RWTH AACHEN UNIVERSITY

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Bachelorarbeit im Studiengang
Scientific Programming

Entwicklung eines erweiterbaren, sicheren Systems zur zentralen Benutzerverwaltung von dezentralen Benutzerdatenbanken

Fachhochschule Aachen, Campus Jülich
Fachbereich Medizintechnik und Technomathematik

von Mathias Kohs
Matrikelnummer 845781

03.01.2015

Erstbetreuer: Prof. Dr. -Ing. Andreas Terstegge
Zweitbetreuer: Dipl.-Inform. Roman Breuer

Eigenständigkeitserklärung

Hiermit versichere ich, Mathias Kohs (Matrikel Nr. 845781), dass ich diese Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und diese Arbeit in gleicher oder ähnlicher Form noch nicht Bestandteil einer Studienleistung war.

03.01.2015, Aachen

_____

Mathias Kohs

Danksagung

Hiermit bedanke ich mit bei allen, die mir bei der Ausarbeitung dieser Arbeit geholfen haben.

Mein Dank gilt meinen Betreuern Prof. Dr.-Ing. Andreas Terstegge und Dipl.-Inform. Roman Breuer die mich bei der Anfertigung hilfsbereit unterstützt und immer fleißig Fragen beantwortet haben.

Einen besonderen Dank möchte ich an meine Menschen richten, die mich bei Fragen der Rechtschreibung beraten haben.

Ich bedanke mich auch bei meiner Familie und meinen Freunden und Arbeitskollegen für super Unterstützung über das ganze Studium ☺

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Liste von Akronymen

LFI	Lehr-und Forschungsgebiet Ingenieurhydrologie der RWTH-Aachen University
CRUD	Create Read Update Delete (Erstellen Lesen Editieren Löschen)
GUI	Grafic User Interface (Grafische Benutzer Oberfläche)
Config	Configuration (Konfiguration im allgemeinem Sinne kontextabhängig auch Konfigurationsdatei)
EzPh.	Entwickler zweiter Phase
OU / OrgaUnit	OrganisationsUnit (Künstliche Struktur zum Ordnen von Personengruppen)
JSON	JavaScript Object Notation (Datenformat zum Austauschen zwischen Applikationen)
XML	Extensible Markup Language (Hierarchische Sprache zur Strukturierung von Daten und Austausch zwischen Applikationen)
DTD	Document Type Definition / Schema-Definition (Sprache zum Beschreiben von XML)
XSD	XML Schema Definition (Schemasprache zum Beschreiben von XML, in XML)
IDE	Integrated Development Environment (Integrierte Entwicklungsumgebung)
MATSE	Mathematisch technische Softwareentwickler
Fisi	Fachinformatiker Fachrichtung Systemintegration
Tec-Hiwi	Nichtwissenschaftliche Hilfskräfte mit technischem Aufgabenbereich
UDP	User Datagram Protocol (Netzwerkprotokoll der Transportschicht)
TCP	Transmission Control Protocol (Netzwerkprotokoll der Transportschicht)
OSI-Model	Open Systems Interconnection Model (Schichtenmodell für Netzwerkprotokolle)
DBMS	Database Management System (Verwaltungs-System, was das Anlegen und Benutzen von Datenbankstrukturen ermöglicht)
ER-Model	Entity-Relationship-Model (Modell zum Darstellen von Datenbank-Modellen, mit einem hohen Abstraktion-Grad, als Relationen und Entitäten)
DB-Model	Datenbank-Model (Modell zum Darstellen von Datenbank-Modellen, mit einem mittlerem Abstraktions-Grad, als Tabellen mit Attributen)
nRt	nicht Referenztabelle (alle Tabellen, die nicht als Referenztabelle deklariert sind)
WhereJoin	Ein JOIN der nicht im FROM-Part eines SQL-Statements beschrieben wird, sondern im WHERE-Part
DML	Data Manipulation Language Entspricht dem Teil der Datenbanksprache, der CRUD formuliert

1 Motivation

Das Lehr- und Forschungsgebiet Ingenieurhydrologie, der RWTH-Aachen University, hat viele Services, die auf eigenen Servern laufen, welche von Institutsmitarbeitern, Hochschulmitarbeitern anderer Institute und Studenten genutzt werden. Bei diesem Unterfangen gibt es einen hohen administrativen Aufwand der Benutzerverwaltung, der durch Fachinformatiker und Mathematisch technische Softwareentwickler (MATSE) in diesem Institut getragen wird. Ein wichtiger Punkt in der Verwaltung ist die Übersicht.

In diesem Kontext sind die gängigsten Fragen:

- Welche Benutzerkonten dieses Instituts sind auf dem System X eingetragen?
- Wie viele Benutzerkonten hat das Institut A auf dem System Y eingetragen?
- Auf welchen Systemen hat die Person B Benutzerkonten?

Diese Übersicht ist schwer herzustellen, wenn man alle Systeme einzeln abfragen muss. Aufgrund dessen gab es schon Ansätze für Systeme zum Pflegen dieser Informationen. Diese Systeme sind aus verschiedenen Gründen gescheitert.

Beispiele der genannten Gründe sind:

- Wartungsunfreundlichkeit und damit starke Alterung
- Fehlende Akzeptanz unter den Benutzern aufgrund nötiger manueller Pflege
- Fragliche Sicherheit der Daten

Im Rahmen dieser Bachelorarbeit soll ein Softwaresystem entworfen werden, welches zentral die Daten über Benutzerkonten verschiedener Systeme hält und diese administrieren kann. Erreicht werden soll, dass die Übersicht verbessert und der Administrationsaufwand minimiert wird. Zu berücksichtigen ist die Erweiterbarkeit und die Absicherung der Daten gegen böswillige Fremdbeeinflussung.

2 Anforderungsanalyse

Die Anforderungsanalyse wurde mehrfach durchlaufen und damit die Anforderung schrittweise im Dialog mit den Verantwortlichen verbessert. Auf diese Anforderungsanalyse stützt sich der folgende Softwareentwurf.

2.1 Stakeholder

Im Englischen werden diejenigen Personen als Stakeholder bezeichnet, welche Anforderungen formulieren können (vgl. [1], S.52). Die Gruppe der Stakeholder besteht aus vielen Untergruppen, die nicht ausschließlich aus den Endanwendern besteht, sondern aus allen, die durch das zu entwickelnde System tangiert werden. In Tabelle 1 bis Tabelle 9 werden die betrachteten Stakeholder und deren Tätigkeit aufgeführt.

Administrator	
Wer ist das genau?	Eine ausgewählte Gruppe aus Tec-Hiwis, Matse und Fisi
Was sind die Tätigkeiten?	Installieren und konfigurieren von Produktteilen und Serversysteme.

Tabelle 1: Stakeholder Administratoren

Endanwender	
Wer ist das genau?	Ein ausgewählte Gruppe aus Tec-Hiwis, Matse und Fisis (nicht identisch mit der Gruppe der Administratoren)
Was sind die Tätigkeiten?	Verwenden das fertige Produkt und verwalten die Benutzerkonten der unterschiedlichen Systeme.

Tabelle 2: Stakeholder Endanwender

Halter der Benutzerkonten	
Wer ist das genau?	Studenten, Mitarbeiter des Instituts, Mitarbeiter der Hochschule und externe Partner
Was sind die Tätigkeiten?	Sind nur indirekt betroffen, da ihre Konten mit dem System verwaltet werden.

Tabelle 3: Stakeholder Halter der Benutzerkonten

Systemprüfer	
Wer ist das genau?	Mathias Kohs, Roman Breuer
Was sind die Tätigkeiten?	Machen die Abnahme nach der Inbetriebnahme und legen fest, ob das System in den Produktivbetrieb geht.

Tabelle 4: Stakeholder Prüfer

Entwickler (erster Phase)	
Wer ist das genau?	Mathias Kohs
Was sind die Tätigkeiten?	Erstellung und Entwurf des Systems sowie der Dokumentation für Vervollständigung und Erweiterung des Systems und Umsetzung der Software.

Tabelle 5: Stakeholder Entwickler (erster Phase)

Entwickler (zweiter Phase) [EzPh]	
Wer ist das genau?	Matse
Was sind die Tätigkeiten?	Entwickeln die Software nach erster Inbetriebnahme weiter und passen diese an.

Tabelle 6: Stakeholder Entwickler (zweiter Phase)

Inbetriebnehmer	
Wer ist das genau?	Mathias Kohs
Was sind die Tätigkeiten?	Nimmt das fertige System in Betrieb und schult die Administratoren, Entwickler und Endanwender.

Tabelle 7: Stakeholder Inbetriebnehmer

Projekt und Produktgegner	
Wer ist das genau?	Intern gibt es keine Produktgegner, da sich alle Vereinfachung und Beschleunigung erhoffen. Nach außen hat das System keine Schnittstelle.
Was sind die Tätigkeiten?	---

Tabelle 8: Stakeholder Projekt und Produktgegner

Kulturkreis	
Wer ist das genau?	Institutsmitarbeiter sind Personen, die den westeuropäischen Kulturen angehören und der deutschen Sprache mächtig sind.
Was sind die Tätigkeiten?	---

Tabelle 9: Stakeholder Kulturkreis

2.2 Anwendungsfälle

Die Anwendungsfälle wurden zusammen mit den Stakeholdern gezeichnet und mehrfach verfeinert. Schon während des Zeichnens und Formulieren der Anwendungsfälle wurde klar, dass ein mehrteiliges System von Vorteil sein wird. In Abbildung 1 bis Abbildung 7 sind UseCases beschrieben, die in die unterschiedlichen Teile des Systems gegliedert sind.

Die zu unterscheidenden Bestandteile des Systems sind:

- Drittsystem

  Das Drittsystem ist das zu administrierende System. Dieses System stellt kein direktes Bestandteil des Gesamtsystems dar, da es durch den Satelliten-Client angebunden wird. Im Lauf der Arbeit auch Satellitensystem genannt.

- Central-Server

  Der zentrale Server hat Schnittstellen zum GUI-Client, zum Setup-Tool und über den Satellitenclient zum Satellitensystem. Dieser Server soll die Daten der Satelliten halten und auswerten.

- Satelliten-Client

  Der Satelliten-Client ist die Verbindung zum Drittsystem und kommuniziert mit dem Central-Server.

- GUI-Client

  Der GUI-Client ist die Schnittstelle zum Endanwender (siehe 2.1 Stakeholder) des Gesamtsystems. Über diesen Client werden alle Vorgänge der Benutzerverwaltung durch den Endanwender getätigt.

- Setup-Tool

  Das Setup-Tool ist ein Werkzeug für das Einrichten des Gesamtsystems, welches eine Verbindung zum Central-Server aufbauen kann.

