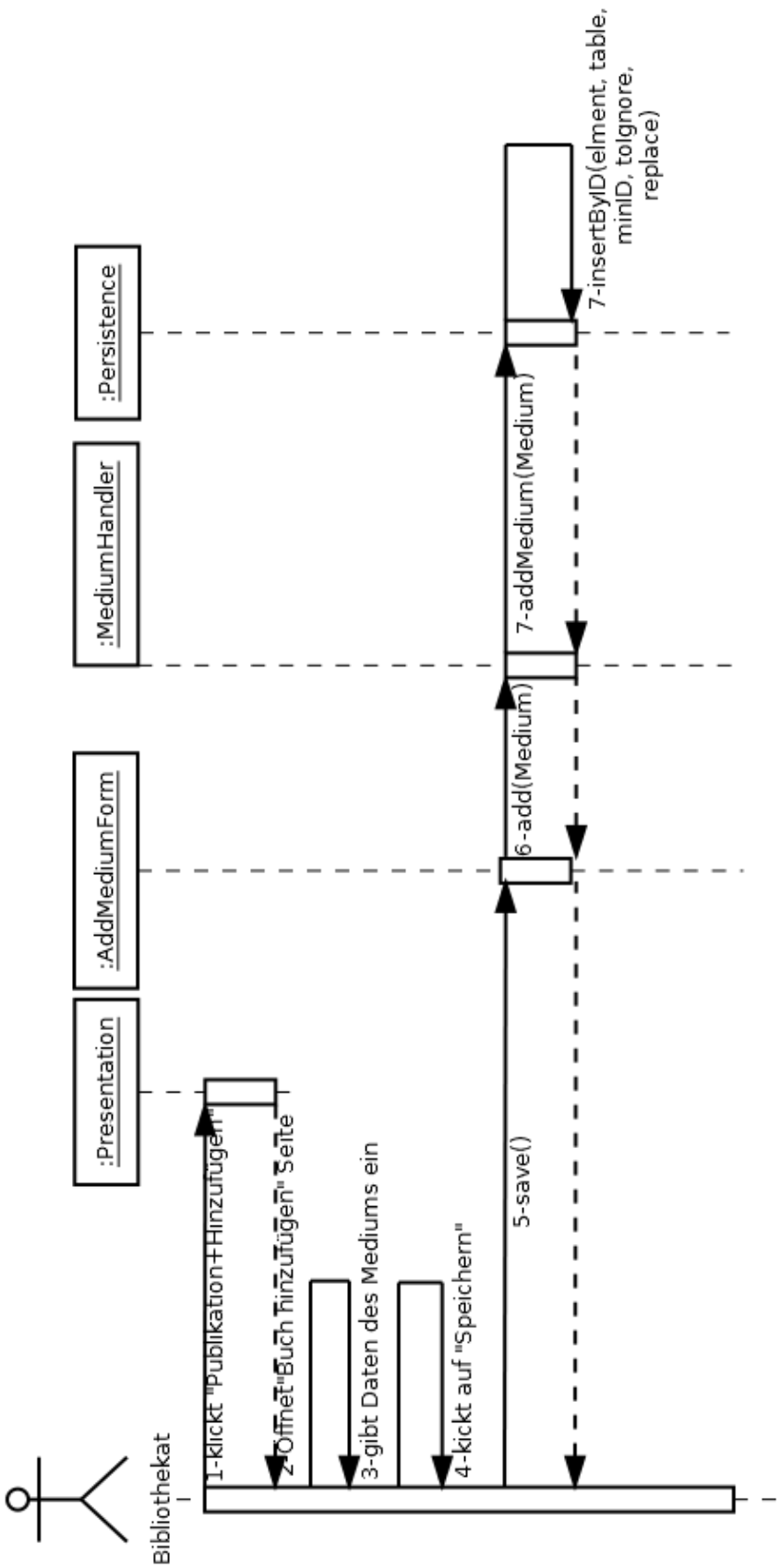

Abbildung 12: Medium hinzufügen

Anwendungsfall: Medium verleihen

Akteure: Berta Bibliothekar, Leser

Vorbedingung: Berta ist eingeloggt und die Startseite wird angezeigt.