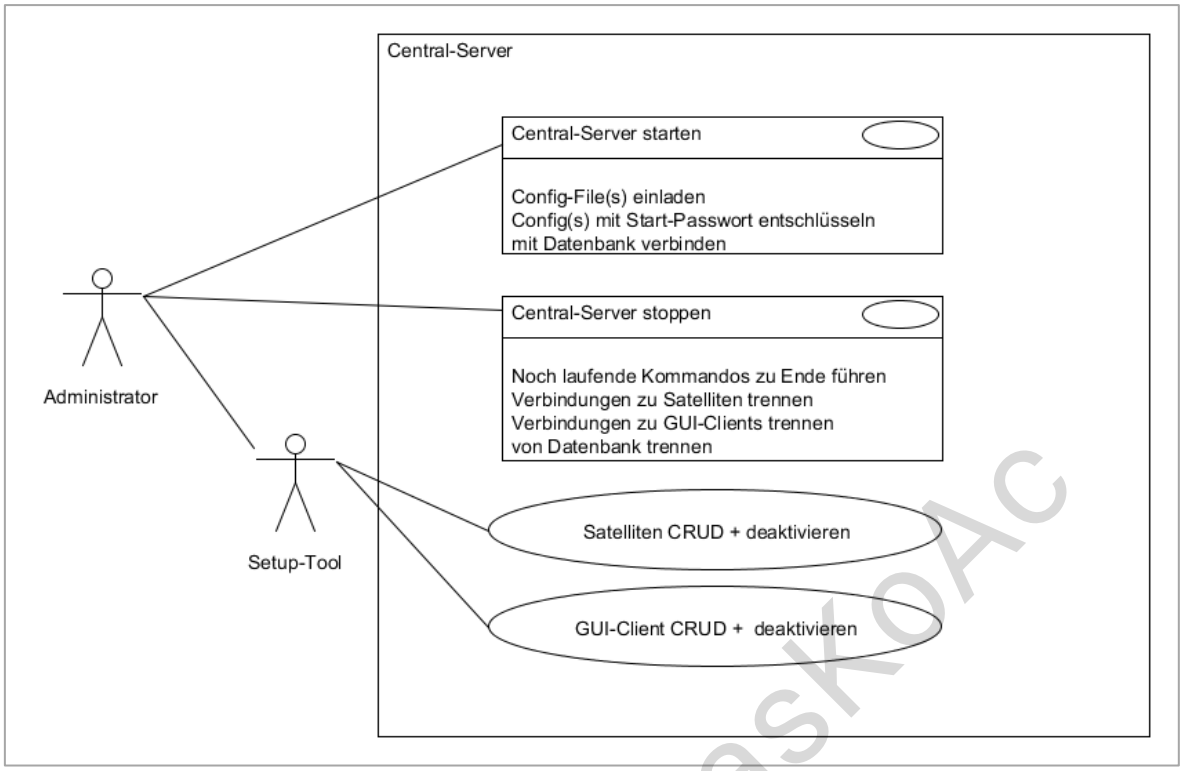

Abbildung 1: UseCase Central-Server Administrator

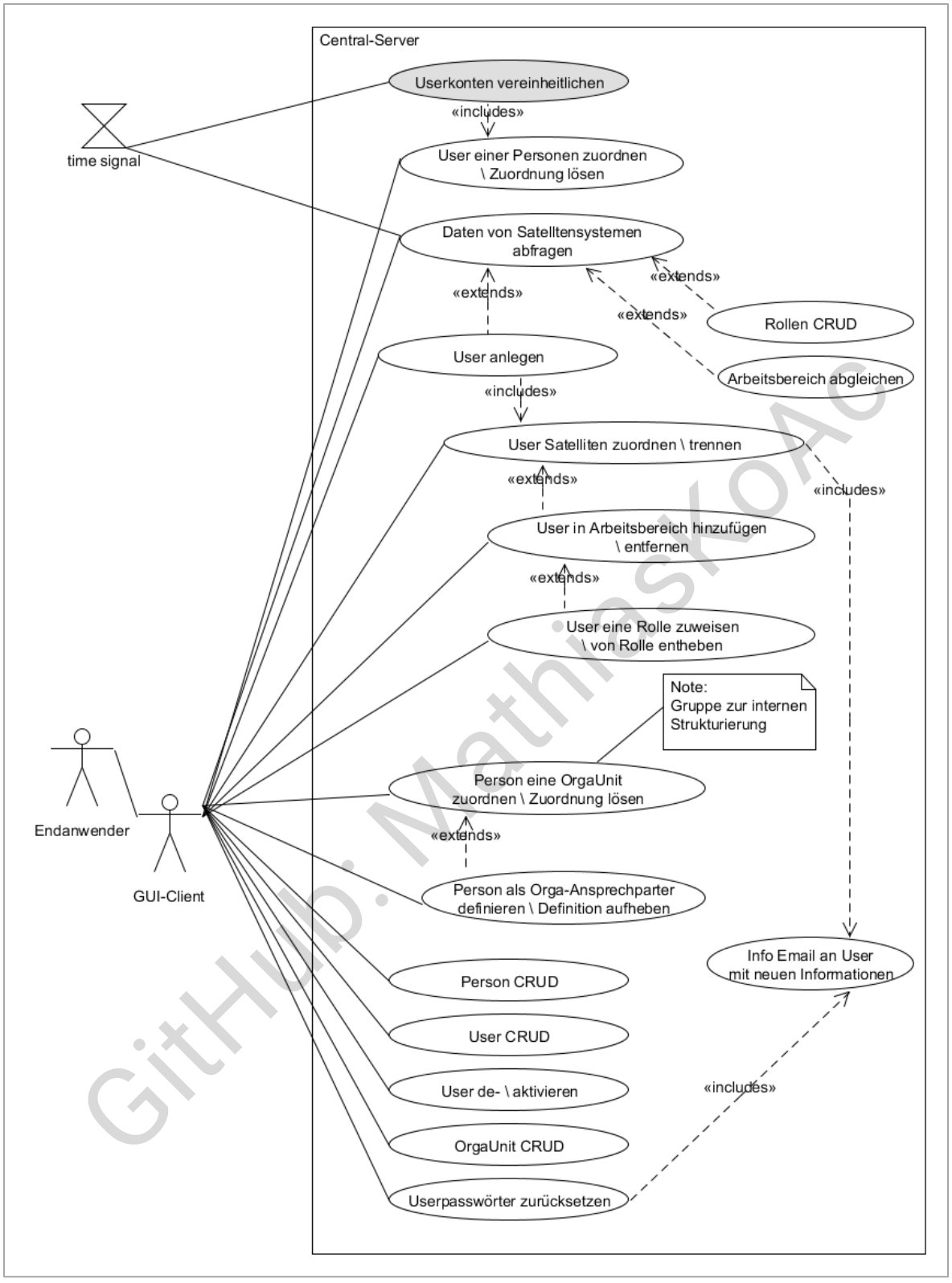

Abbildung 2: UseCase Central-Server Endanwender

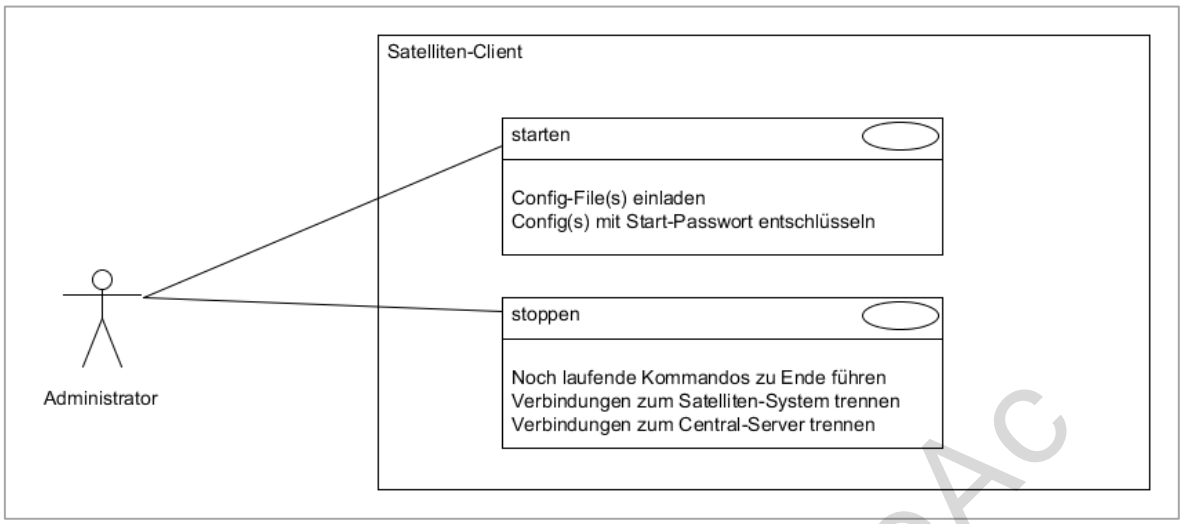

Abbildung 3: UseCase Satelliten-Client Starten

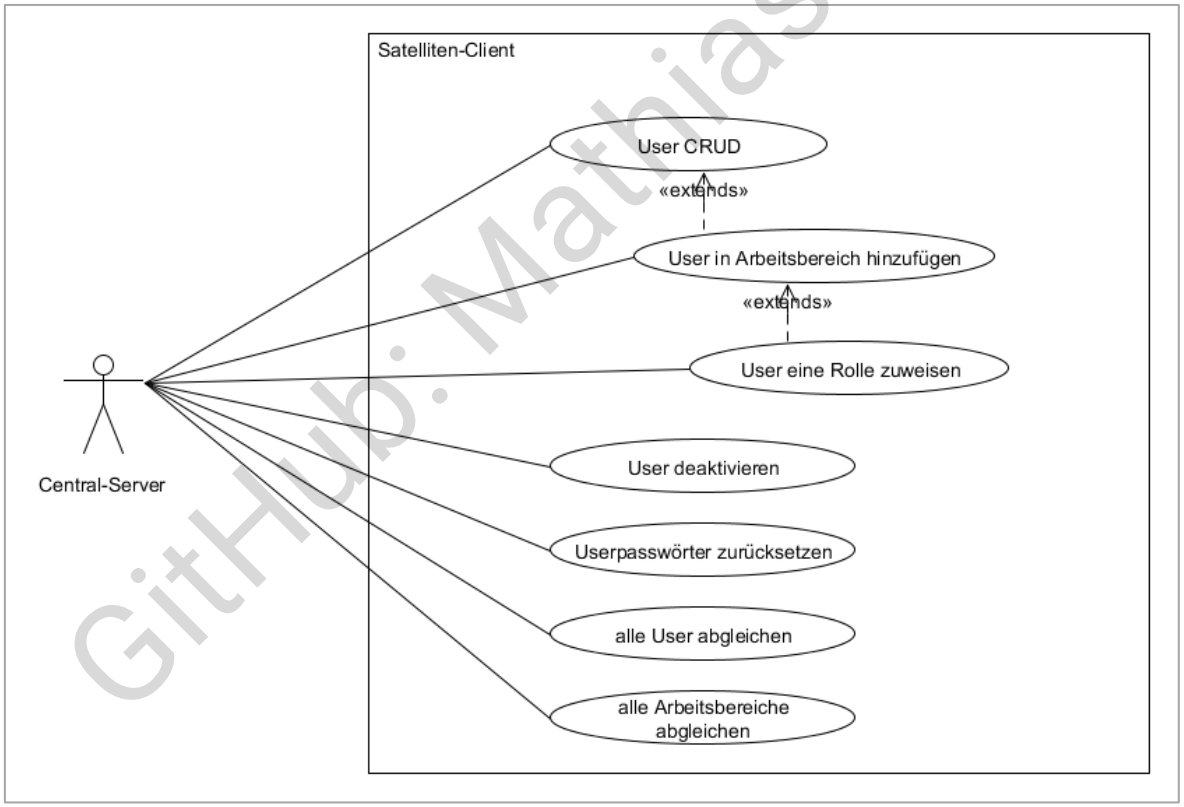

Abbildung 4: UseCase Satelliten-Client Central-Server

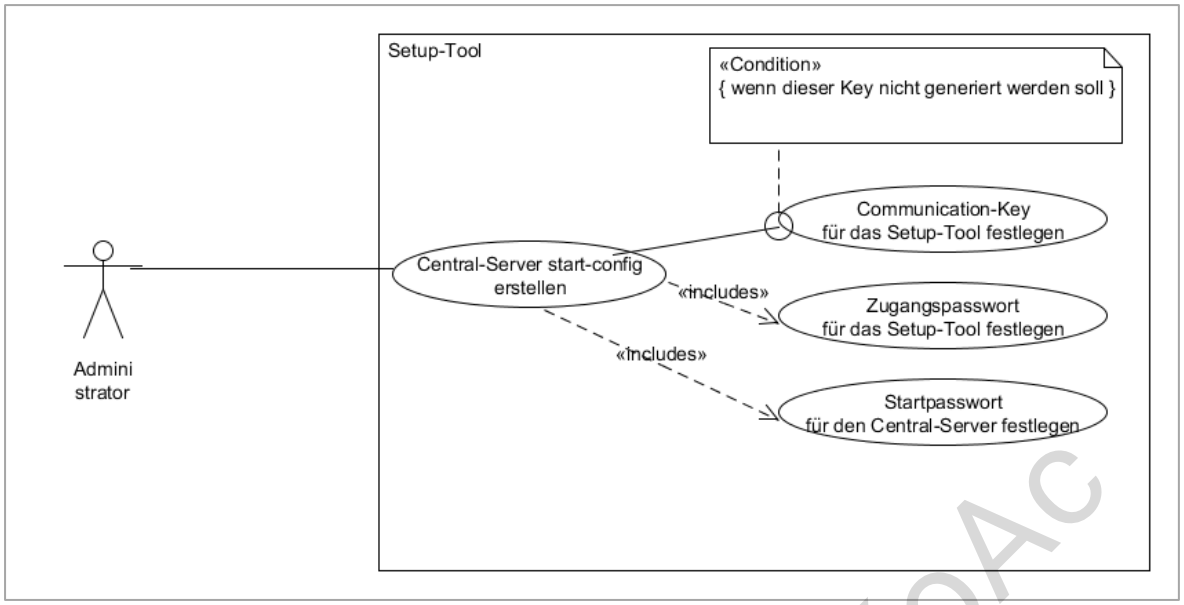

Abbildung 5: UseCase Setup-Tool Central-Server

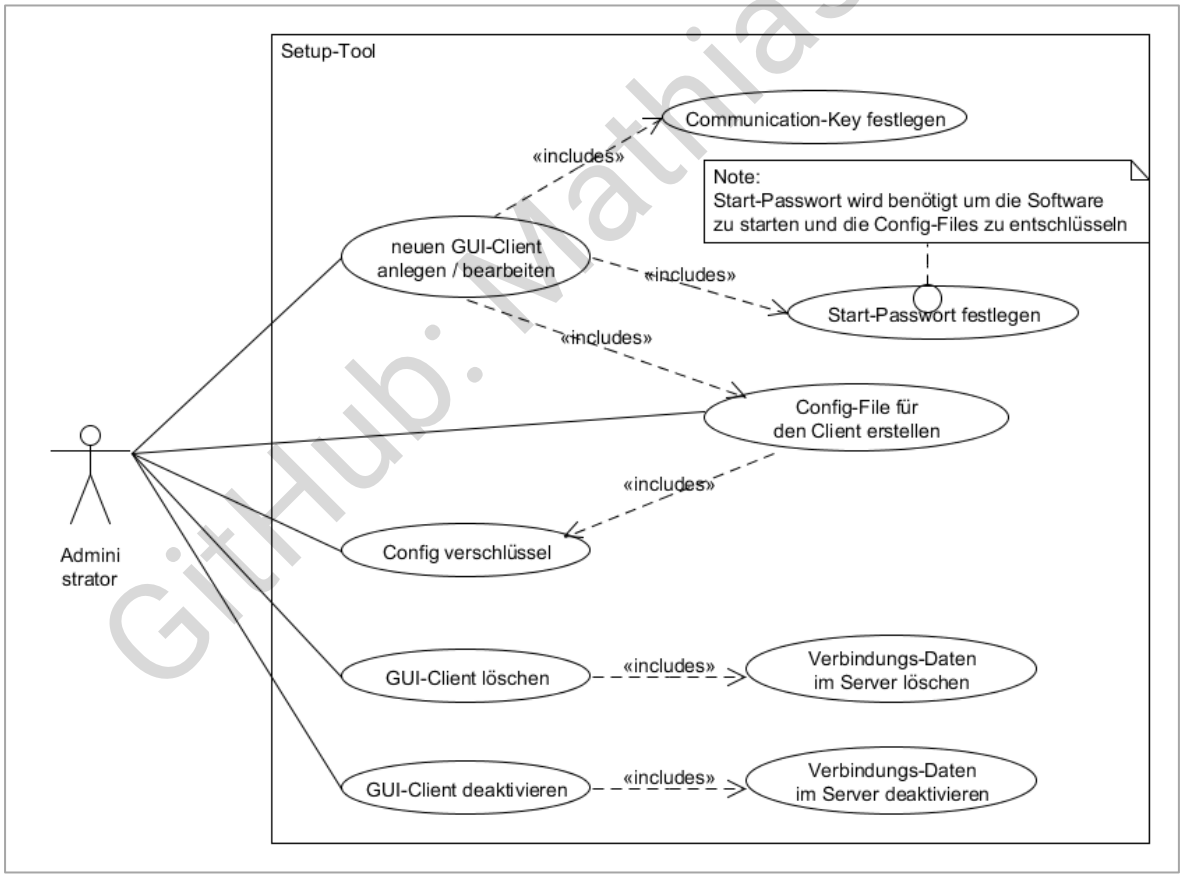

Abbildung 6: UseCase Setup-Tool GUI-Client

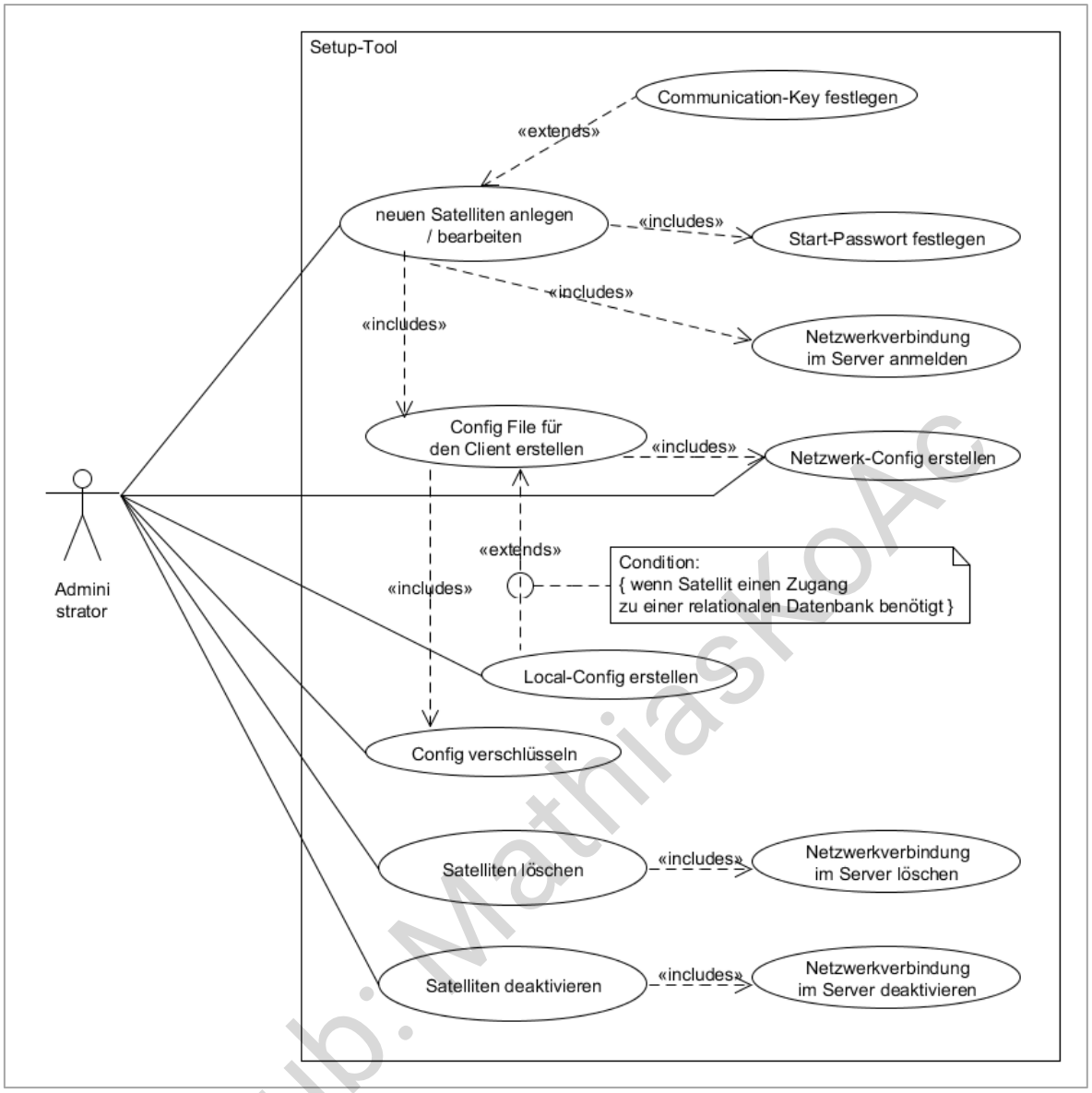

Abbildung 7: UseCase Setup-Tool Satellit

2.3 Funktionale Anforderungen

Die Funktionalen Anforderungen wurden aus den UseCases (Anwendungsfällen) abgleitet und teilweise erweitert. Die Formulierung der Anforderungen geschah in Anlehnung an die Anforderungsschablone nach Rupp (vgl. [1] S.73).

2.3.1 Central-Server

Die textuellen Funktionalen Anforderungen für den Central-Server beziehen sich auf die Abbildung 1 und Abbildung 2.

2.3.1.1 Start / Stopp

Der Central-Server muss:

1. Anforderung	Durch den Administrator gestartet werden können
2. Anforderung	Dem Administrator die Möglichkeit geben, das Start-Passwort einzugeben
3. Anforderung	Nach dem Start die Verbindung mit der Datenbank aufbauen
4. Anforderung	Durch den Administrator gestoppt werden können
5. Anforderung	Nach dem Erhalt des Stoppsignals laufende Kommandos zu Ende führen
6. Anforderung	Nach dem Erhalt des Stoppsignals alle Verbindungen zu den Satelliten trennen
7. Anforderung	Nach dem Erhalt des Stoppsignals alle Verbindungen zu den GUI-Clients trennen
8. Anforderung	Nach dem Erhalt des Stoppsignals die Verbindung zur Datenbank trennen

2.3.1.2 Zeitgesteuert

Der Central-Server muss nach eingestelltem Zeitsignal:

9. Anforderung	Userkonten vereinheitlichen
10. Anforderung	Nach Durchlauf der 9. Anforderung automatisch User-Person-Zuordnungen durchführen
11. Anforderung	Daten von Satellitensystemen abfragen
11.1.	User abgleichen
11.2.	User Satelliten zuordnen
11.3.	User Arbeitsbereiche zuordnen / Zuordnung entfernen (optional)
11.4.	Arbeitsbereiche abgleichen (optional)
11.5.	Userrollen abgleichen (optional)

2.3.1.3 Über den GUI-Client

Nach erfolgreicher Verbindung von GUI-Client und Central-Server muss das System dem Endbenutzer die Möglichkeit geben:

12. Anforderung	User einer Person zuzuordnen / Zuordnung zu trennen
13. Anforderung	Die Daten von Satellitensystemen abzufragen
14. Anforderung	User Satelliten zuzuordnen / Zuordnung zu trennen
15. Anforderung	User in Arbeitsbereiche hinzuzufügen / zu entfernen
16. Anforderung	User Rollen zuzuweisen / zu entheben

17. Anforderung	Organisationseinheit CRUD
18. Anforderung	Personen einer Organisationseinheit zuzuordnen / Zuordnung trennen
19. Anforderung	Eine Person als Ansprechpartner für eine Organisationseinheit zu definieren / Definition aufheben
20. Anforderung	Person CRUD
21. Anforderung	User CRUD
22. Anforderung	Userpasswörter zurückzusetzen
23. Anforderung	Nach Ausführen von der 14. Anforderung und 22. Anforderung soll das System selbständig eine Email an den betroffenen User versenden, um ihn über die neuen Daten zu informieren

2.3.2 Satelliten-Client

Die textuellen Funktionalen Anforderungen für den Satelliten-Client beziehen sich auf die UseCases aus Abbildung 3 und Abbildung 4.

2.3.2.1 Start / Stopp

Der Satelliten-Client muss:

24. Anforderung	Durch den Administrator gestartet werden können
25. Anforderung	Durch den Administrator gestoppt werden können
26. Anforderung	Dem Administrator die Möglichkeit geben das Start-Passwort einzugeben
27. Anforderung	Nach der Eingabe des Start-Passworts das Config-File entschlüsseln
28. Anforderung	Nach dem Erhalt des Stopp-Signals die laufenden Kommandos zu Ende führen
29. Anforderung	Nach dem Erhalt des Stopp-Signals die Verbindung mit dem Satelliten-System trennen
30. Anforderung	Nach dem Erhalt des Stopp-Signals die Verbindung mit dem Central-Server trennen

2.3.2.2 Einfluss durch Central-Server

Nach erfolgreicher Verbindung von Central-Server und Satelliten-Client muss der Satelliten-Client dem Central-Server die Möglichkeit geben:

31. Anforderung	User CRUD
32. Anforderung	User in Arbeitsbereiche hinzuzufügen
33. Anforderung	User eine Rolle zuzuweisen
34. Anforderung	User zu deaktivieren

35. Anforderung	Userpasswörter zurückzusetzten
36. Anforderung	Alle User abzugleichen
37. Anforderung	Alle Arbeitsbereiche abzugleichen
38. Anforderung	Nach Ausführung von 31. Anforderung in direkter Verbindung 32. Anforderung auszuführen
39. Anforderung	Nach Ausführung von 32. Anforderung in direkter Verbindung 33. Anforderung auszuführen

2.3.3 Setup-Tool

Die textuellen Funktionalen Anforderungen für das Setup-Tool beziehen sich auf die UseCases aus Abbildung 5, Abbildung 6 und Abbildung 7.

2.3.3.1 Konfiguration des Central-Servers

Das Setup-Tool muss dem Administrator die Möglichkeit geben:

40. Anforderung	Eine Konfigurationsdatei für den Start des Central-Server zu erstellen
41. Anforderung	Bei Erstellung der Konfigurationsdatei
41.1.	einen Communications-Key für das Setup-Tool festzulegen
41.2.	ein Zugangspasswort für das Setup-Tool festzulegen
41.3.	ein Startpasswort für den Central-Server festzulegen

2.3.3.2 Konfiguration des GUI-Clients

Das Setup-Tool muss dem Administrator die Möglichkeit geben:

42. Anforderung	GUI-Clients CRUD + deaktivieren
42.1.	GUI-Client zu löschen
42.2.	GUI-Client zu deaktivieren
43. Anforderung	Bei der Durchführung der 42. Anforderung :
43.1.	den Kommunikations-Key festzulegen
43.2.	das Startpasswort festzulegen
43.3.	die Konfigurationsdatei zu erstellen
44. Anforderung	Nach der Durchführung der 43. Anforderung die erstellte Datei zu verschlüsseln
45. Anforderung	Bei der Durchführung der 42.1 Anforderung die zugehörige Netzwerkverbindung im Central-Server zu löschen
46. Anforderung	Bei der Durchführung der 42.2 Anforderung die zugehörige Netzwerkverbindung im Central-Server zu deaktivieren

2.3.3.3 Konfiguration des Satelliten-Clients

Das Setup-Tool muss dem Administrator die Möglichkeit geben:

47. Anforderung	Satelliten-Client CRUD + deaktivieren
47.1.	Satelliten-Client zu löschen
47.2.	Satelliten-Client zu deaktivieren
48. Anforderung	Bei der Durchführung der 47. Anforderung :
48.1.	den Kommunikations-Key festzulegen
48.2.	das Startpasswort festzulegen
48.3.	die Konfigurationsdatei zu erstellen
48.4.	die Netzwerkverbindung im Central-Server anzumelden
48.5.	die lokale Konfiguration des Satelliten festzulegen (optional)
49. Anforderung	Nach der Durchführung der 48. Anforderung die erstellte Datei zu verschlüsseln
50. Anforderung	Bei der Durchführung der 47.1 Anforderung die zugehörige Netzwerkverbindung im Central-Server zu löschen
51. Anforderung	Bei der Durchführung der 47.2 Anforderung die zugehörige Netzwerkverbindung im Central-Server zu deaktivieren

2.3.4 Schnittstellen

2.3.4.1 Verbindung annehmend

Der Central-Server muss die Möglichkeit bieten,

52. Anforderung	Dass sich ein oder mehrere GUI-Clients ,mit dem Central-Server, verbinden können
53. Anforderung	Dass sich ein Setup-Tool verbinden kann

Der Satelliten-Client muss die Möglichkeit bieten,

54. Anforderung	Dass sich der Central-Server verbinden kann

2.3.4.1.1 Verbindung erstellend

55. Anforderung	Der Central-Server muss in der Lage sein, sich mit dem Satelliten-Client zu verbinden
56. Anforderung	Der GUI-Client muss in der Lage sein, sich mit dem Central-Server zu verbinden
57. Anforderung	Das Setup-Tool muss in der Lage sein, sich mit dem Central-Sever zu verbinden

2.4 Technische Anforderung

Technische Anforderungen werden getrennt von „Funktionalen Anforderungen“ und „Nicht Funktionalen Anforderungen“ betrachtet und sind nötig damit das System in dem bestehenden technischen Umfeld funktionierend eingebettet werden kann (vgl. [2] S.15f).

58. Anforderung	Der GUI-Clients muss auf Windows 7, Windows 8 und Debian 6 und Debian 7 laufen
59. Anforderung	Der Central-Server muss auf Debian 6 und 7 ohne GUI laufen
60. Anforderung	Der Satellit muss auf Debian 6 und 7 ohne GUI laufen

2.5 Nicht Funktionale Anforderungen

Nicht Funktionale Anforderungen oder auch Qualitätsanforderungen sind Anforderungen, die die Güte oder die Effizienz, Änderbarkeit, Erweiterbarkeit, Benutzbarkeit und Zuverlässigkeit eines Produktes oder Prozesses beschreiben (vgl. [2] S.16, S.256f / vgl. [1] S.77f).

2.5.1 Konfigurierbarkeit

61. Anforderung	Das System soll Administratoren die Möglichkeit geben, mittels des Setup-Tools Einstellungen für das System und dessen Teilsystem festzulegen

2.5.2 Erweiterbarkeit

62. Anforderung	Das System soll Entwicklern (zweiter Phase) die Möglichkeit mittels offener Schnittstellen geben, neue GUI-Clients zu implementieren
63. Anforderung	Das System soll Entwicklern (zweiter Phase) mittels Java-Interfaces die Möglichkeit geben, Konnektoren zu anderen Datenbank-Systemen zu implementieren
64. Anforderung	Das System soll Entwicklern (zweiter Phase) die Möglichkeit geben, mittels Java-Interfaces Algorithmen für die Bindung von Personen und Benutzerdaten anzulegen
65. Anforderung	Das System soll Entwicklern (zweiter Phase) mittels offener Schnittstellen die Möglichkeit geben, neue Satelliten Clients zu implementieren

2.5.3 Benutzbarkeit

66. Anforderung	Der Zugriff auf Teilanwendungen darf nur über Passwortabfrage geschehen
67. Anforderung	Der Start von Teilanwendungen darf nur nach Passwortabfrage geschehen

3 Technologiewahl

Die Wahl der Technologie ist abhängig von den Anforderungen und den Stakeholdern. Wichtig für dieses System ist, nach Inbetriebnahme weiter entwickelt und angepasst zu werden, damit das System nicht veraltet. Für die weitere Anpassung und Weiterentwicklung sind die „Entwickler zweiter Phase“ (EzPh) verantwortlich (vgl. Kapitel 2.1).

3.1 Kommunikation

Mit Kommunikation ist der Informationsaustausch zwischen den Teilsystemen gemeint. Es liegt im Sinne der Wartbar- und Erweiterbarkeit, für die Kommunikation Sprachen zu nehmen, die durch die Häufigkeit der Verwendung einem „quasi Standard“ nahe kommen.

3.1.1 JSON oder XML

Sowohl JSON als auch XML sind generell für die Übertragung und Speicherung von Daten geeignet, den EzPh vertraut sowie plattformunabhängig. Beide Sprachen bzw. Notationen sind für die Verwendung in einer objektorientierten Umgebung gedacht. Für die Programmiersprache Java passt XML besser als JSON, da XML und Java beide typisierte Sprachen sind.

	JSON	**XML**
Plattformunabhängig	JA	JA
Den EzPh bekannt	JA	JA
Typisiert	NEIN	JA

Tabelle 10: Technologie JSON oder XML

3.1.2 XML-Schema Sprache

DTD (Documenttypedefinition) und XSD (XML-Schema-Definition) sind geeignet, um XML zu beschreiben. Die XSD ist mächtiger, als die DTD und ermöglicht speziell XML an Datenmodelle von Java zu binden. Durch die Bindung von XML und Java entfällt für den Entwickler das Parsen des XML.

	DTD	**XSD**
Plattformunabhängig	JA	JA
Den EzPh bekannt	NEIN	JA
Passend zur Programmiersprache	NEIN	JA
Unterschiede in Datentypen herausstellbar	NEIN	JA
Benutzung für XML-Binding möglich	NEIN	JA

Tabelle 11: Technologie DTD oder XSD

3.1.3 XML-Binding-Tool

Für das Binden von XML an Java gibt es mehrere Werkzeuge. Die bekanntesten sind JiBX von Dennis Sosnoski, Apache XMLBeans der XMLBeans Community und JAXB von Sun Microsystems. Alle drei Systeme haben verschiedene Vorteile und leicht unterschiedliche Funktionsweisen. JAXB wird seit der Version 6 mit dem JDK ausgeliefert, ist den EzPh bereits bekannt und wird voraussichtlich den folgenden EzPh weiter bekannt bleiben, was die Wahl auf JAXB fallen lässt.

	JiBX	**Apache XMLBeans**	**JAXB**
Weiterentwicklung	Unklar	NEIN	JA
IDE Plugin	JA (Eclipse)	Unklar	JA (Eclipse)
Ausreichend dokumentiert	JA	JA	JA
Lizenz	BSD	Apache Lizenz 2.0	CDDL and GPL
Bezugsmöglichkeit	vgl. [3]	vgl. [4]	im JDK und vgl. [5]
Den EzPh bekannt	NEIN	NEIN	JA

Tabelle 12: Technologie XML-Binding

3.1.4 Netzwerk Protokoll

Für die Art der Kommunikation kommen generell sowohl UDP (User Datagram Protocol) als auch TCP (Transmission Control Protocol) infrage (Protokoll der OSI-Ebene 4). Da für die Anwendung eine stabile Verbindung wichtiger als eine schnelle und verzögerungsfreie ist, wird TCP als verbindungsorientiertes, zuverlässiges Protokoll gewählt.

Das Attribut „zuverlässig“ bezieht sich hier auf die Vollständigkeit, Richtigkeit der Reihenfolge und Verhinderung von Duplikaten bei der Übertragung von Daten.

	UDP	**TCP**
verbindungsorientiert	NEIN	JA
zuverlässig	NEIN	JA
verzögerungsarm	JA	NEIN

Tabelle 13: Technologie OSI-Ebene 4 Protokoll

3.2 Datenspeicherung

Die Datenspeicherung ist in diesem Projekt wichtig, um Anwendungsdaten persistent zu halten sowie die Konfigurationen der System-Teile zu tätigen und zu verteilen.

3.2.1 Datenhaltung für Anwendungsdaten

Für Datenhaltung der Anwendungsdaten bietet sich eine Datenbank an. Die Daten werden im Laufe der Anwendung zunehmen und die Abfragen komplexer. Den EzPh sind auf Grund Ihrer Ausbildung Relationalen Datenbanken bekannt. Im LFI wird häufig das Datenbank-Management-System MySQL verwendet, was den Administratoren ebenfalls sehr geläufig ist. Das DBMS MySQL bietet die Möglichkeit große Datenmengen abzubilden und ist ACID vollständig, was dann Ansprüchen genügt. Aufgrund des vorhandenen KnowHows und der damit verbunden Akzeptanz, fällt die Wahl auf MySQL.

3.2.2 Speicherung der Konfigurationen

Die Speicherung der Konfiguration ist für alle vier Teile des Gesamtsystems wichtig. Jedes System hat Einstellungen, die geladen werden müssen, bevor der Start der Hauptarbeit beginnt. Der Central-Server hat die Verbindungsdaten zur Datenbank, GUI-Client und Satelliten-Client haben die Verbindungsdaten zum Central-Server. Für die Übertragung wurde bereits XML gewählt, aus denselben Gründen aus denen XML für die Kommunikation gewählt wurde, wird es für die Speicherung von Einstellungen gewählt. Zusätzlich muss man durch die Wahl von XML keine zwei Technologien mischen. Bereits bei der Übertragung stellte sich XML als besonders geeignet heraus, auch für die Speicherung von Einstellungen wird XML gewählt.

4 Softwareentwurf

Der Softwareentwurf basiert auf der Anforderungsanalyse und der Technologiewahl, die in Teilen in Kapitel 2, Kapitel 3 und im Anhang dokumentiert ist.

4.1 Komponenten

Laut dem Buch „UML 2 glasklar“ lassen sich die Notationselemente verschiedener Strukturdiagramme zusammen in einem Diagramm nutzen (vgl. [6] S.14-15). Für die Beschreibung der Komponenten in dem Gesamtsystem werden nicht wie im UML-Standard beschrieben „Component Diagrams“ genutzt, sondern eine Kombination aus „Component Diagram“ und „Deployment Diagram“ verwendet.

Das System setzt sich aus mehreren Komponenten zusammen:

- GUI-Client (Abbildung 9)
- Setup-Tool (Abbildung 9)
- Central-Server (Abbildung 10)
- Satelliten-Client (Abbildung 11, Abbildung 12)

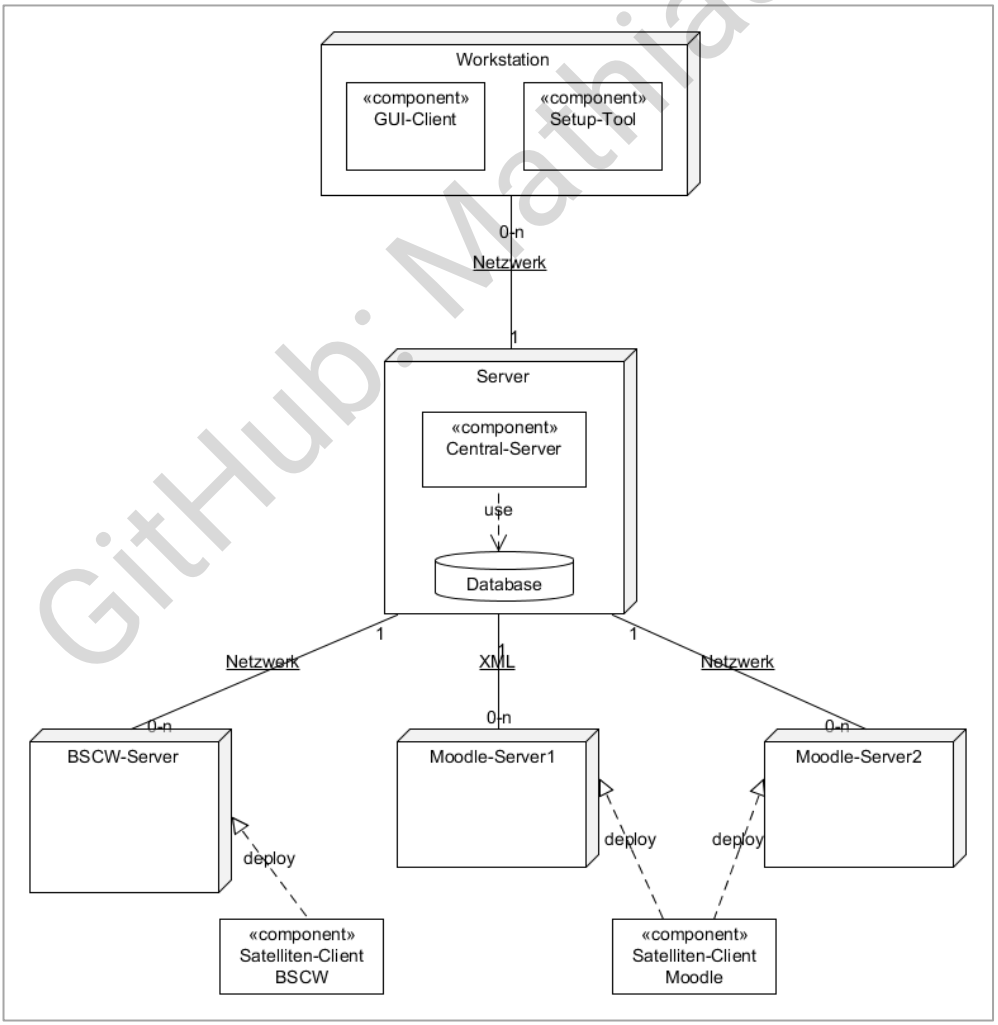

Abbildung 8: Komponentendiagramm Gesamtsystem

In dem Gesamtsystem kann es mehrere Workstations und Satelliten geben (in Abbildung 8 Moodle-Server1, Moodle-Server2, BSCW-Server). Die betrachteten Workstations bekommen alle den gleichen GUI-Client und das gleiche Setup-Tool. Die Satelliten-Systeme unterscheiden sich bezüglich der angesprochenen Services, daher variieren die Satelliten-Clients wie in Abbildung 8 (und später in Abbildung 12) zu sehen. Server mit gleichen Services können unterschiedliche Instanzen des gleichen Satelliten-Clients verwenden.

4.1.1 Workstation

Auf der Workstation sollen der GUI-Client und das Setup-Tool laufen (vgl. Abbildung 9). Der GUI-Client wird durch das Cryptsystem erweitert, welches mit dem auf dem Central-Server eingestellten Cryptsystem übereinstimmen muss. Das Cryptsystem ver- und entschlüsselt die Kommunikation. Für die Einstellungen des GUI-Clients wird eine Konfigurationsdatei eingeladen. Das Setup-Tool wird ebenfalls durch das Cryptsystem erweitert und wird durch eine Konfigurationsdatei konfiguriert.

Das Cryptsystem ist eine Erweiterung für die Verschlüsselung und Entschlüsselung der Konfigurationsdatei und der Netzwerkverbindung.

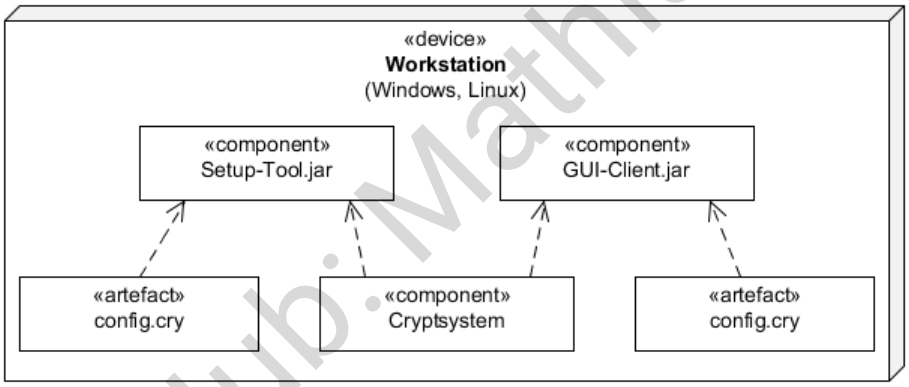

Abbildung 9: Komponentendiagramm Workstation

4.1.2 Central-Server

Der Central-Server wird durch vier Komponenten erweitert. Die erste Komponente ist das Cryptsystem, welches mit dem in den Cryptsystemen auf den GUI-Clients und Satelliten-Clients identisch sein muss. Die Zweite ist der Datenbank-Konnektor, welcher die Verbindung zum DBMS herstellt und somit die Datenbank anbindet. Das Auslagern des Datenbank-Konnektors sorgt für eine bessere Flexibilität in Bezug auf die Wahl des DBMS, da an dem Central-Server selbst keine Änderungen vorgenommen werden müssen. Die dritte Komponente ist das Relationsystem, was die Anwendung zum Finden der Beziehung zwischen natürlicher Person und User unterstützt. Die letzte Komponente ist das Syncsystem, in welches das Verhalten zum Synchronisieren mit dem Satelliten-Client ausgelagert werden kann.

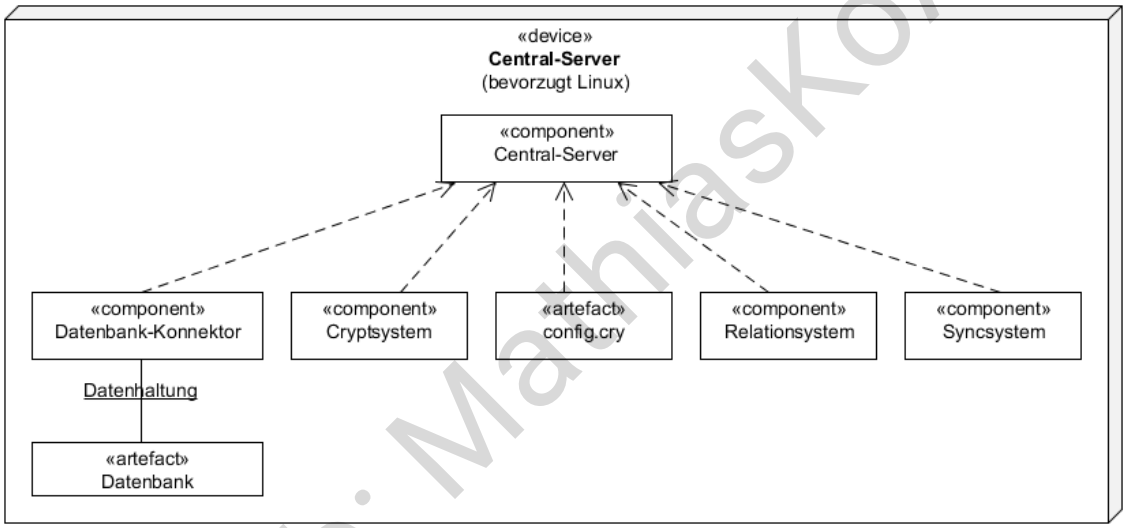

Abbildung 10: Komponentendiagramm Central-Server

4.1.3 Satelliten-System

Der Satelliten-Client des Satelliten-Systems erzeugt die Anbindung an das Dritt-System (Satelliten-Server). Dieser Client ist auf das Anbinden von Services eines bestimmten Typs ausgerichtet (vgl. Abbildung 11, Abbildung 12).

Die Konfiguration erfolgt, wie bei den anderen Komponentenarten, über eine Konfigurationsdatei. Das Verschlüsseln geschieht mittels des Cryptsystem, welches durch den Satelliten-Client eingebunden wird.

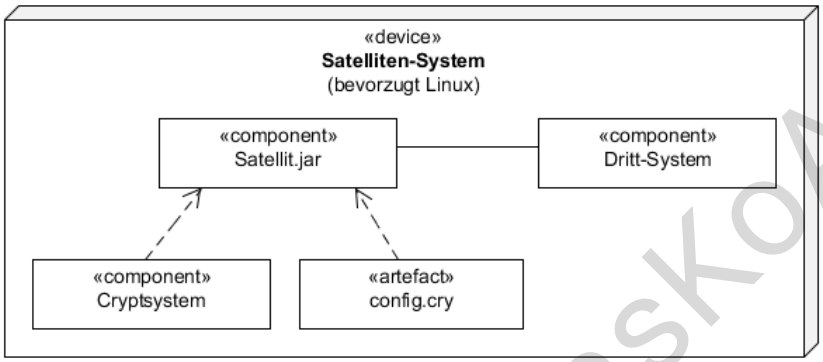

Abbildung 11: Komponentendiagramm Satelliten-System

Ein Satelliten-System besteht aus mindestens einem Dritt-System und einem vollständigen Satelliten-Client. Jede Dritt-System-Instanz benötigt eine Satelliten-Client-Instanz auf demselben Server, unabhängig davon, ob die Satelliten-Client-Instanzen vom selben Typ sind (vgl. Abbildung 11, Abbildung 12).

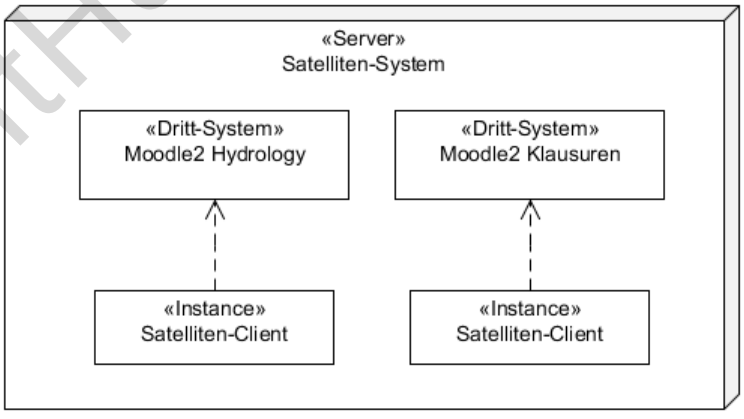

Abbildung 12: Dritt-System und Satelliten-Client

4.2 Einstellungen

Jedes der vier Systeme hat Einstellungen, die geladen werden müssen, bevor der Start des Hauptarbeitszyklus beginnt. In diesem Kapitel werden die Einstellungen konkretisiert und deren Speicherung erläutert.

4.2.1 Central-Server-Setup

Der Central-Server besitzt eine Konfigurationsdatei, welche die Einstellungen der Datenbankverbindung, des Netzwerk-Servers und das Einladen der Plugins festlegt.

4.2.1.1 Plugin Einstellung

- **Relation** ist Plugin, welches für die Bindung von Usern und Personen zuständig ist. (siehe 9. Anforderung 10. Anforderung
- **Synchronisation** bestimmt das Vorgehen bei Synchronisationen mit Satelliten-Systemen.
- **Crypt** ist das Plugin des Cryptsystem, was für die Verschlüsselung der Netzwerkverbindungen sorgt.
- **DB** erzeugt die Anbindung an das DBMS.

4.2.1.2 Datenbank Einstellungen

- **Server-Adresse** ist die Adresse, über die der Datenbank-Server zu erreichen ist. Es wird empfohlen den Datenbank-Server auf demselben Server zu belassen wie die Anwendung des Central-Servers.
- **Datenbank-User** ist der User, mit dem die Datenbank angesprochen werden soll.
- **Datenbank-Name** ist der Name der Datenbank, in der die Daten des Central-Servers gespeichert werden sollen.
- **Datenbank-Passwort** ist das Passwort des Datenbank-Users.

4.2.1.3 Netzwerk Einstellungen

- **Eigene IP-Adresse** ist die Adresse, über die der Central-Server erreichbar sein soll.
- **Server-Socket-Port** ist der Port des Server-Sockets, auf den sich die Clients verbinden sollen.
- **System-Id** ist eine UUID zum eindeutigen Bestimmen des Systems.

4.2.2 GUI-Client-Setup

Der GUI-Client lädt nur ein Plugin und setzt wenige Einstellungen, die für die Anbindung an den Central-Server wichtig sind.

4.2.2.1 Plugin Einstellung

- **Crypt** ist das Plugin des Cryptsystem, welches für die Verschlüsselung der Netzwerkverbindungen sorgt.

4.2.2.2 Netzwerk Einstellungen

- **Central-Server IP-Adresse** ist die IP-Adresse, über die der Central-Server erreichbar sein soll.
- **Server-Socket-Port** ist der Port des Server-Sockets, auf den sich die Clients verbinden sollen.
- **System-Id** ist eine UUID zum eindeutigen Bestimmen des Systems.
- **Central-Server-Id** ist eine UUID zum eindeutigen Bestimmen des Central-Servers.

4.2.2.3 Session Einstellungen

- **Start-Crypt-Key** ist der Schlüssel, mit dem die erste Verbindung zum Central-Server verschlüsselt wird.
- **Sessio-Id-Start** ist die erste Session-Id, die verwendet wird, um mit dem Central-Server in Kontakt zu treten.

4.2.3 Satelliten-Client-Setup

Ähnlich wie der GUI-Client lädt der Satelliten-Client nur ein Plugin und setzt wenige Einstellungen.

4.2.3.1 Plugin Einstellungen

- **Crypt** ist das Plugin des Cryptsystem, welches für die Verschlüsselung der Netzwerkverbindungen sorgt.

4.2.3.2 Netzwerk Einstellungen

- **Eigene IP-Adresse** ist die IP-Adresse, über die der Netzwerk-Server des Satelliten-Clients erreichbar sein soll.
- **Central-Server IP-Adresse** ist die IP-Adresse, über die der Central-Server die Verbindung aufbauen wird.
- **Server-Socket-Port** ist der Socket-Port des Netzwerk-Servers-Sockets, auf den sich der Central-Server verbindet.
- **System-Id** ist eine UUID zum eindeutigen Bestimmen des Systems.
- **Central-Server-Id** ist eine UUID zum eindeutigen Bestimmen des Central-Servers.

4.2.3.3 Session Einstellungen

- **Start-Crypt-Key** ist der Schlüssel, mit dem die erste Verbindung vom Central-Server verschlüsselt wird.
- **Sessio-Id-Start** ist die erste Session-Id, die verwendet wird, um mit dem Central-Server in Kontakt zu treten.

4.3 Central-Server

Der Server kann in sechs logische Bereiche unterteilt werden, Eingangs- & Ausgangs-Kontrolle, Verarbeitung, Datenmodel, Anbindung der Datenbank, Netzwerk und Session-Verwaltung.

4.3.1 Paket-Struktur

Die Paket-Struktur des Servers trennt sichtbar die logischen Bereiche. Die Struktur des Servers, wie in Abbildung 13 zu sehen ist, hat das Root-Element in CUSMS, welches den Projektnamen trägt, und ab dort acht Pakete mit einer maximalen Tiefe von drei.

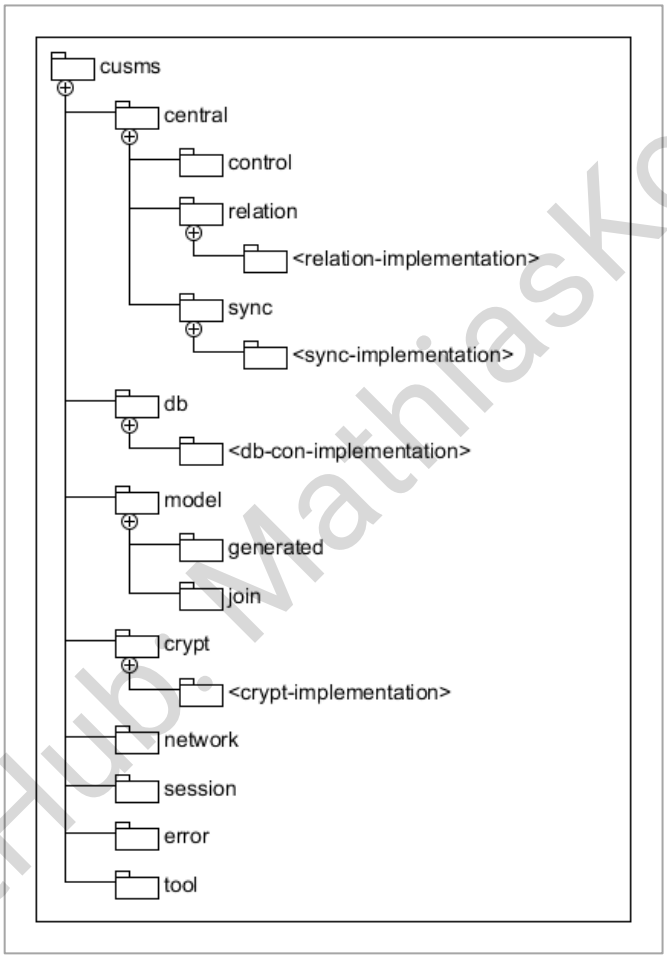

Abbildung 13: Paket-Struktur des Central-Servers

4.3.1.1 Paket „central“

Dieses Paket enthält die Klassen und Unterpakete, die das Vorgehen des Servers bestimmen. Sie decken die logischen Bereiche Eingangs- und Ausgangs-kontrolle und Verarbeitung ab. Die Hauptkontrolle liegt bei den Klassen des Unterpakets `control`.

Die Klassen des Unterpakets **relation** kontrollieren, wie die Vereinheitlichung der User (siehe: 9. Anforderung) und das automatische Verbinden der Personen mit den User abläuft (siehe: 10. Anforderung). Wie in Abbildung 13 zu sehen ist und in der 64. Anforderung gefordert wurde, ist die Erweiterung der Bindung durch neue Algorithmen möglich und erfolgt als Unterpaket von **central.relation**.

Die Klassen in dem Unterpakt **sync** beschreiben das Synchronisieren und Synchronhalten der Daten mit den Satelliten-Systemen, die auch autark ihre Daten verändern dürfen. Aufgrund der 65. Anforderung , welche die Erweiterbarkeit des Gesamtsystems durch neue Satelliten-Clients fordert, ist die Erweiterbarkeit der Synchronisationsarten und Synchronisationsvorgänge ein logischer Schritt. Dies ist als Unterpaket von **central.sync** möglich.

4.3.1.2 Paket „db“

Dieses Paket verbindet die Datenbank mit dem Klassen-System des Central-Servers. Für die Anbindung unterschiedlicher DBMS, wie in der 63. Anforderung beschrieben, liegen die Implementationen der Anteile, die spezifisch für einen DBMS sind, in einem Unterpaket von **db**. Entsprechend sind Erweiterungen als Unterpaket abzulegen.

4.3.1.3 Paket „model“

Das Paket **model** trägt die Klassen für die Datenhaltung während des Verarbeitungsprozesses. In dem Unterpaket **generated** befinden sich die Models, die aus der XSD-Datei erstellt wurden und die Daten tragen, welche über das Netzwerk übertragen werden können. Das Unterpaket **join** hält die Klassen, welche die JOIN-Struktur des Datenbank-Modells widerspiegelt.