Ablauf:

1. Berta klickt auf die Schaltfläche „Publikationen“ .
2. Es öffnet sich eine Dropdown-Liste, in der Berta „Verleihen“ auswählt.
3. Berta trägt die User-ID in das unterste Feld ein.
4. Berta gibt die Buch-ID in das dafür vorgesehene Feld ein.
5. Berta wählt das Buch in der linken Liste aus und klickt auf den Pfeil um es in die Liste der auszuleihenden Bücher hinzuzufügen.
6. Berta klickt auf „Ausleihen“ und übergibt die Bücher.

Fehler-/Ausnahmefälle: Sollte der Leser gesperrt sein, wird Berta Bibliothekar informiert und muss sein Vorgehen bestätigen oder abbrechen. (`confirm`)

Nachbedingung: Der Leser ist bei den ausgeliehenen Medien aus momentaner Besitzer eingetragen. Der Ausleihbildschirm wird angezeigt.

Sequenzdiagramm: Medium verleihen

Bei dem Diagramm (Abb. 13) setzen wir Voraus, dass der Benutzer als Bibliothekar eingeloggt ist und sich auf der Startseite befindet.
Beim Drücken auf den ”Publikation” und dann „Verleihen“ wird die Seite „Medium Verleihen“ geöffnet, in dem der Bibliothekar die Daten des Buches eintragen kann. Nun gibt der Bibliothekar die Medium ID ein, jetzt wir im MediumHandler die Methode get(id) aufgerufen. Diese Methode ruft die Methode getMedium(id) in der Persistence auf. Die Methode getMedium(id) gibt dann das gefundene Medium zur Seite zurück. Der Bibliothekar wählt nun das auszuleihende Medium aus. Jetzt gibt der die Id der Readers ein. Die Methode get(id) im ReaderHandler ruft nun die Methode getReader(id) in der Persistence auf, die wiederum den Reader zurück gibt. Jetzt drückt der Bibliothekar auf den Ausleihen Button. Dies ruft nun die Methode save() im GiveMediumsForm auf. Im LoanHandler wird nun die Methode add(Loan) aufgerufen, diese ruft dann die Methode saveLoan(Loan) in der Persistence auf. Die Ausleihung ist nun in der Datenbank gespeichert.