4.3.1.4 Paket „crypt“

Das Paket **crypt** kapselt die Klassen, die für das Verschlüsseln nötig sind. Da sich die Sicherheit der Verschlüsselung oder die Anforderungen im Lebenszyklus des Systems ändern können, können weitere Verschlüsselungen in den Unterpaketen von **crypt** hinterlegt werden.

4.3.1.5 Paket „network“

In dem Paket **network** liegen die Klassen, die verantwortlich für das Annehmen von Verbindungen und das Versenden der Netzwerkdaten sind.

4.3.1.6 Paket „session“

In dem Paket **session** befinden sich die Klassen, deren Aufgabe die Verwaltung und die Haltung der aktiven Sessions sind. Sessions tragen die Information der Verschlüsselung, der aktiven Verbindung, des verbundenen Users und des verbundenen Systems.

4.3.1.7 Paket „error“

In dem Paket **error** sind alle spezifischen Errors und Exceptions für den Central-Server aufgeführt.

4.3.1.8 Palet „tool“

Im dem Paket **tool** sind die Klassen hinterlegt, die kontextunabhängig durch alle Pakete genutzt werden.

4.3.2 Ordner-Struktur

Aus der Paket-Struktur aus Kapitel 4.3.1 ergibt sich folgende Ordner-Struktur, in welche Einstellungen und die Plugins abzulegen sind.

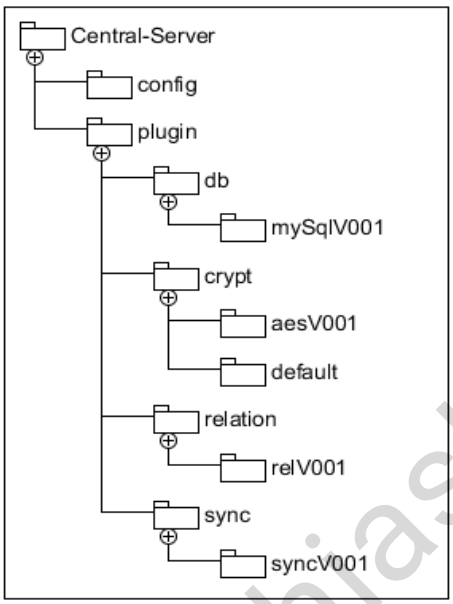

Abbildung 14: Ordner-Struktur des Central-Servers

Die Einstellungen werden in `config` abgelegt und tragen den Namen setup.cry.

Damit die Einstellungen entschlüsselt werden können, muss bereits ein Crypt-Plugin vorhanden sein. Dieses Crypt-Plugin muss unter `plugin/crypt/default` liegen und wird vor dem Entschlüsseln der Einstellungen geladen.

Alle Plugins, die eingeladen werden sollen, müssen in dem Ordner Plugin und ihrer Paket-Struktur, als Class-Dateien liegen. In den Einstellungen sind die konkreten Paketnamen der Plugins aufgeführt. Die Klassennamen der Klassen, welche die Interfaces implementieren, müssen den Namen des Interfaces abzüglich des vorangestellten „I“s tragen.

4.3.3 Klassenmodell

Das Klassenmodell kann aufgrund der Größe nicht im Ganzen korrekt dargestellt werden. Deshalb werden in den folgenden Unterkapiteln die Klassendiagramme mit Beschränkung auf ihre Pakete dargestellt.

4.3.3.1 Klassenmodell „central.control“

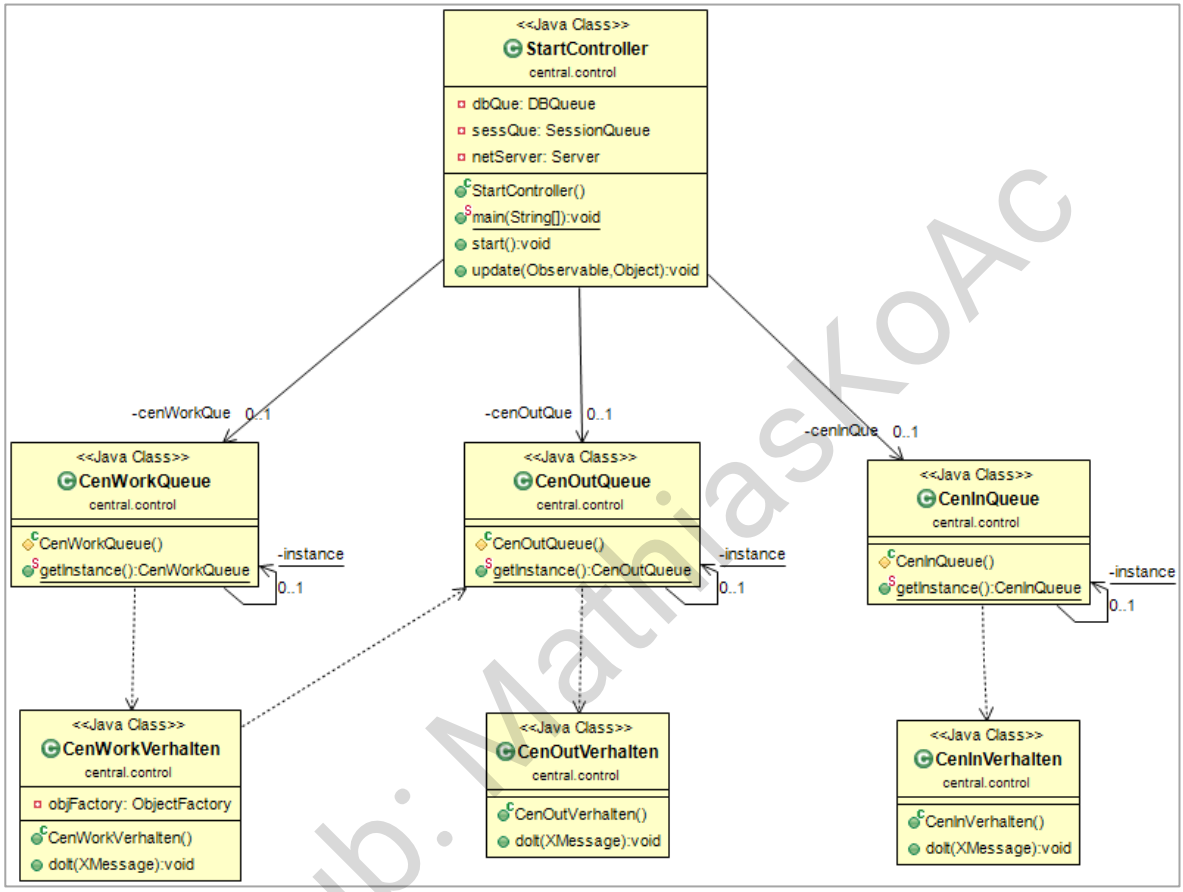

Abbildung 15: Klassendiagramm central.control

Das Paket **`central.control`** (vgl. Abbildung 15) ist die Hauptinstanz des Systems. Diese Kontrolle teilt sich auf den **`StartController`** und die Kombination aus Queue und Verhalten von **`CenWork`**, **`CenOut`** und **`CenIn`** auf. Die Queue-Klassen sorgen dafür, dass die Aufträge in Form einer Warteschlange vorgehalten werden und arbeiten die diese so ab, wie es im Verhalten definiert ist. (mehr Informationen über die Queue-Verhalten-Paarung in Kapitel 4.3.4.1 Queue-Modell)

- **`StartController`** startet alle Threads und wird zum **`Timethread`**, der zeitgesteuerte Events auslöst.
- **`CenWorkQueue`** und **`CenWorkVerhalten`** steuern das Aufbereiten der Daten nach einer Abfrage in der Datenbank.
- **`CenOutQueue`** und **`CenOutVerhalten`** steuern das Absenden der **`Messages`**.
- **`CenInQueue`** und **`CenInVerhalten`** steuern das Eingehen und Weiterleiten der **`Messages`** in andere Queues.

4.3.3.2 Klassenmodell „central.sync“ und „central.relation“

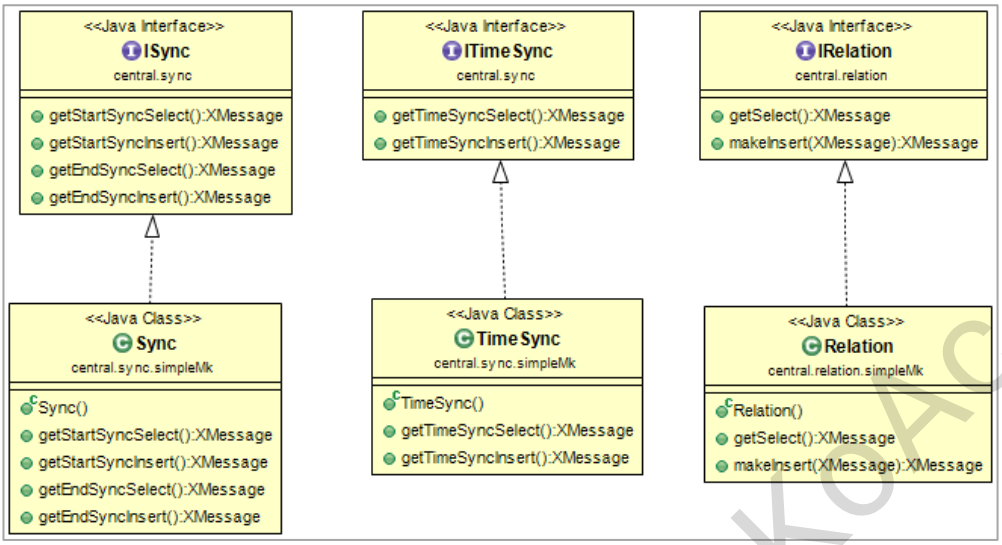

Abbildung 16: Klassendiagramm central.sync und central.relation

Die Pakete **`central.sync`** und **`central.relation`** sind Pakete, die selbst nur Interfaces halten (vgl. Abbildung 16). Die Implementationen gehören in Unter-Paket.

Alle drei Klassen arbeiten intern, wie ein Iterator, der komplette **`Commands`** in der **`XMessage`** herausgibt, bis ein kompletter Durchlauf getätigt wurde.

Das Paket **`relation`** hält nur Interfaces, da die Interpretation, nach welchen Kriterien die Relation von Person und User entstehen soll, unterschiedlich sein kann. Des Weiteren ist anzunehmen, dass sich in die Kriterien in dem Lebenszyklus der Software noch verfeinern werden.

Das Paket **`sync`** hält nur Interfaces, da pro Satelliten-Client die Art der Synchronisation und welche Datenbereiche genau abgeglichen werden stark schwanken können.

Es wird zwischen **`Sync`** und **`TimeSync`** unterschieden. **`TimeSync`** wird, wie der Name schon andeutet, zeitgesteuert verwendet. **`Sync`** wird, wenn dieses Plugin gemappt und aktiviert ist, für den Abgleich von **`Commands`** in Richtung des Satelliten-Clients verwendet.

4.3.3.3 Klassenmodell „crypt“

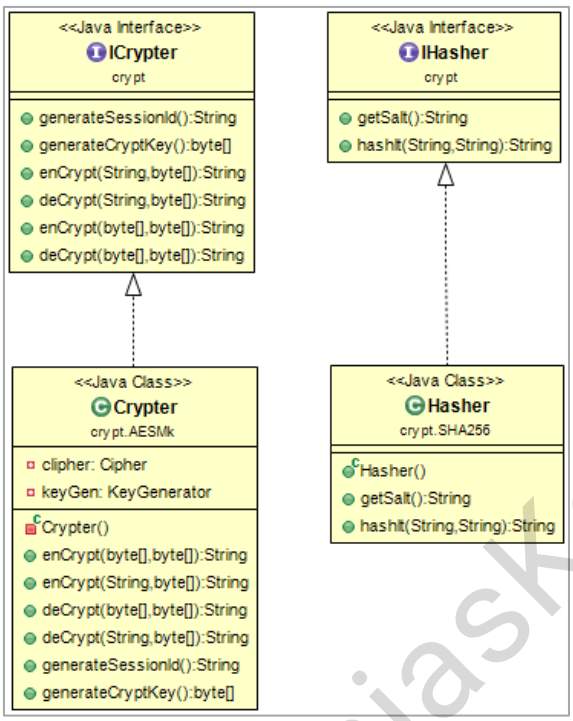

Abbildung 17: Klassendiagramm crypt

In dem Paket **`crypt`** findet man das Interface **`ICrypter`** und **`IHasher`**. An dieser Stelle kommt ein Interface zum Einsatz, um die Möglichkeit zu bieten andere Verschlüsselungsmethoden, einen anderen Generator für **`CryptKeys`**, **`SessionIds`** oder Hashes einzusetzen (vgl. Abbildung 17).

- **`ICrypter`** und die Implementation Crypter ver- und entschlüsseln symmetrisch.

- **`IHasher`** und die Implementation Hasher hashen Strings und geben feste oder generierte „Salt-Werte“ zurück.

4.3.3.4 Klassenmodell „db“

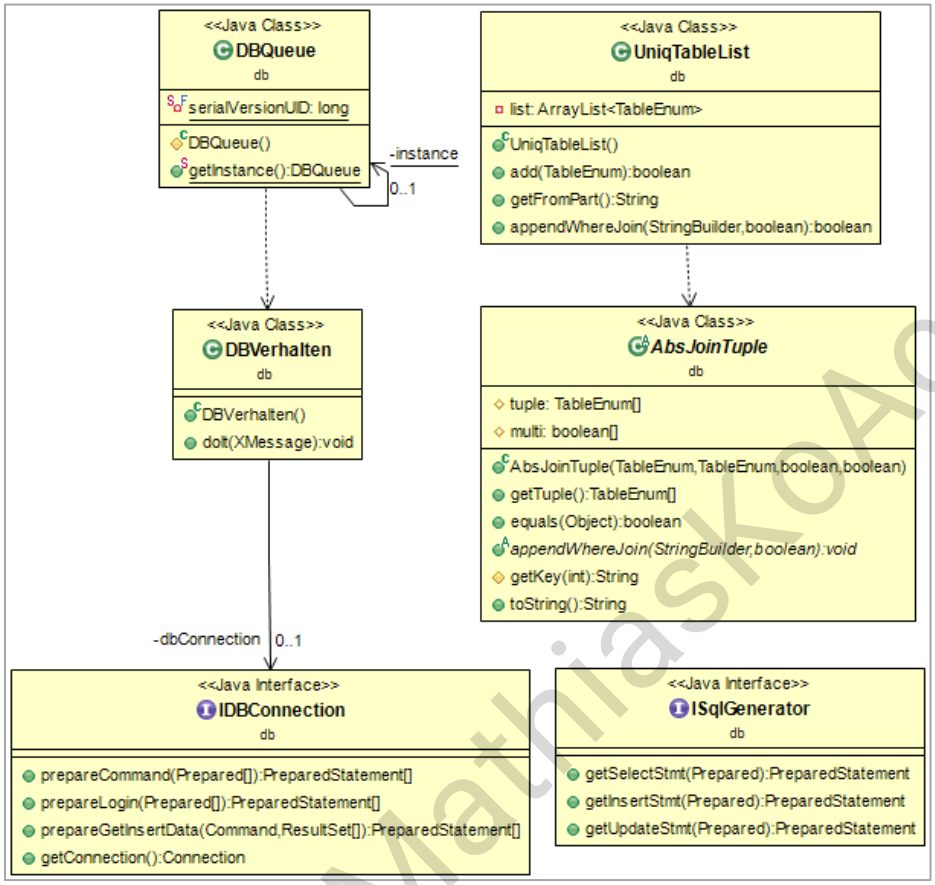

Abbildung 18: Klassendiagramm db

In dem Paket **db** ist die Kontrolle dieses logischen Bereichs in Form von **DBQueue** und **DBVerhalten** zu finden. Zudem ist die Vorbereitung auf die Datenbankanbindung und das Generieren des SQL passend zum gewünschten DBMS durch die beiden Interfaces, **IDBConnection**, **ISqlGenerator** und das abstrakte **JoinTuple** (vgl. Abbildung 18) zu finden.

4.3.3.5 Klassenmodell „db.mysql“

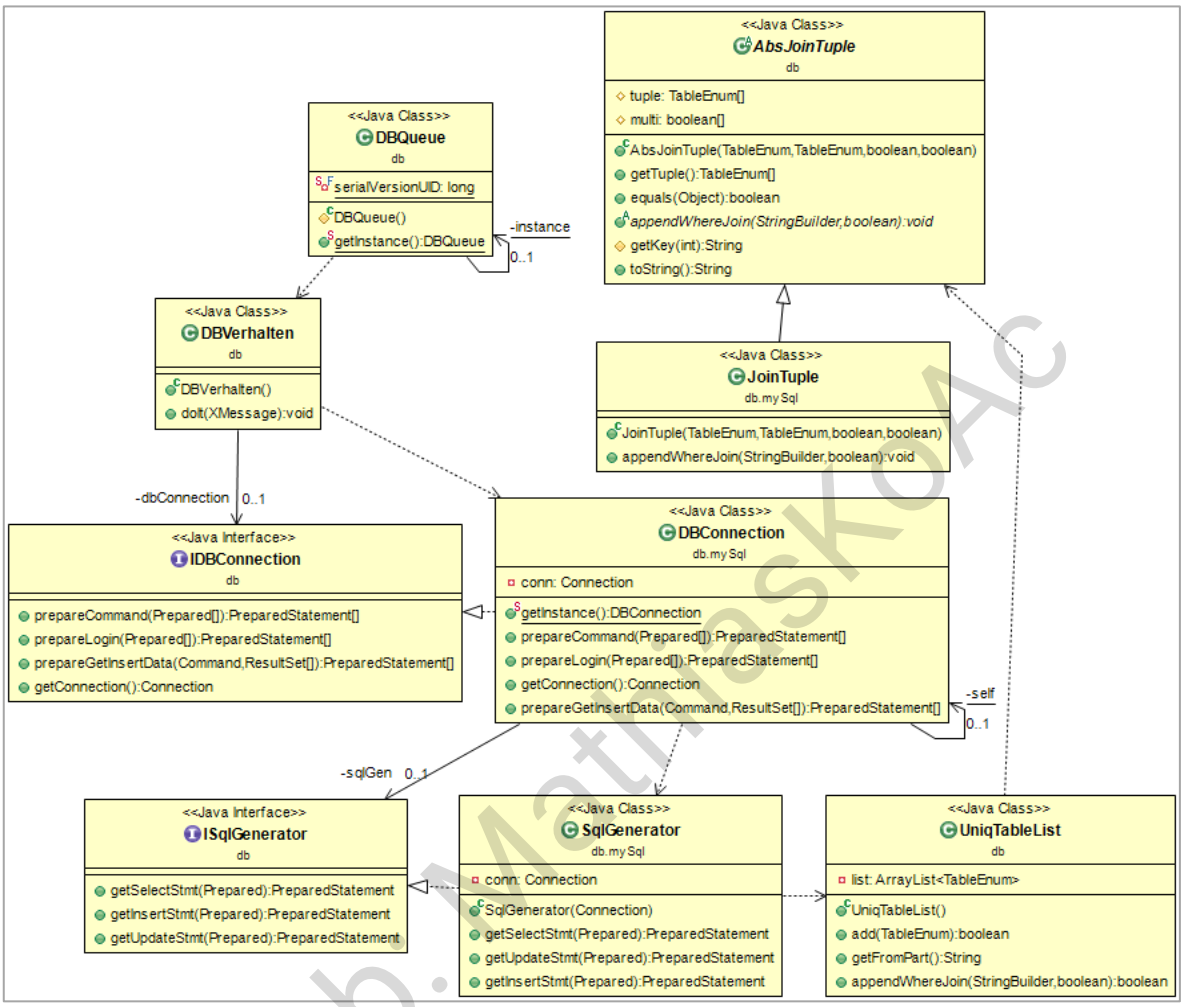

Abbildung 19: Klassendiagramm db.mysql

Dieses Paket **`db.mysql`** soll als Beispielimplementation für die zukünftigen Implementation der Anbindungen an DBMS dienen und in der Form, für die Anbindung von dem DBMS-MySQL, eingesetzt werden. Anhand der Implementation sieht man stärker wie, das Anbinden eines DBMS gedacht ist.

- Die **`DBConnection`** hält die Verbindung zur Datenbank und steuert zusammen mit dem **`DBVerhalten`** das Aufbauen und Absenden des **`PreparedStatements`**.

- Der **`SqlGenerator`** erstellt das **`PreparedStatement`** aus dem **`Prepared`** und kontrolliert das Erstellen der Bestandteile.

- Das **`JoinTuple`** sammelt die zusammengehörigen Tabellen für die Joins und erstellt in dieser Variante ein WhereJoin.

4.3.3.6 Klassenmodell „model“

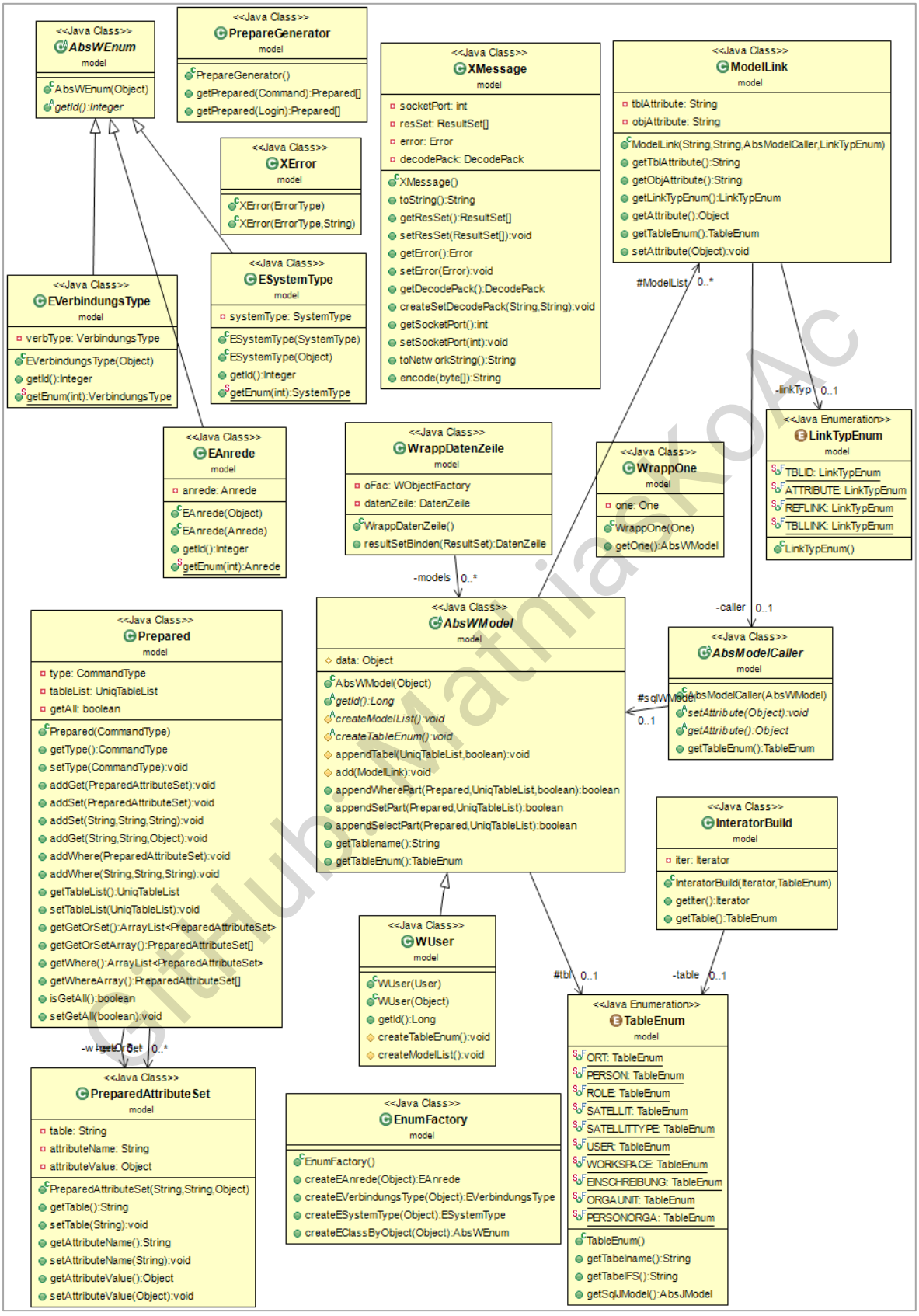

Abbildung 20: Klassendiagramm model reduziert

Abbildung 20 zeigt das Klassendiagramm des Pakets `model` ohne Unterpaket und in reduzierter Form, da einige `WModels`, die `WObjektFactory,` ausgespart wurden und das Diagramm keine Abhängigkeiten (dependencies) anzeigt, da es sonst noch unleserlicher wäre.

- **XMessage** ist von `generated.Message` abgeleitet und bildet das Hauptmodell, welches über die Queues verteilt wird. Da es das Hauptmodell ist, wurde es um die Attribute `socketPort`, `resSet` (`ResultSet`), `error` und `decodePack` erweitert.
- **XError** ist von `generated.Error` abgeleitet und bietet zusätzliche Konstruktoren.
- **PrepareGenerator** erstellt `Prepareds` aus `Logins` und `Commands.`
- **Prepared** ist ein Model, welches einfach in ein `PreparedStatemtent` umgesetzt werden kann und zur Kapselung von Daten und DBMS beiträgt.
- **PreparedAttributeSet** ist eine Klasse, deren Objekt genau ein Attribut innerhalb einer Datenbanktabelle darstellt.
- **AbsWEnum** ist eine abstrakte Wrapper-Klasse für Enums, damit die Implementationen dieser Klasse die Enums aus `model.generated` erweitern können.
- **EVerbindungsType** ist eine Implementation der Klasse `AbsWEnum` und adaptiert das Enum `VerbindungsType`.
- **ESystemType** implementiert die Klasse `AbsWEnum` und adaptiert das Enum `SystemType`
- **EAnrede** ist eine Implementation der Klasse `AbsWEnum` und adaptiert das Enum `AnredeType`.
- **WrappDatenZeile** adaptiert die Klasse `DatenZeile,` um die Methode `resultSetBinden()` zu implementieren, die ein `ResultSet` in die ummantelte `DatenZeile` einträgt.
- **EnumFactory** ist eine Factory-Klasse für Enum-Wrapper.
- **ModelLink** ist eine Klasse, die Verbindung von `AbsWModel` und `AbsModelCaller` aufbaut, um den entkoppelten Zugriff von `AbsWModel` auf ein Attribut eines `Daten-Model` zu ermöglichen.
- **AbsWModel** ist eine abstrakte Wrapper Klasse für `Daten-Models`, mit Methoden zur Erstellung der `Prepards`.
- **LinkTypEnum** ist ein Enum zur Unterscheidung ob es sich um eine Tabellen-Id, ein Attribut, ein Link zu einer Referenztabelle oder ein Link zu einer nRt handelt.
- **AbsModelCaller** ist eine abstrakte Klasse zur Deklaration der Methoden `setAttribute()` und `getAttribute()`.
- **WrappOne** ist eine Wrapper-Klasse der Klasse `One` um eine Methode anzuhängen, die ein `WModel` aus einem `One`-Objekt extrahiert.
- **WUser** ist eine `WModle`-Klasse des `Daten-Models` `User`, welche in Abbildung 20 als Repräsentant aller `WModel` Klassen abgebildet ist.

- **InteratorBuild** ist eine Hilfsklasse zum Verbinden von `TableEnums` und `Iteratoren`, um über Tabellen zu iterieren.
- **TableEnum** ist ein Enum mit je einem Wert aller nRt und ein FactoryEnum, welches nach dem DesignPattern Factory `JModels`, abhängig von dem einen State, erstellt.

4.3.3.7 Klassenmodell „model.generated“

Die Klassen im Paket **`model.generated`** sind, mit Hilfe des Jaxb aus der XSD-Datei für die Kommunikation erstellt worden (mehr Information zur XSD-Datei im Anhang 8.1). Die Klassen halten nur Daten und haben Get- und Set-Methoden. Das Klassenmodell in diesem Paket lässt sich grob in zwei Teile teilen. Zum einen der Kontroll-Teil (siehe Abbildung 21) und zum anderen der Daten-Teil (siehe Abbildung 23).

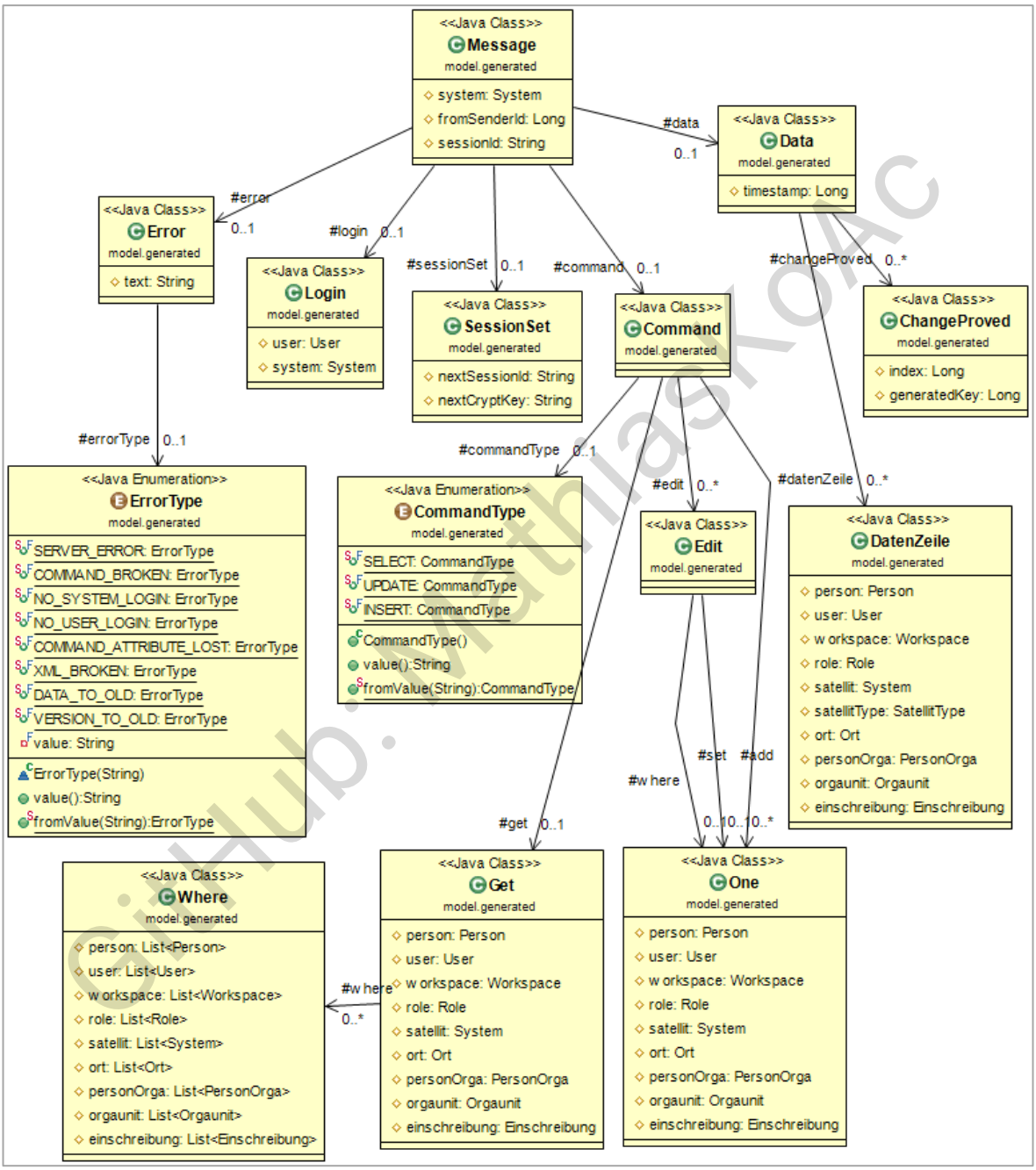

Abbildung 21: Klassendiagramm model.generated Kontroll-Teil

In Abbildung 21 sieht man den Kontroll-Teil des Klassen-Models. Dieser Teil ist der Kontroll-Teil, weil hier definiert wird, welche Art von Message gesendet wird. Dieser Bereich lässt sich noch mal in zwei Bereiche trennen, zum einen in den Session-Kontroll-Bereich und zum anderen in den Daten-Kontroll-Bereich.

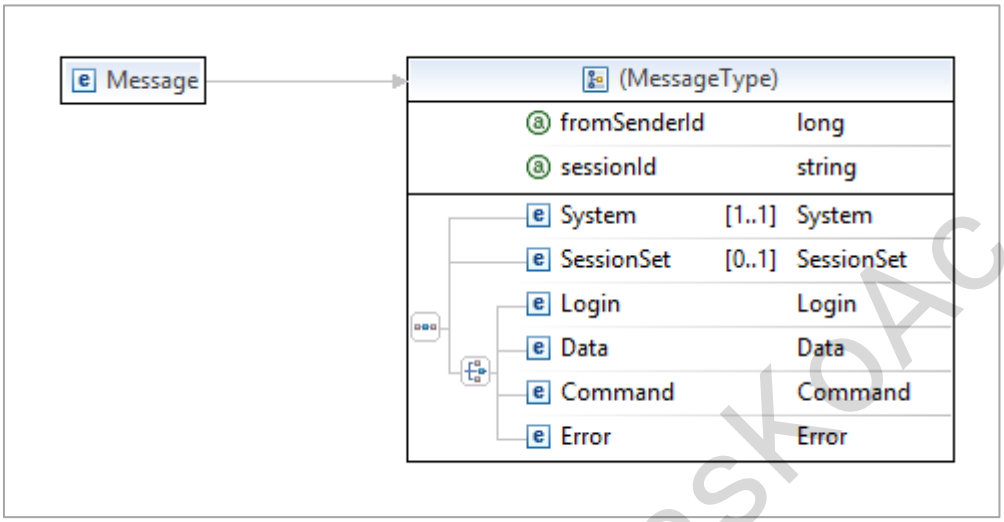

Abbildung 22: XSD-Message

In Abbildung 22 ist zu sehen, dass die Oberste Ebene die **`Message`** von Type **`MessageType`** aus den Attributen **`fromSenderId`**, **`sessionId`** sowie der Sequenz[1] aus **`System`**, **`SessionSet`** und einem Choice[2] besteht. Die **`Message`** bis zum Choice beschreibt den Session-Kontroll-Bereich (in Abbildung 21 beschrieben durch die Klasse **`Message`** und **`SessionSet`**), der dazu da ist, die aktuelle Verbindung zu sichern.

- **`FromSenderId`** ist die Identifikation des sendenden Systems. Diese Id wird beim Erstellen der Einstellungen für das System erzeugt.
- **`SessionId`** ist die Identifikation der Session, die für jede Paarung aus Hin- und Rückkommunikation zum Central-Server eindeutig ist.
- **`System`** beschreibt das empfangende System.
- **`SessionSet`** beinhaltet die Session-Information für die nächste Session.

Mit dem Choice im Abbildung 22 setzt der Daten-Kontroll-Bereich ein. Zunächst mit der Entscheidung (Choice), ob es sich um ein **`Login`**, eines **`Systems`** oder **`Users`**, reiner Daten, einem **`Command`** oder einem **`Error`** handelt.

[1] Sequenz-Symbol

[2] Choice-Symbol 

Der Login ist nicht tief verzweigt. Es gibt den **Login** für das **System** und den **Login** für den **User** (vgl. Abbildung 56 im Anhang). Die Klassen, die Daten für den Login-User oder das Login-System halten, sind dieselben, die später im Daten-Teil verwendet werden.

Die **Data**-Klasse hält die Daten nach einer Abfrage, Eingabe- oder einem Editiervorgang (vgl. Abbildung 57 im Anhang). Wenn **Data** Daten als Ergebnis einer Abfrage hält, wird die **DatenZeile** verwendet. Wenn es Daten eines Eingabe- oder eines Editiervorgangs hält, wird **ChangeProved** verwendet. **ChangeProved** hält den Index der Manipulation und die Id des entstandenen oder manipulierten Eintrags, während **DatenZeile** eine objektorientierte Entsprechung eines **ResultSets** darstellt.

Der **Command** beschreibt ein Kommando ähnlich einem SQL-Statement der DML (Data Manipulation Language). Das Command, hat einen **Command-Type** und abhängig von dem Type wird ein **Add** (als **One**), ein **Edit** oder ein **Get** formuliert (vgl. Abbildung 59 im Anhang). Ein **Add** besteht nur aus dem Auftrag etwas hinzuzufügen und benötigt keine Where-Clause. Ein **Get** oder ein **Edit** benötigen eine Where-Clause. Das **Get** hat die Where-Clause in Form einer **Where**-Klasse und das **Edit** in reduzierter Form, als zweite **One**-Klasse.

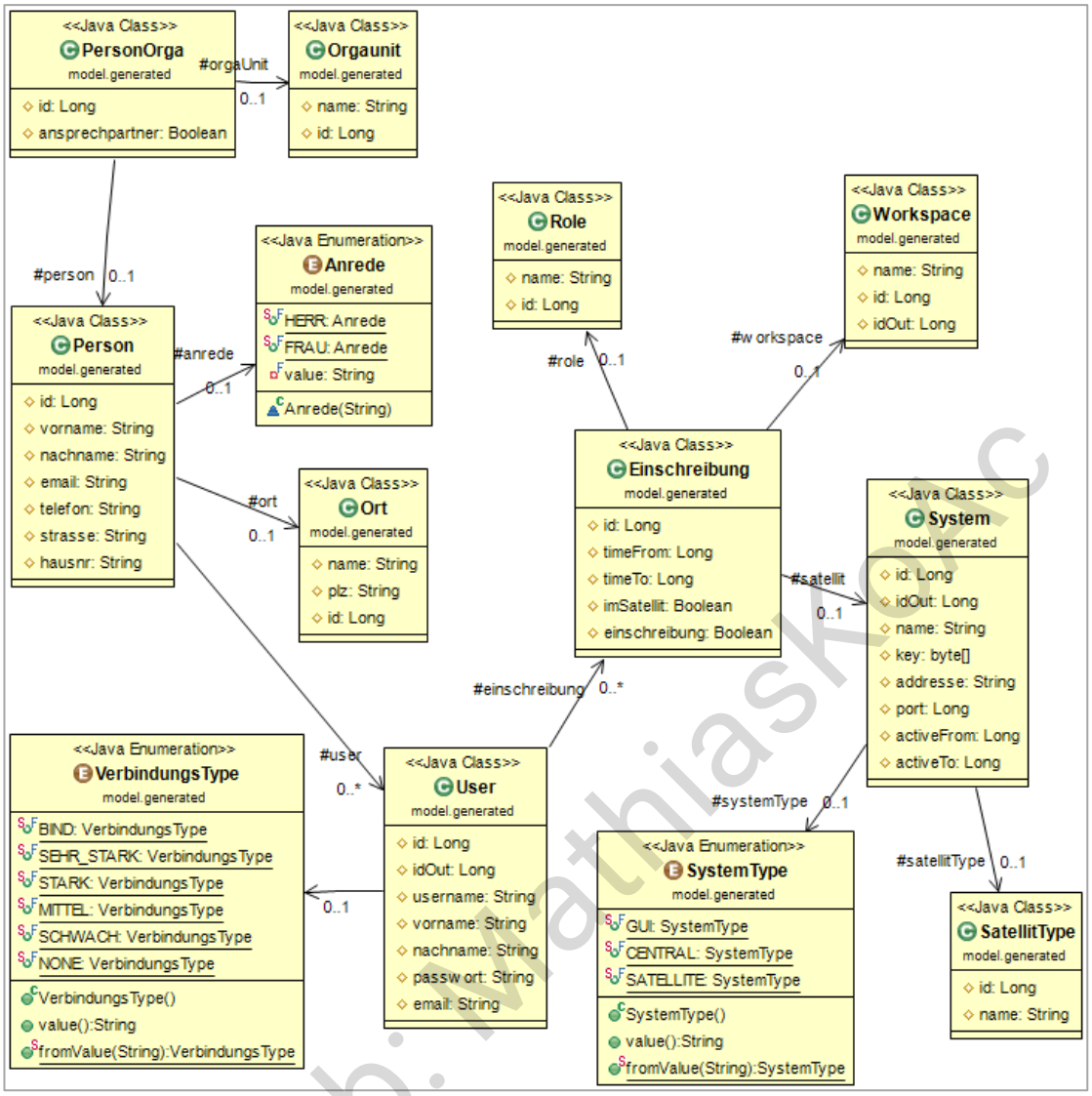

Abbildung 23: Klassendiagramm model.generated Daten-Teil

Der Daten-Teil des generierten Modells entspricht dem Datenbank-Modell des Systems. Es ermöglicht die unterschiedlichen Varianten des Einschreibens in ein System, das Binden von User und Person, sowie das Eintragen in OUs.

4.3.3.8 Klassenmodell „model.join“

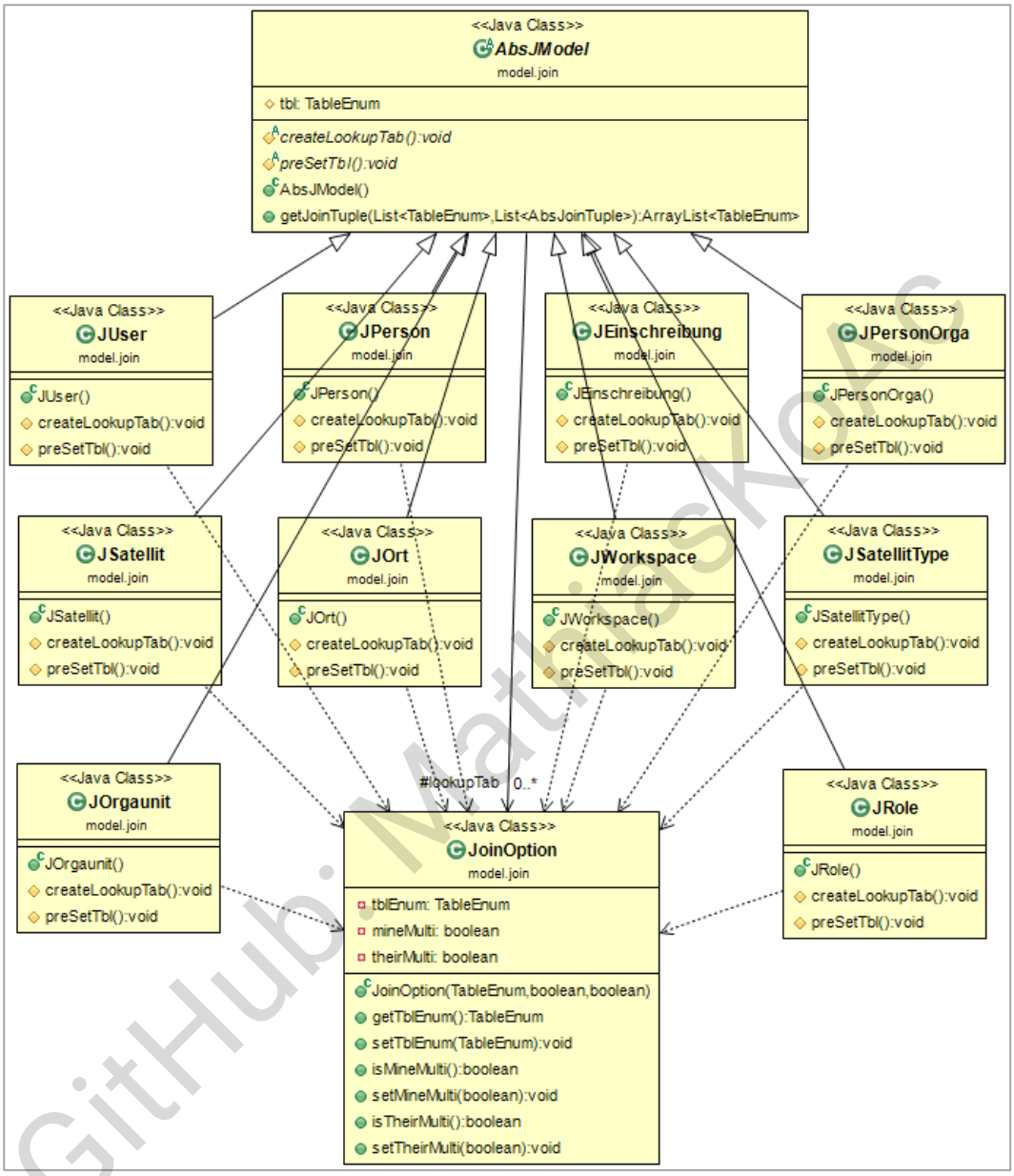

Abbildung 24: Klassendiagramm JModel

In dem Paket **JModel** befindet sich die abstrakte Klasse **AbsJModel**, welche die Hauptintelligenz des **JModel** bzw. **Link-Models** hält. **JoinOption** ist die Klasse über, welche die Verbindung zwischen den **JModels** geschieht. Die Implementierungen des **AbsJModel**, ist passend zu den Tabellen des DB-Models um die Verbindung aufzubauen (mehr Information zum **JModel** bzw. **Link-Model** in Kapitel 4.3.5.4 Link-Model).