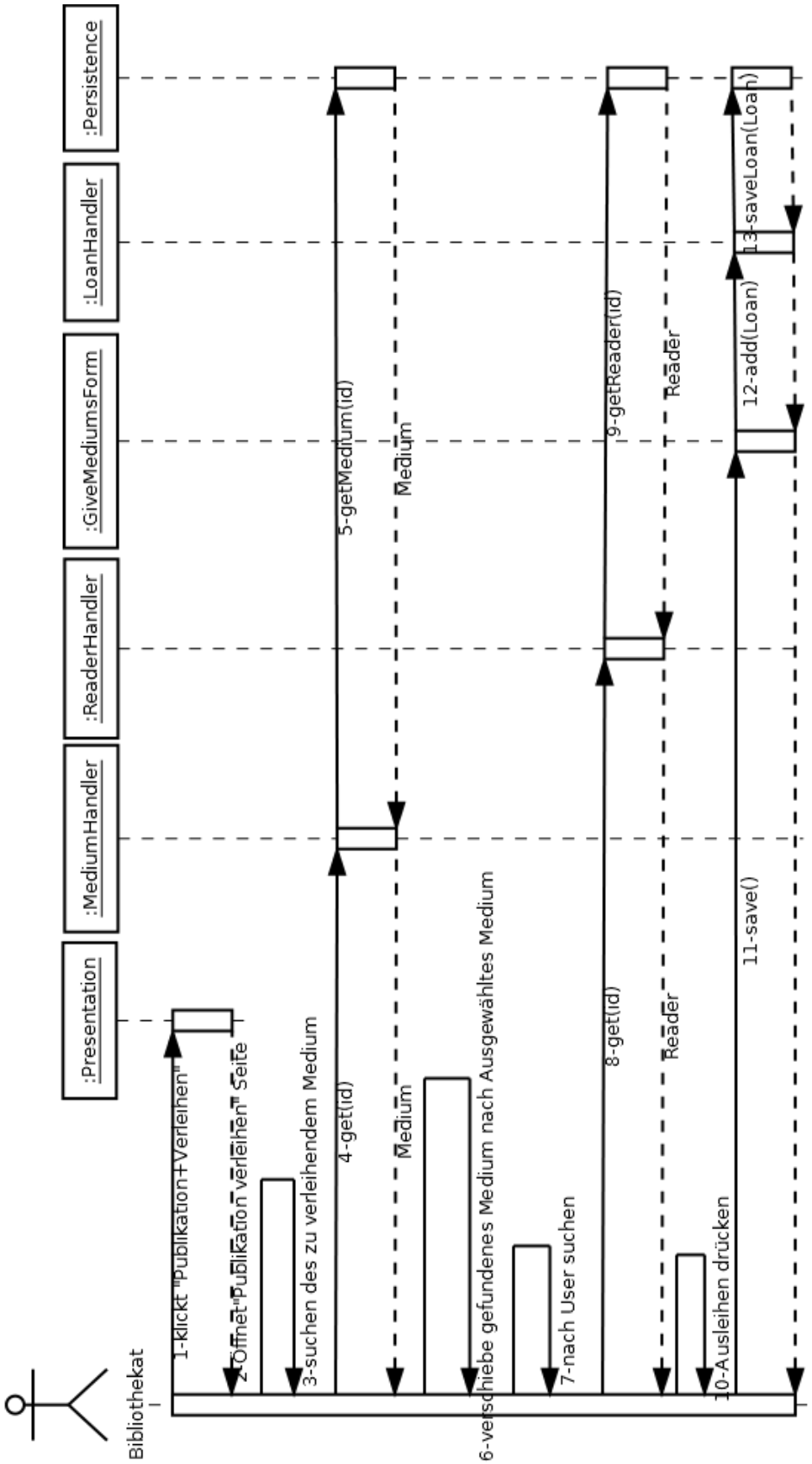

Abbildung 13: Medium Verleihen

8 Evolution

bearbeitet von Sylvia Kamche Tague und Jannes Uken

In der Anforderungsspezifikation dieses Projekts haben wir im „Ausblick“ ein paar weitere Anforderungen, die später auch erweitert werden können, zitiert. Wir können also die folgende Punkte entnehmen:

- Optionen für detailliertere Statistiken:

  Die Persistence ist so konstruiert, dass leicht neue Methoden hinzugefügt werden können. Man muss nur die Kriterien konfigurieren,dann wird die Anfrage an die Datenbank geschickt. Die Datenbank bearbeitet die Anfrage und gibt das Ergebnis an den Benutzer zurück.

- Möglichkeit für Nutzer untereinander zu kommunizieren:

  Um die Kommunikation zwischen der Benutzern zu ermöglichen, müssen sämtliche Komponenten überarbeitet werden. So müssen die grafische Oberfläche und die Netzwerkkommunikation um einen Chat erweitert werden. Dafür wird eine Liste von Onlinebenutzern erstellt und allen Onlinebenutzer gezeigt. Dazu wird auch eine „Kontakt Anfrage „ implementiert. Dann könnte also eine Person, die online ist und mit einer verfügbaren Person kommunizieren möchte, eine Einladung an eine verfügbare Person schicken. Nachdem die Person die Einladung akzeptiert hat, könnten dann die Beiden kommunizieren.

- verschlüsselte Datenübertragung:

  Hier werden alle schützenswerten Informationen sicher über ein Netzwerk von einem Standort zu einem anderen übertragen. Bei der verschlüsselten Datenübertragung werden die zu transportierenden Dateien vor der Übertragung mit einem Kryptographie-Verfahren unlesbar gemacht. Es wird dann sowohl beim Server als auch beim Client die SSL-Verschlüsselung verwendet.

- ein „Undo-Button“:

  Dieses Button wird der GUI hinzugefügt. Die Aktion dieses Buttons wäre mindestens seine letzte Handlung rückgängig zu machen. Es wird dafür eine Methode „Undo()“ implementiert. Server und Datenbank-Seitig würde diese Erweiterung einen großen Aufwand darstellen.

- automatische Löschung von Benutzerkonten abgegangener Schüler:

  Mit dieser Option werden die Konten von allen Schülern, die nicht mehr in der Schule sind automatisch gelöscht. Es wird jedes Mal geprüft, ob alle Schüler, die ein Konto besitzen, auch in der Liste der aktuellen Schüler der Schule enthalten sind. Wenn nicht, dann wird sein Konto gelöscht. Dies würde durch eine Anfrage an die Datenbank durchgeführt.