4.3.3.9 Klassenmodell „network“

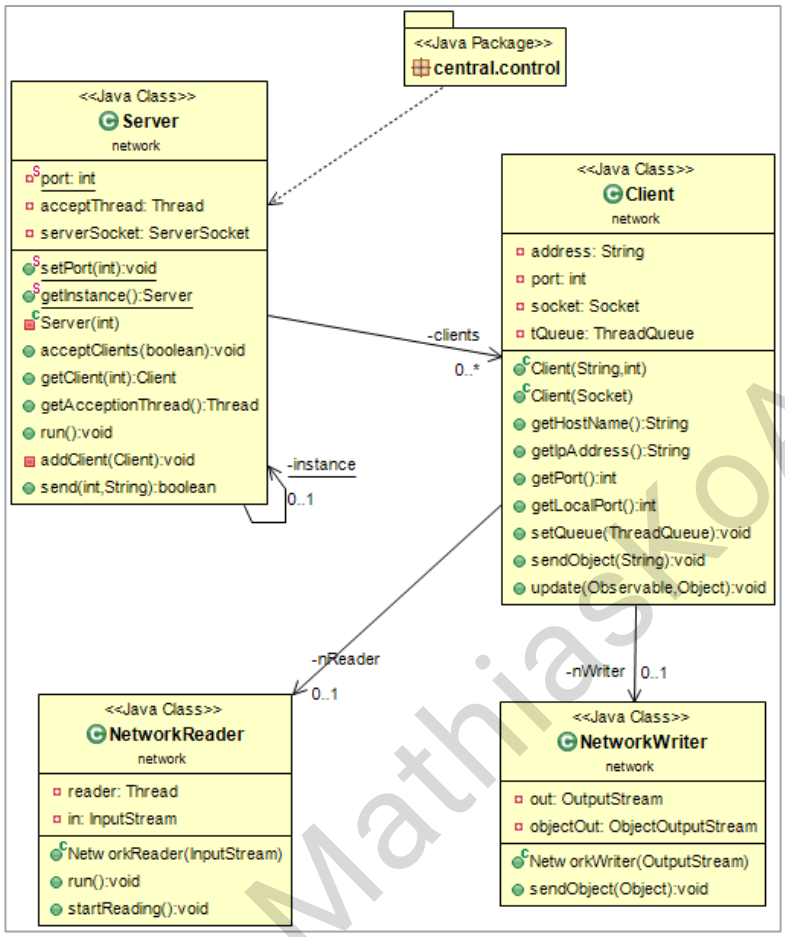

Abbildung 25: Klassendiagramm network

In dem Paket **network** ist der Socket-Server und die Verarbeitung der Socket-Verbindungen gekapselt. Der hier verwendete Netzwerk-Server stellt eine überarbeitete Version des im SWE-Projekt „Rundenbasiertes Netzwerkspiel“ alias „Square-Commander“ verwendeten Netzwerk-Servers dar. (vgl. [7]).

4.3.3.10 Klassenmodell „session“

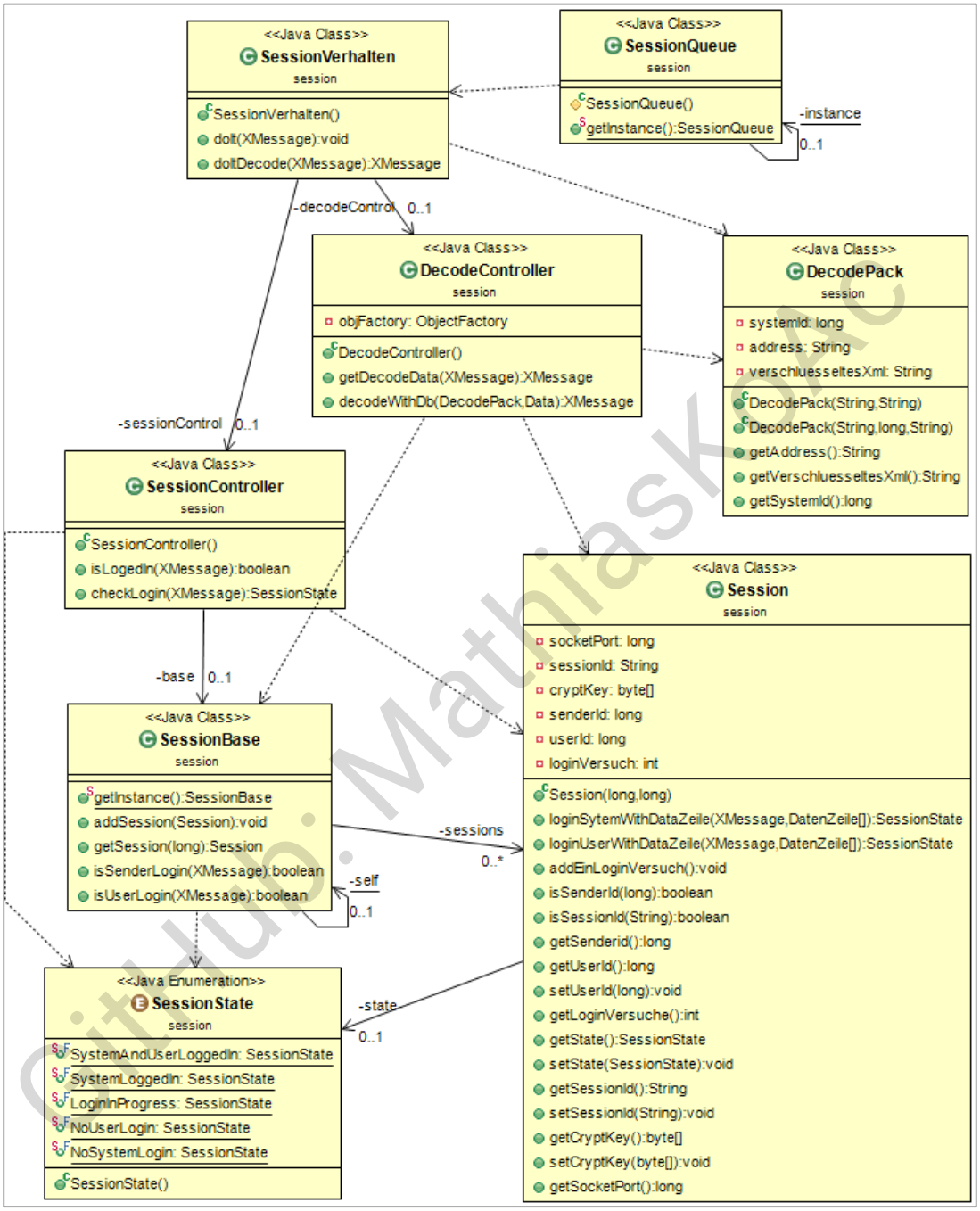

Abbildung 26: Klassendiagramm session

In dem Paket **`session`** befindet sich sowohl die Entschlüsselung als auch die Verwaltung der **`Sessions`**. Der Eingang in diese Verwaltung geschieht über die **`SessionQueue`**, welche die weitere Kontrolle dem **`SessionController`** bzw. **`DecodeController`** übergibt.

- **SessionQueue** und **SessionVerhalten** steuern das Prüfen der `Session` und dekodieren des verschlüsselten XMLs.
- Der **SessionController** prüft die `Session` der `Message` und steuert das Einloggen eines Systems oder Users.
- Die **SessionBase** hält zur Laufzeit die aktiven `Sessions` und verifiziert Logins.
- Der **SessionState** ist ein Enum, das den konkreten Status der `Session` angibt.
- Die **Session** ist die Daten-Klasse, welche die Informationen der aktiven `Session` zur Laufzeit speichert.
- Der **DecodeController** decodiert mit Hilfe der `Session` und dem `Crypter` das verschlüsselte XML.
- Das **DecodePack** hält die Informationen über die `Message` vor der Decodierung, darunter auch das verschlüsselte XML. Es wird über eine `XMessage` an die `SessionQueue` geliefert.

4.3.3.11 Klassenmodell „tool“

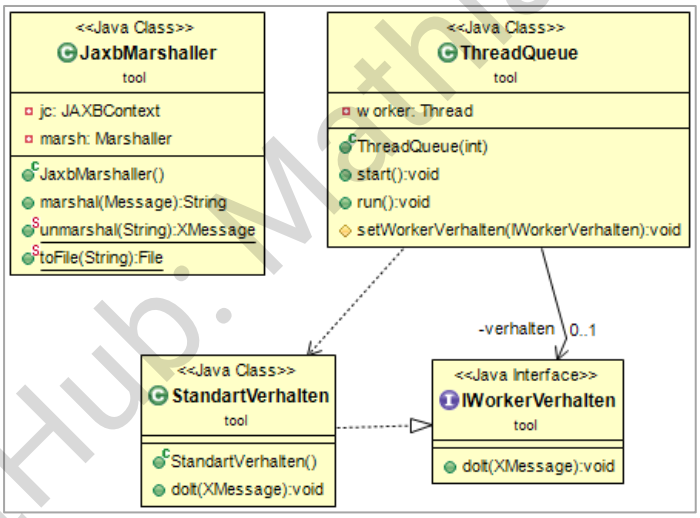

Abbildung 27: Klassendiagramm tool

Im Paket `tool` befinden sich diejenigen Klassen, die durch mehre Paket genutzt werden. Zum einen der `JaxbMarshaller` der für das un- und marshallen zuständig ist und zum anderen die Basis des `Queue-Models`, welches weiter in dem Kapitel 4.3.4.1 Queue-Modell beschrieben wird.

4.3.4 Parallelität

Aus der Teilung in logisch getrennte Bereiche lassen sich unabhängige Funktionen ableiten. Diese Funktionen laufen mit der Hilfe von Queues in diesem System parallel, um Wartezeiten und Deadlocks zu vermeiden. Jeder Bereich, der unabhängige Funktionen durchführt, hat eine eigene Queue und ein dazugehöriges Verhalten mit Ausnahme des `Netzwerk-Server` und des `StartControllers`.

4.3.4.1 Queue-Modell

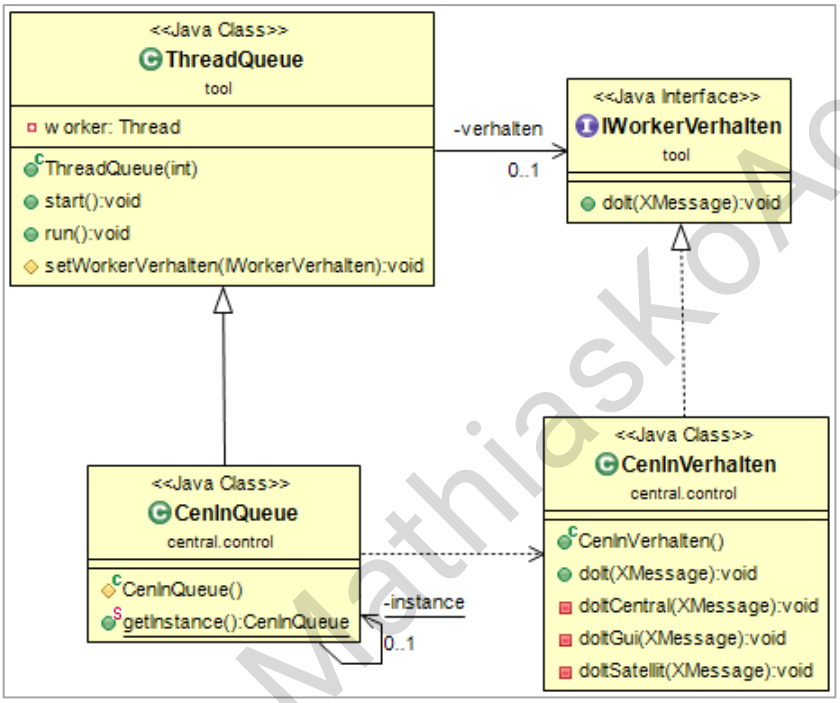

Abbildung 28: Klassendiagramm Queues und Verhalten

Um die Queues und die Verhalten in dem System zu vereinheitlichen, leitet sich jede Queue in dem System von `ThreadQueue` ab. `ThreadQueue` implementiert das Interface `Runnable` und leitet von der Klasse `LinkedBlockingQueue` ab. Die Queue führt Objekte von Typ `XMessage` und wird durch den Queue eigenen Thread mit dem entsprechenden Verhalten der Queue abgearbeitet.

Das Verhalten wird durch eine Klasse vorgegeben, die von `IWorkerVerhalten` abgeleitet und der Queue zugewiesen wurde. Die Methode, die durch `IWorkerVerhalten` festgelegt ist, trägt den Namen `doIt()` und bekommt Objekte vom Typ `XMessage` übergeben.

Um in dem System nur eine Queue pro logischen Bereich zu haben, wurde das Design Pattern Singleton angewandt (vgl. [8] S.180 - 183). Dadurch, dass die Methode `getInstance()` zusätzlich durch das „zweifach geprüfte Sperren“ ([8] S.182) gesichert ist, wird verhindert, dass mehr als eine Instanz dieser Klasse erzeugt wird.

Abbildung 29 zeigt den Quellcode für das „zweifach geprüfte Sperren“, das die Nachteile des Sychronized-Bereiches minimiert, indem dieser nur durchlaufen wird, wenn die Instanz der Klasse `null` ist. Sychronized-Bereiche sind wenn möglich zu vermeiden, da dort Threads aufeinander warten, diesen Bereich zu abzuarbeiten.

```
public class CenInQueue extends ThreadQueue{

    private volatile static CenInQueue instance;
    …
    public static CenInQueue getInstance() {
        if(instance == null){
            synchronized (CenInQueue.class) {
                if(instance == null){
                    instance = new CenInQueue();
                    instance.setWorkerVerhalten(new CenInVerhalten());
                  }
            }
        }
        return instance;
    }
}
```

Abbildung 29: Quelltext für zweifach geprüftes Sperren

Das Schlüsselwort **volatile** heißt wörtlich übersetzt „flüchtig“, „beweglich“ oder „gasförmig“. In diesem Zusammenhang sorgt es für das richtige Verhalten der Threads, in dem es das Attribut als threadübergreifend einzigartig vermerkt (vgl. [8] S.182f). Somit wird unter anderem das Zwischenspeichern der Variable in den Threads sowie das gleichzeitige Schreiben auf den Speicher unterdrückt (vgl. [9]).

In der Methode **getInstance()** sorgen alle Queues selbstständig dafür, dass sie sich das richtige Verhalten zuweisen. Somit entsteht immer eine feste Paarung aus Queue und Verhalten, wie in Abbildung 28 beispielhaft an **CenInQueue** und **CenInVerhalten** zu sehen ist.

Nach dem beschriebenen Muster aus **Queue** und **Verhalten** entstehen so fünf Paarungen, die sich gegenseitig die Aufträge in Form von Objekten des Typs **XMessage** zuteilen (vgl. Abbildung 30).

- **CenInQueue** **CenInVerhalten**
- **CenOutQueue** **CenOutVerhalten**
- **CenWorkQueue** **CenWorkVerhalten**
- **DBQueue** **DBVerhalten**
- **SessionQueue** **SessionVerhalten**

Alle Queues werden zum Start des Systems durch den **StartController** erzeugt und gestartet. Nach dem Start der Threads, der Queues und des Netzwerk-Servers (Klasse **Server** aus Paket **network**) wird der Thread des **StartControllers** zum **TimeThread**. Dieser **TimeThread** startet selbständig die eingestellte, zeitgesteuerte Synchronisierung (vgl. Anforderung unter Abschnitt 2.3.1.2 Zeitgesteuert).

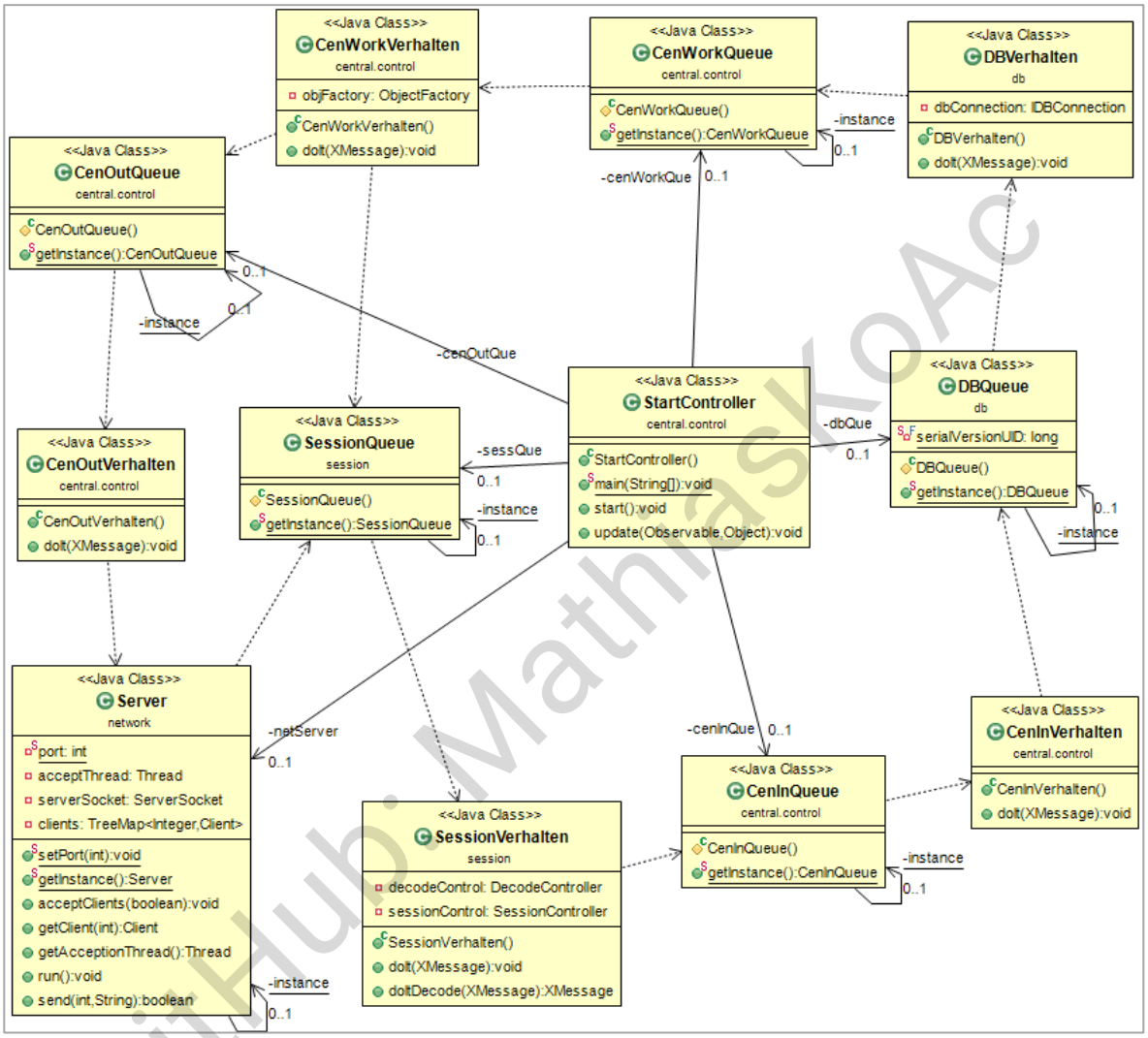

Abbildung 30: Klassendiagramm Queues

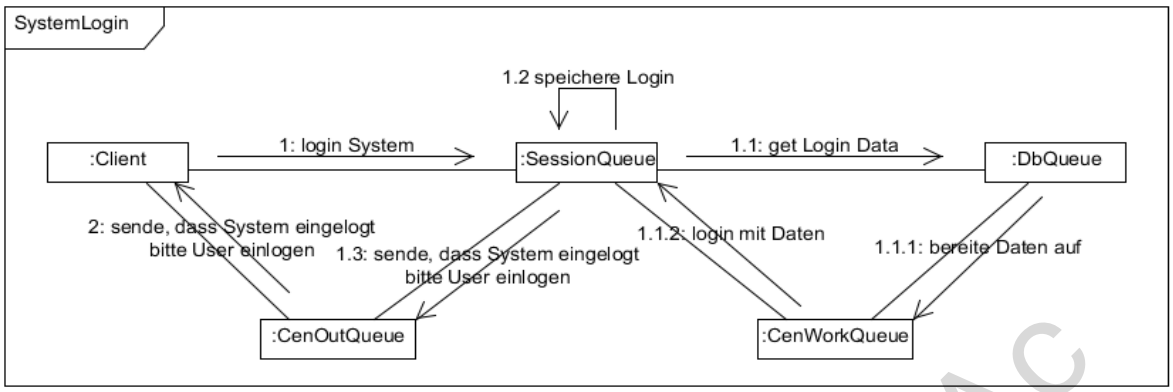

Abbildung 31: Kommunikationsdiagramm SystemLogin

In Abbildung 31 ist die Kommunikation zwischen den Queues für den Login eines Systems ohne laufende Session zu sehen. Diese Kommunikation ist stark vereinfacht und vernachlässigt die Aufgaben der Verhalten sowie andere Klassen, die in genauerer Betrachtung für die Aufgabe nötig sind.

- Im `Client` trifft eine `Message` ein.
- Die `Message` wird mit der Bitte um Login an die `SessionQueue` delegiert.
- Die `SessionQueue` findet keine Daten in der `SessionBase` und wird mit der Bitte um Login Daten an die `DbQueue` weitergegeben.
- Die `DbQueue` lädt die Daten und delegiert das Aufbereiten der Daten an `CenWorkQueue.`
- Die `CenWorkQueue` bereitet die Daten auf und sendet den Login mit den Daten an die `SessionQueue.`
- Die `SessionQueue` speichert die Daten des Login in die `SessionBase` und delegiert das Senden der Information, dass das System eingeloggt ist und sich der User einloggen soll, an die `CenOutQueue.`
- Die `CenOutQueue` prüft die `Message` auf Richtigkeit, verschlüsselt diese und schick sie an den `Client.`
- Der `Client` marshaled die `Message` und versendet diese an den Netzwerkpartner.

4.3.4.2 Netzwerk-Server

Eine weitere Ausnahme zum **`Queue-Modell`** stellt der **`Netzwerk-Server`** dar, der in Abbildung 30 durch die Klasse **`Server`** repräsentiert wird. Der Server läuft in einem eigenen Thread und generiert für jede neu eingehende Socket-Verbindung einen neuen **`Client`**. Dieser **`Client`** stellt sowohl den **`NetworkWriter`** als auch den **`NetworkReader`** zur Verfügung (vgl. Abbildung 32). Der **`NetworkReader`** läuft seinerseits in einem eigenen Thread. Somit wird für jeden **`Client`** ein Thread für den **`NetworkReader`** erzeugt. Die Kommunikation zwischen **`NetworkReader`** und dem **`Client`** geschieht über das Design Pattern Observer (vgl. [8] S. 37ff). Der Client liefert die ankommenden Daten an die **`SessionQueue`**.

Der verwendete Netzwerk-Server stellt eine überarbeitete Version des im SWE-Projekt „Rundenbasiertes Netzwerkspiel“ alias „Square-Commander“ verwendeten Netzwerk-Servers da. (vgl. [7]).

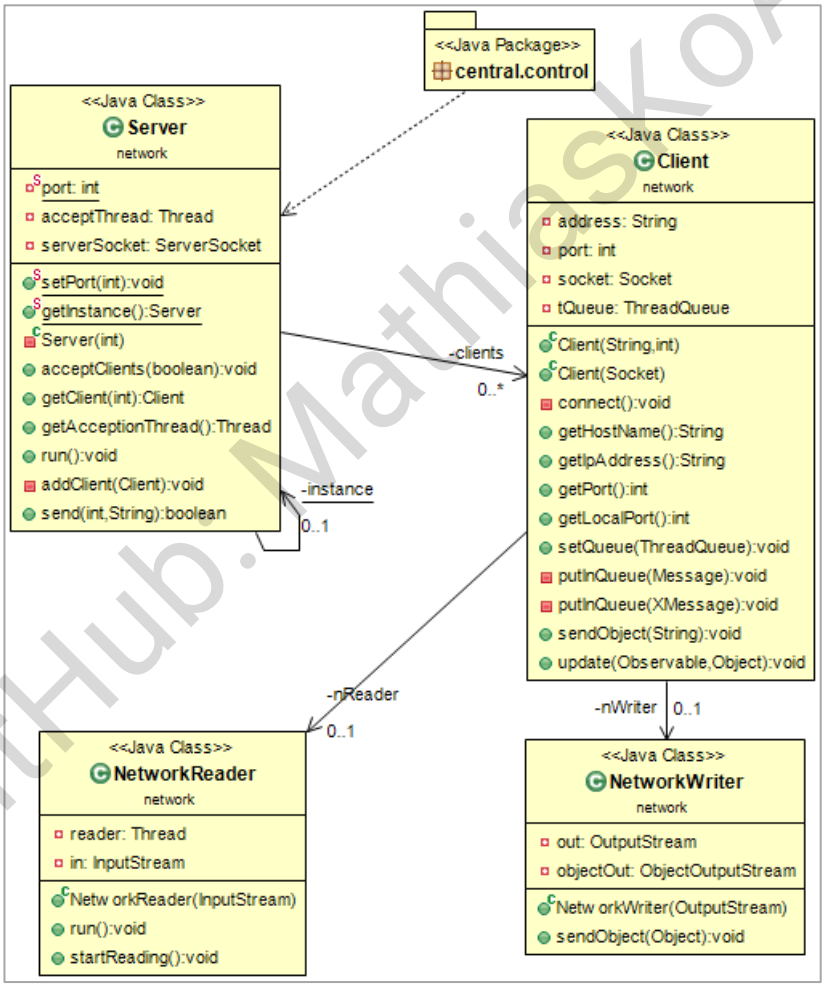

Abbildung 32: Klassendiagramm Netzwerk-Server

4.3.5 Automatische Abfragesprache

Die „Automatische Abfragesprache“ ist die Brücke zwischen dem aus der XSD-Datei erstellten Daten-Modell, das im Paket `model.generated` zu finden ist und der Datenbankanbindung. Mit Hilfe der Automatischen Abfragesprache wird ein Model erzeugt, dass sich schnell durch die Datenbankanbindung in den jeweiligen SQL-Standard, des angebunden DBMS übersetzen und abschicken lässt. In der Abbildung 33 wird dies durch den Schritt von „Java-Model generated“ zu „Java-Model prepared“ veranschaulicht.

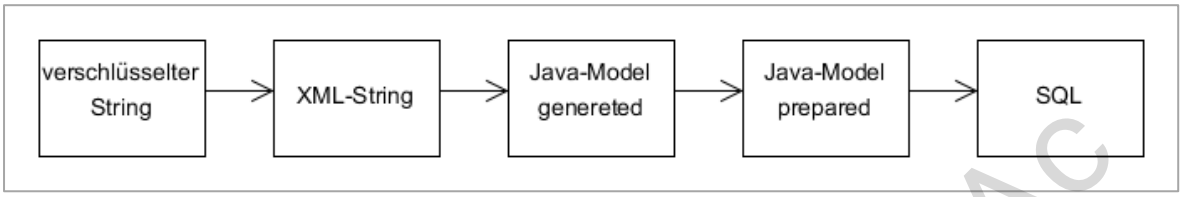

Abbildung 33: Datenumwandlung

Die Automatische Abfragesprache ist in Verbindung mit der Idee von PHP-Frameworks entstanden. Ein Grundsatz dieser Frameworks ist „Konvention vor Konfiguration“. Der andere Grundsatz in Bezug auf die Erleichterung des Datenbankzugriffs ist, dass die Modelle die Datenbank-Struktur kennen und mit dieser verbunden sind.

Daraus lässt sich eine Lösung finden, die generalisiert, objekt-orientierte Modelle auf Datenbank-Strukturen abbilden kann.

4.3.5.1 Voraussetzungen

Damit die Abfragesprache so funktioniert, wie im Konzept beschrieben, müssen drei Voraussetzungen erfüllt werden:

- Es muss zu jeder echten Tabelle eine Klasse in dem Paket `model.generated` existieren
- Das Datenbank-Modell darf keine Kreisschlüsse aufweisen
- Zwischentabellen aus der Auflösung von M-zu-N-Verbindungen müssen in der Struktur vermerkt sein

4.3.5.2 Trennung in Wrap-Model und Link-Model

Die Wrap-Models tragen diesen Namen, aufgrund der Verbindung mit dem Design Pattern Wrapper (alias Adapter). In diesem Design Pattern ummantelt eine Klasse eine andere, um diese an Schnittstellen anzupassen. Die Aufgabe des Wrap-Models ist es, die Daten einer Tabelle mit der einer Klasse zu verbinden. In diesem konkreten Anwendungsfall gibt es keine „echte“ Schnittstelle, sondern es geht darum, die Methoden der Abstrakten Klasse `AbsWModel` zu nutzen, ohne die generierten Models (`model.generated`) zu verändern.

Die Aufgabe des Link-Models ist es, die Relationen zwischen Tabellen darzustellen und mit diesen Informationen das Formulieren eines Inner-Joins in der SQL-Abfrage zu unterstützen und im From-Part der SQL-Abfrage eventuell fehlende Tabellen zu ergänzen.

Die Wrap-Models adaptieren jede Get- und Set-Methode der verbundenen, generierten Models, um diese an die zu generierenden SQL-Statements anzupassen. Somit funktionieren die Wrap-Models nur, wenn die zu ummantelnden Models existieren. Die Link-Models müssen jedoch auch die Struktur abbilden und Tabellen für einen JOIN ergänzen, wenn keine Instanzen der generierten Models vorhanden sind.

Daraus lässt sich ableiten, dass die Trennung zwischen Struktur und Daten notwendig ist, da das Link-Model funktionieren muss, wenn keine Daten vorliegen und das Wrap-Model nur funktionieren kann, wenn Daten vorliegen.

4.3.5.3 Wrap-Model

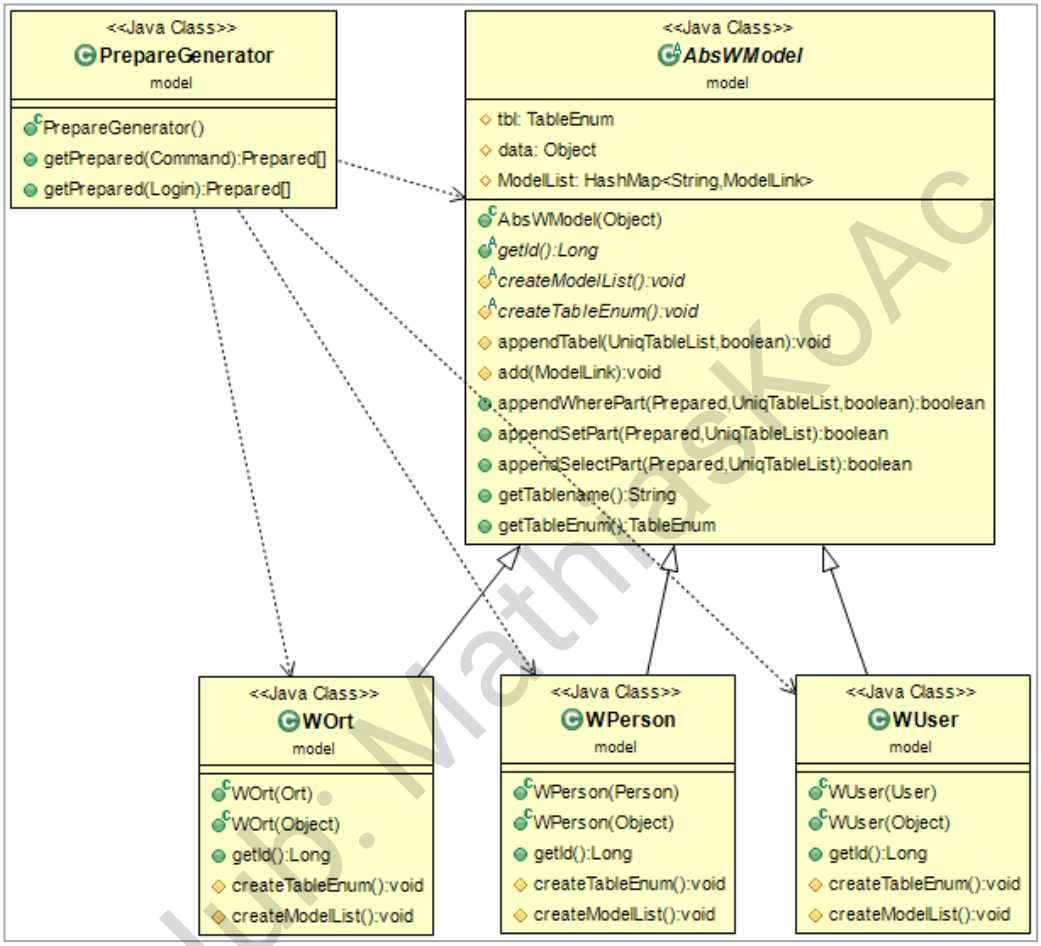

Abbildung 34: Klassendiagramm AbsWModel

Die Aufgabe des Wrap-Model ist es, gesteuert durch den `PrepareGenerator` das `Prepared` zu erzeugen, welches später durch die Datenbankanbindung genutzt wird. Um Redundanzen zu sparen wurde das Design Pattern Template angewendet und ein Großteil der Aufgaben zum Erstellen der `Prepared` in die Klassen `AbsWModel` ausgelagert (vgl. [8] S. 275ff). Wie in Abbildung 34 durch drei Klassen angedeutet, werden alle Wrap-Models von `AbsWModel` abgeleitet. Um eine Übersichtlichkeit zu wahren, sind alle Wrap-Models der Konvention unterworfen, den Eigennamen zusammengesetzt aus „W“ und dem Namen der zu adaptierenden Klasse.

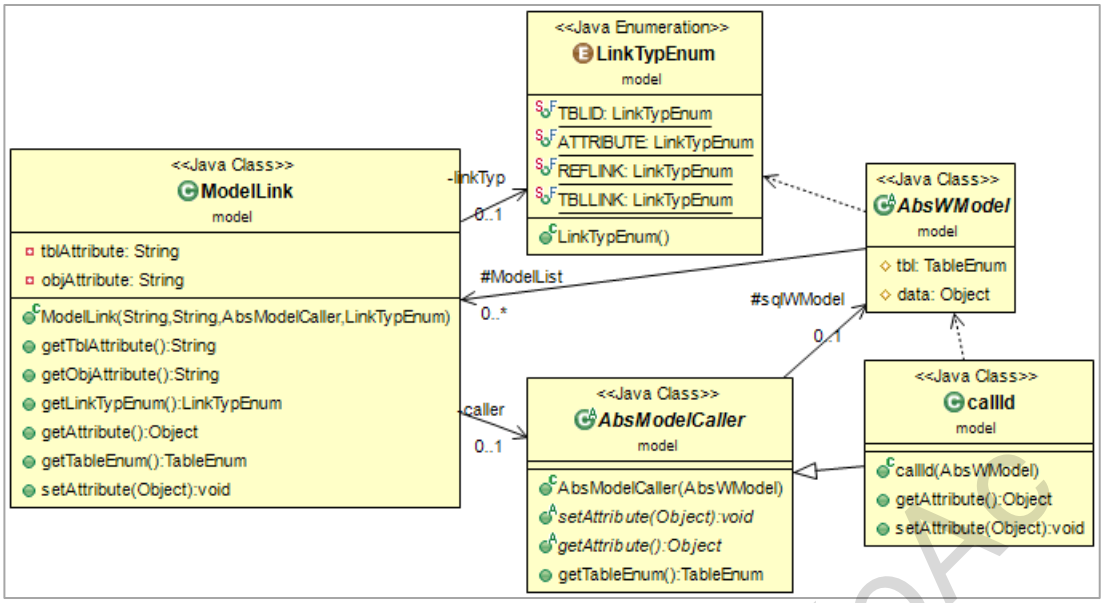

Abbildung 35: Klassendiagramm AbsModelCaller und ModelLink

Die WModels adaptieren die generierten Models, indem sie für jedes Paar aus Get- und Set-Methode eine Anonyme Innere Klasse erstellen, die von **`AbsModelCaller`** abgeleitet ist. Wie in Abbildung 35 zu sehen ist, besteht **`ModelLink`** aus einem **`ModelCaller`** (Implementierung eines **`AbsModelCaller`**, in Abbildung 35 **`callId`**) einem **`LinkTypEnum`** und zwei String Attributen. Die Aufgabe von **`ModelLink`** ist es, die direkte Verbindung zwischen Tabellenattribut und Klassenattribut zu schaffen und Informationen über die Art der Attribute bereitzustellen. Der Name des Tabellenattributs wird in **`tblAttribute`** und der Name des Klassen- bzw. Objektattributes wird in **`objAttribute`** gespeichert.

Die Methoden in **`AbsWModel`** erstellen aus den **`ModelLinks`** in der Map, die durch konkrete Implementierung des „AbsWModels“ erstellt wurden, das **`Prepared`**. Die Map ist in Abbildung 35, als **`#ModelList (0…*)`** zu erkennen. Durch die Implementierung der **`ModelCaller`** kann das **`AbsWModel`** direkt durch Iteration über die Map und das Anwenden der Methoden der **`ModelCaller`** auf die Attribute zugreifen. Somit müssen die Attribute dem **`AbsWModel`** nicht direkt bekannt sein.

Durch die logische Trennung zwischen dem Adapter und Template Design Pattern sowie der Anwendung von Anonymen Inneren Klassen, dessen Methoden wie Funktionspointer[3] genutzt werden, ist es möglich, sehr einfach **`WModels`** zu erstellen. Die geringe Kopplung sorgt dafür, dass für die Erstellung der **`WModels`** nur eine Überschreibung von abstrakten Methoden und das Erstellen von Anonymen Inneren Klassen notwendig ist (mehr Informationen zu Kopplung [10] S.316f).

Abbildung 36 ist ein Beispiel-Klassendiagramm für das Ummanteln eines Users. In dieser Abbildung wird unter anderem auch die konkrete Namenskonvention kenntlich. Sie beinhaltet als Prefix „call“ für die namentliche Verbindung zur Superklasse **`AbsModelCaller`**. Der Hauptbestandteil des Namens ist der Name des verbundenen Attributs.

[3] Der Begriff Funktionspointer wird in der Sprache von Java nicht verwendet und dient nur zur Beschreibung der Funktionsweise, wie die Methoden der inneren Klassen genutzt werden.

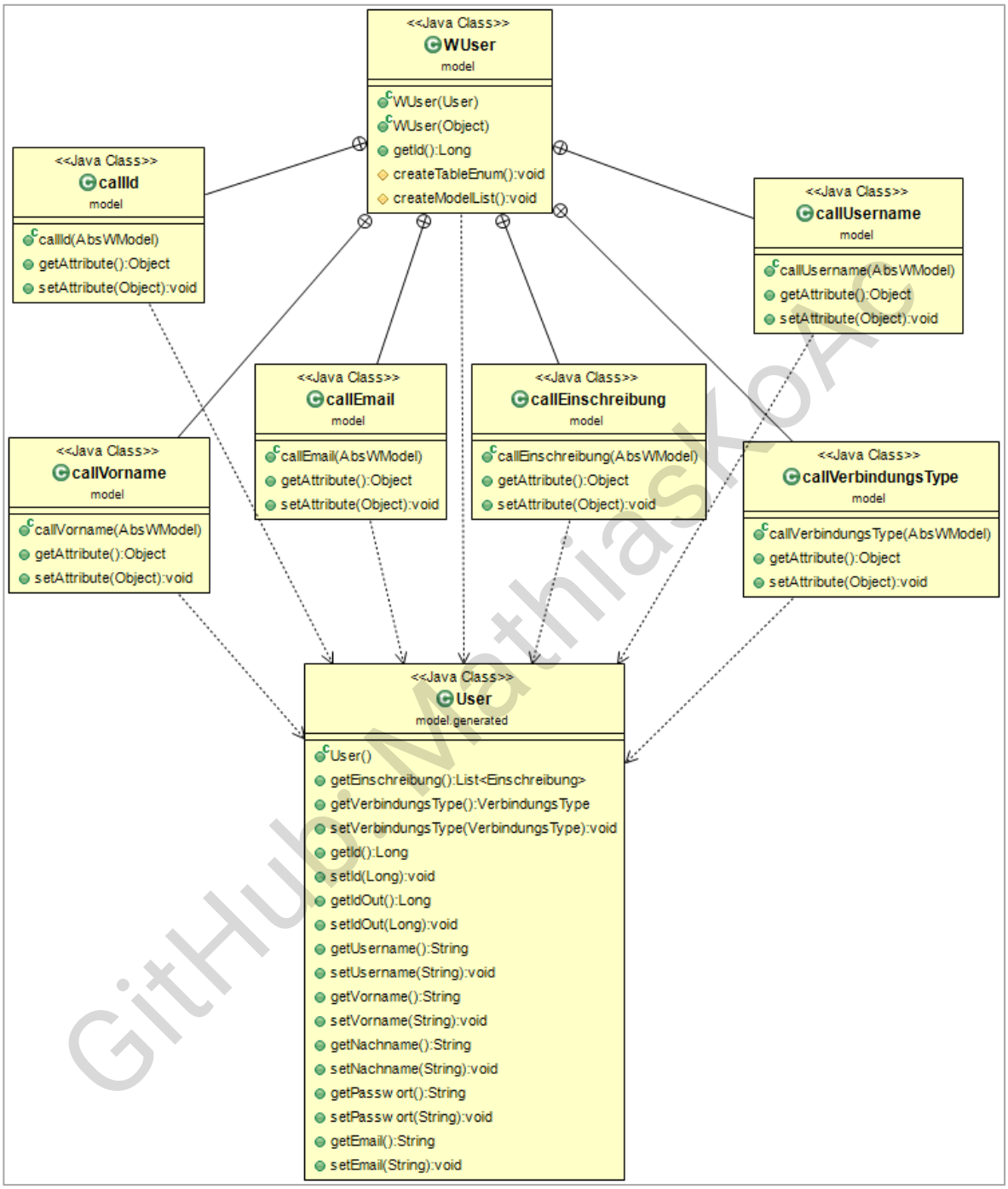

Abbildung 36: Klassendiagramm Wrap-User

In Abbildung 36 sind die mit „call“ beginnenden Klassen Innere Klassen der Klasse **WUser**.

4.3.5.4 Link-Model

Das Link-Model hat die Aufgabe, die Relations-Struktur des Datenbank-Models zu beschreiben. Das ist nötig, um die Zwischentabellen oder Tabellen, die im „Command“ der „Message“ nicht aufgeführt sind, nicht zu überspringen und den From-Teil und den Join-Teil des SQL-Statements richtig zu formulieren.

Für jede „nicht Referenztabelle“ (nRt) im Datenbank-Modell gibt es ein JModel, welches eine Lookup-Tabelle hält, die die Verbindungsinformation zu jeder anderen nRt im System enthält. In Tabelle 14 ist ein Beispiel gegeben, wie eine beliebige nRt über eine Lookup-Tabelle erreicht werden kann. Als Vergleich bietet sich Abbildung 44 aus Kapitel 4.3.7 an.

USER	
zu erreichende Tabelle (Ziel)	**über Tabelle (nächster Schritt)**
person	person
user	this
ort	person
workspace	einschreibung
rolle	einschreibung
…	…

Tabelle 14: Map für das Routing der Link-Models

Die sprachliche Interpretation der Lookup-Tabelle kann wie im folgenden Teilbeispiel aussehen:

- Von der Tabelle „User“ erreicht man die Tabelle „Person“ direkt über die Tabelle „Person“.
- Von der Tabelle „User“ erreicht man die Tabelle „User“ direkt, da es dieselbe Tabelle ist.
- Von der Tabelle „User“ erreicht man die Tabelle „Ort“ über die Tabelle „Person“.
- Von der Tabelle „User“ erreicht man die Tabelle „Workspace“ über die Tabelle „Einschreibung“
- …

Jede dieser Tabellen legt offen, über welche direkt angebundene Tabelle man das Ziel erreicht. Im JModel der nRt sind alle nRt als Ziel angegeben und als nächster Schritt die entsprechend angebundenen nRt. Die nRt werden für den Zweck der Referenz als Enum `TableEnum` gespeichert. Da jede nRt ein JModel mit Lookup-Tabelle hat, wird somit ein Netzwerk aufgebaut, welches das ganze DB-Model abdeckt. Zugunsten der Performance wurden die Referenztabellen nicht in das Modell aufgenommen (mehr dazu in Kapitel 4.3.5.5).

Da durch die Lookup-Tabelle, wie bis zu diesem Punkt vorgestellt, die Kardinalität der Relationen nicht berücksichtig wird, müssen diese noch hinzugefügt werden. Das Hinzufügen der Kardinalität geschieht durch die Klasse `JoinOption`. `JoinOption` enthält das Enum `TabelEnum` und je ein Boolean für die eigene Multi-Kardinalität und die der verbundenen Tabelle (vgl. Abbildung 37). Dadurch kann angegeben werden, ob es sich um eine 1-zu-N-Verbindung, N-zu-1-Verbindung oder 1-zu-1-Verbindung handelt. M-zu-N-Verbindungen werden nicht einzeln, sondern über eine Zwischentabelle als Zusammenwirken von 1-zu-N- und N-zu-1-Verbindungen abgebildet.

Ähnlich wie beim Wrap-Model wurde bei dem Link-Model das Design Pattern Template angewandt, um durch Kapselung Redundanzen zu vermeiden. Die Aufgabe des Link-Models ist es, die Struktur des DB-Models abzubilden. Da dies alle JModels gemeinsam haben, wurden die Methode und der Algorithmus in die Super-Klasse **`AbsJModel`** ausgelagert (vgl. Abbildung 37).

Die ausgelagerte Methode **`getJoinTuple()`** erweitert die übergebene Liste von **`AbsJoinTuple`**. **`AbsJoinTuple`** referenziert 2 **`TableEnum`** und die entsprechende Information, welche der **`TableEnum`** den Fremdschlüssel hält. Die **`AbsJoinTuple`** sind abstrakt, da die einen Join-Teil des SQL-Statements erzeugen und dieser DBMS abhängig ist. Somit wird das **`JoinTuple`** zusammen mit dem **`SqlGernerator`** für das jeweilige DBMS implementiert.

Die Unterschiede der JModel erklären sich aus der Tatsache, dass sie unterschiedliche DB-Tabellen abbilden. Entsprechend implementieren jedes JModel die Methode **`preSetTbl()`**, welche die Tabelle in Form des **`TableEnum`** setzt, und die Methode **`createLookupTab()`**, welche die Lookup-Tabelle aufbaut. Beide Methoden werden direkt durch den Konstruktor aufgerufen und sind deswegen **`protected`**.

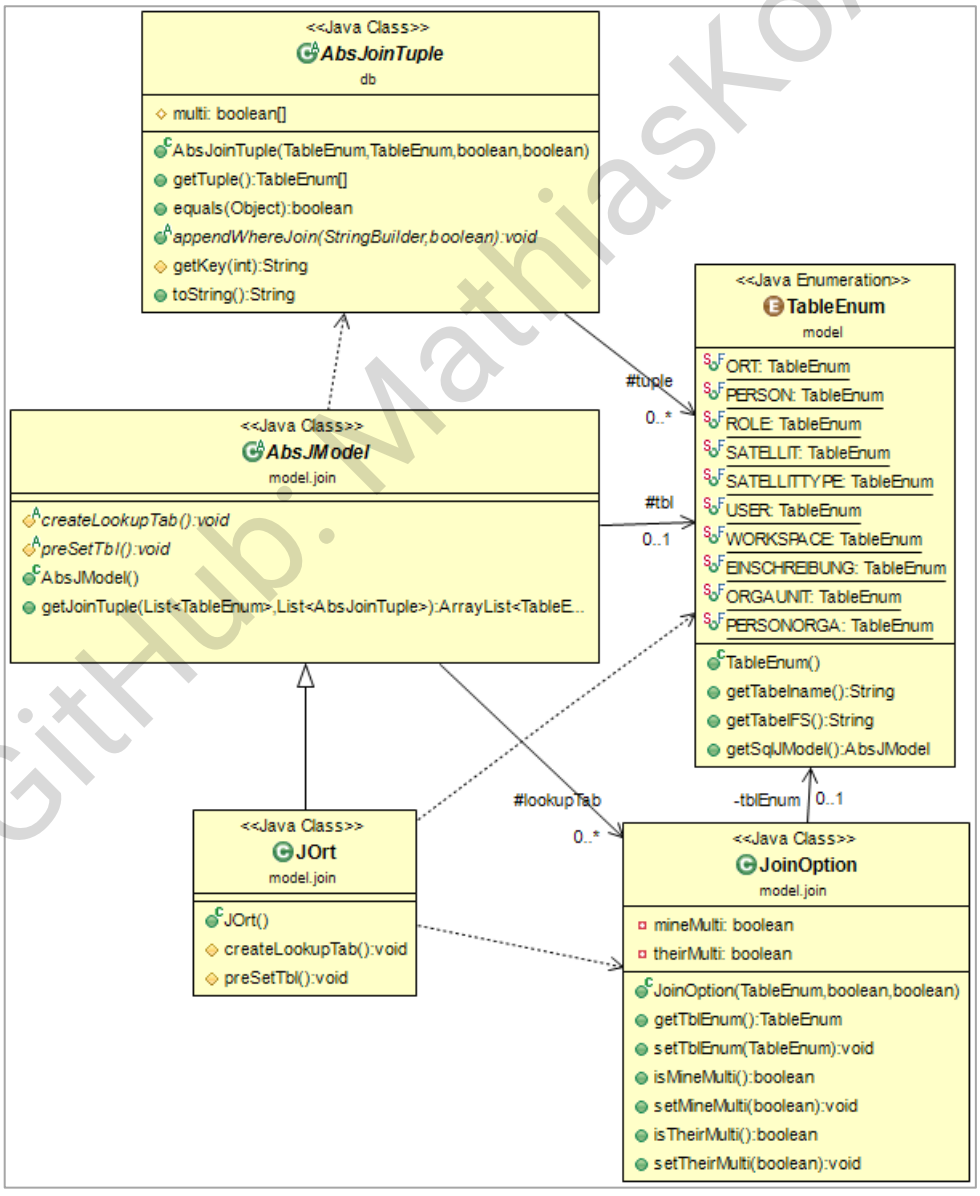

Abbildung 37: JModel / Link-Model

4.3.5.5 Referenztabellen und Enums

Enums und Referenztabellen werden in der Gesamtlösung als statischer Teil der Software angesehen. Sowohl Enums als auch Referenztabellen verbinden statische Werte – meist Strings – mit positiven Ganzzahlen. In XSD wird dies durch einen **`simpelType`** und einer **`restriction`** gelöst. Wenn man dieses Konstrukt in Java übersetzt, ist dies ein Enum. In dem DB-Model ist dies als Referenztabelle an zu sehen und wird durch das Prefix „**`ref_`**“ gekennzeichnet (mehr zum DB-Model in Kapitel 4.3.7.2).

Als Referenztabellen bzw. Enum sind **`SystemType`**, **`Anrede`** und **`VerbindungsType`** definiert.

- **`SystemType`** gibt den Systemtyp an und kann nur vier Werte annehmen: **`CENTRAL`** für den Central-Server, **`SATELLIT`** für Satelliten-Clients, **`GUI`** für GUI-Clients und **`SETUP`** für das Setup-Tool.
- Die **`Anrede`** stellt in dieser Version zwei Anredeformen dar: **`Herr`** und **`Frau`**.
- Der **`VerbindungsType`** zeigt an, wie stark die Verbindung zwischen **`User`** und **`Person`** ist, die das System oder der Endanwender vergeben hat (vgl. Kapitel 2.3.1.2 und 2.3.1.3).

Diese Referenztabellen werden nicht im Link-Model sowie dem Wrap-Model berücksichtigt, jedoch im Paket Model ähnlich dem Wrap-Model ummantelt. Jede dieser „Enum-Wrapper“ ist eine Klasse, die von **`AbsWEnum`** abgeleitet ist und sowohl eine statische Methode **`getEnum()`**, als auch eine objektabhänige Methode **`getId()`** beinhaltet (vgl. Abbildung 38).

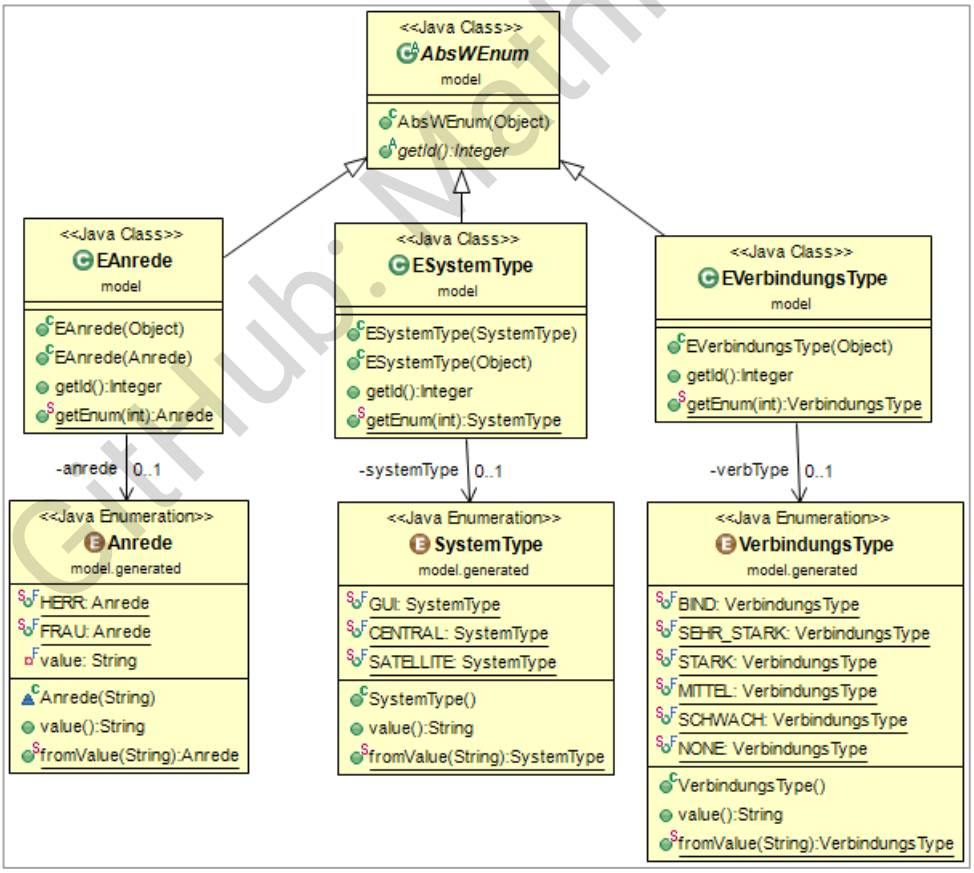

Abbildung 38: Enums und Referenztabellen

Über diese Wrapper werden die Enums in IDs übersetzt. Der JOIN sowie das Einbinden in das Link-Model entfallen, wodurch Aufwand gespart und Leistung gesteigert wird. Der schwerwiegendste Nachteil ist, dass bei Veränderung eines Standartwerts sowohl die Referenz-Tabelle als auch der Quellcode angepasst werden muss. Deshalb wurden in diesem System nur sparsam Enums im Datenmodell vorgesehen, um die Erweiterbarkeit des Systems zu erhalten.

Beispiele für weitere, mögliche Enums sind zum einen der Ort und zum anderen der Satellitentyp. Aufgrund der sehr großen Menge an Daten ist der Ort jedoch ungeeignet. Ebenso sollte der Satellitentyp nicht als Enum dargestellt werden, da in dem Lebenszyklus des Central-Servers neue Dritt-Systeme erschlossen werden sollen ohne den Central-Server anzupassen (vgl. 65. Anforderung). Ähnliche Überlegungen wurden für die Rollen durchgeführt, dabei greift dasselbe Argument, wie bei dem Satellitentyp.

4.3.5.6 Zusammenstellung eines Prepared

Das **Prepared** ist eine Klasse, die genutzt wird, um die für das Erstellen eines **PreparedStatement** relevanten Informationen zu kapseln. Das **Prepared** wird durch den **PreparedGenerator** erzeugt und durch den **SqlGenerator** in SQL übersetzt. Dieser Zwischenschritt über das **Prepared** ist nötig, damit unterschiedliche DBMS angesprochen werden können. Für jedes DBMS muss ein eigenes DB-Erweiterungs-Paket vorliegen.

Das **Prepared** hat im Gegensatz zu den anderen Models keine eigenen Fähigkeiten und trägt nur Daten. Ein **Prepared** besteht aus einem „Get-Teil“ oder einem „Set-Teil“ sowie einem „Where-Teil“, einem **CommandType** und der **UniqTableList** (vgl. Abbildung 39). Der „Get-Teil“ und der „Where-Teil“ bestehen aus einer Liste von **PreparedAttributeSets**. Der **CommandType** gibt an, ob das **Prepared** ein **SELECT**, ein **INSERT** oder ein **UPDATE** ist. Die **UniqTableList** sammelt alle Tabellen (in Form von **TableEnum**), die für den „From-Teil“ nötig sind und sorgt selbständig dafür, dass jede Tabelle nur einmal vorkommt.

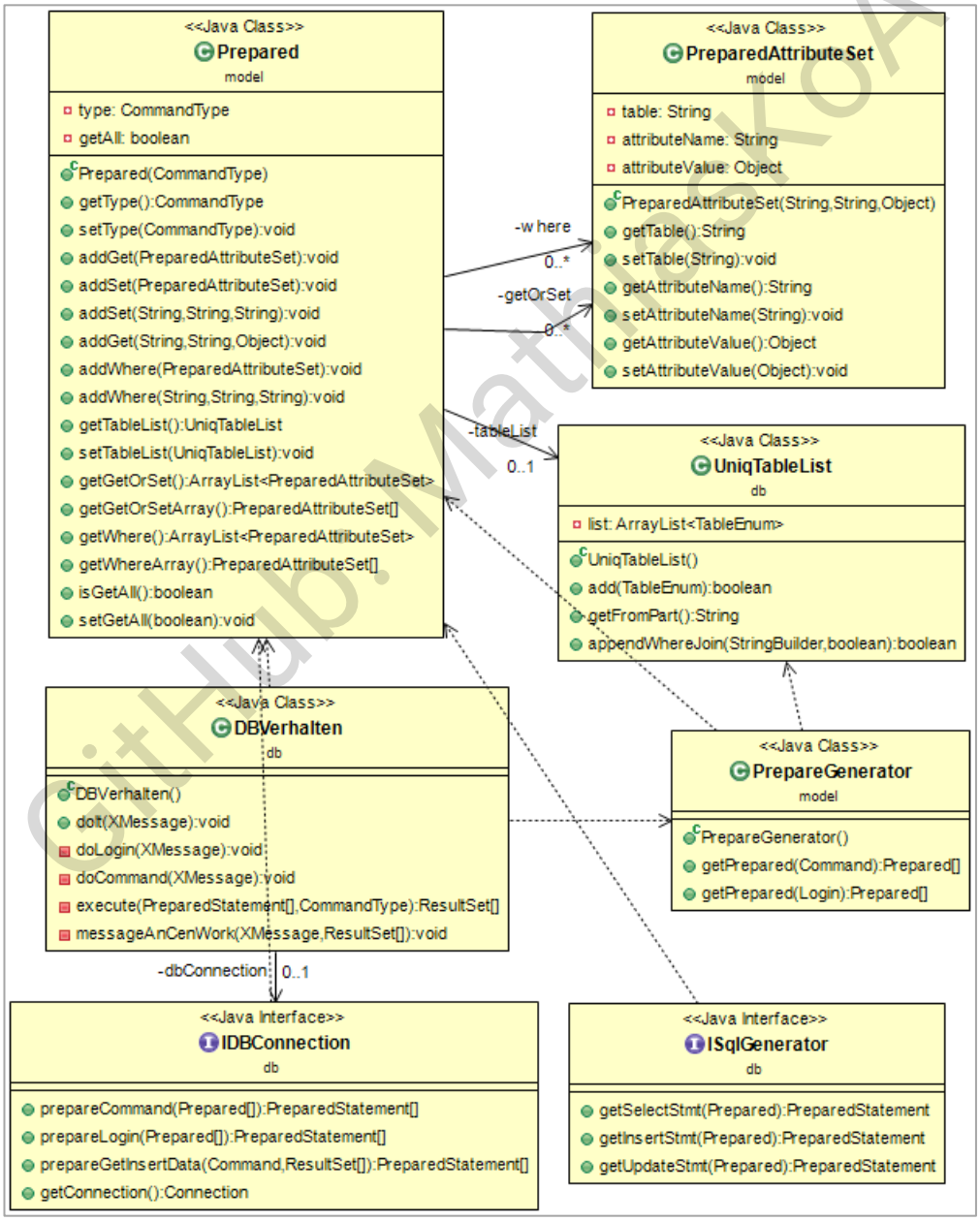

Abbildung 39: Klassendiagramm Prepared

4.3.5.7 Durchlauf eines Commands

Aufbauend auf den Kapiteln zuvor ist in Abbildung 40 ein kleiner Einblick als Durchlauf eines Commands durch **`DBVerhalten,`** mit einem **`Select`**, zusehen.

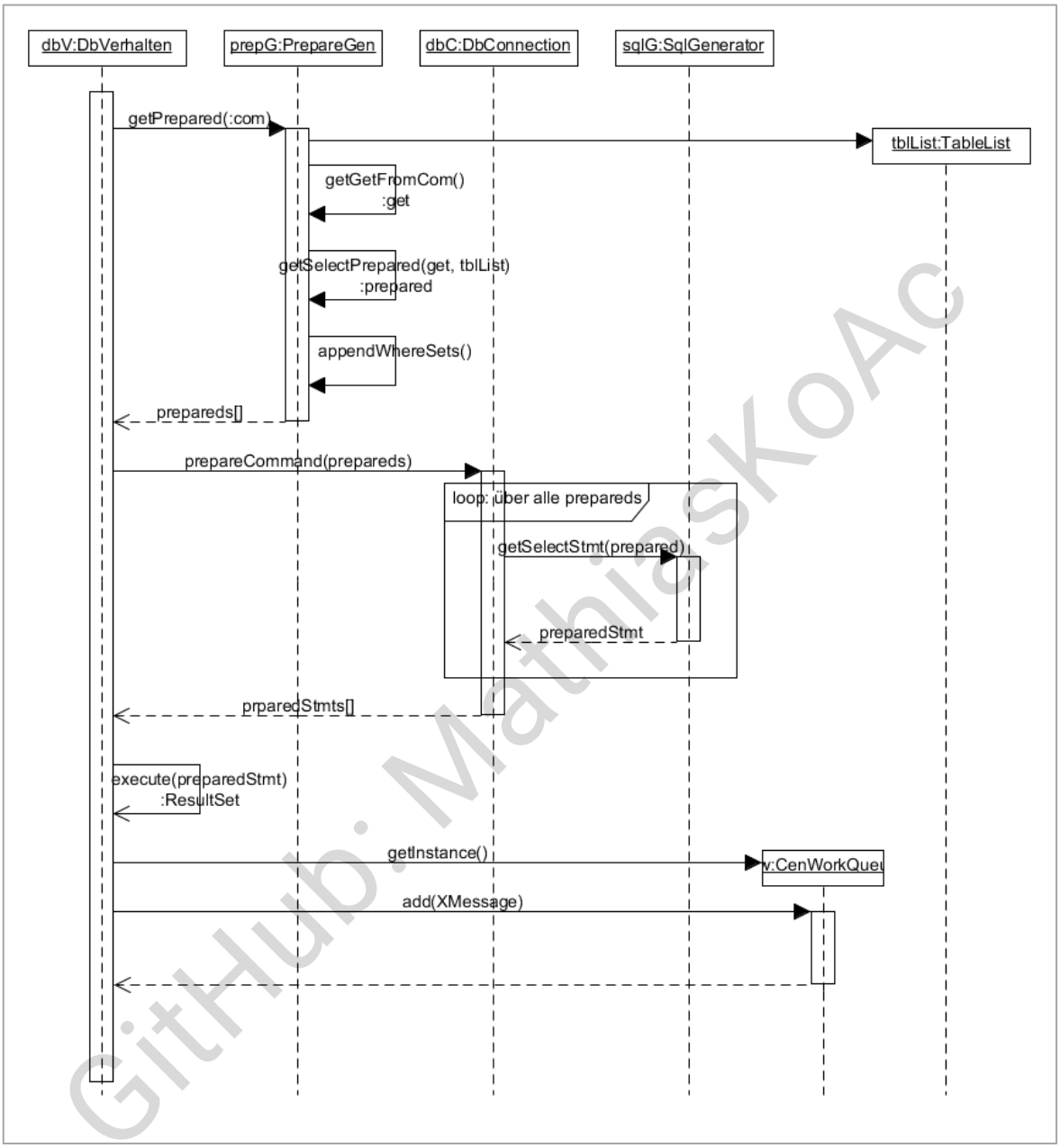

Abbildung 40: Sequenzdiagramm DBVerhalten Command Select

- Zunächst wird mit Hilfe des **`PrepareGen`** (**`PrepareGenerator`**) aus dem **`Com`** (**`Command`**) ein oder mehrere **`Prepareds`** erstellt-
- Diese **`Prepareds`** werden durch den **`SqlGenerator`**, angesprochen durch die **`DbConnection`**, in die **`preparedStmt`** (**`PreparedStatements`**) des entsprechenden DBMS übersetzt

- **DbVerhalten** führt diese **PreparedStatements** aus. In der Methode **execute()** wird intern **DbConnection** verwendet. Dies ist aufgrund der reduzierten Detailliertheit in Abbildung 40 nicht zusehen.

- Das **execute()** liefert ein **ResultSet** zurück, was an die ursprüngliche **XMessage** angehängt und in die **CenWorkQueue** zur Datenbindung weitergegeben wird.

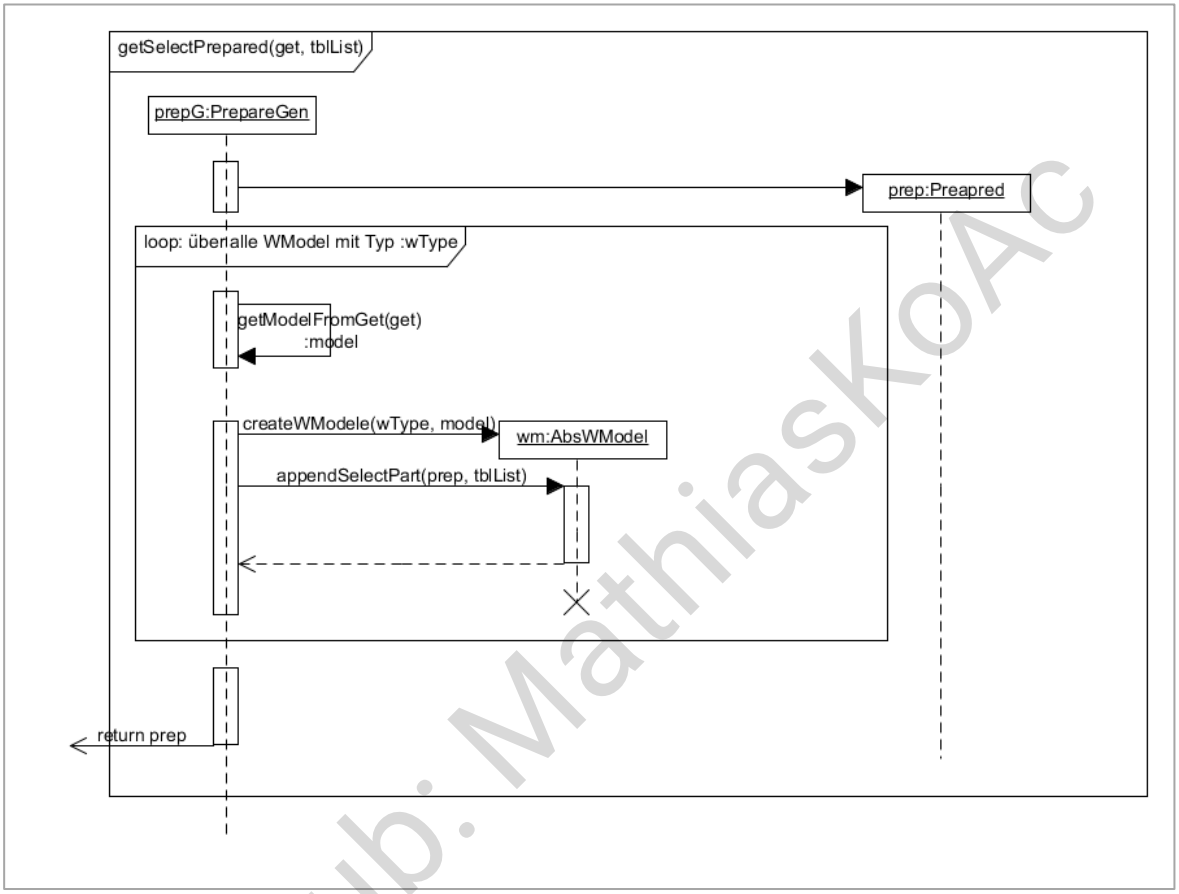

Abbildung 41: Sequenzdiagramm PrepareGen.getSelectPrepared

In Abbildung 41 ist die Methode **getSelectPrepared()** zusehen, die den Select-Part des **Prepareds** liefert.

- Jeder Aufruf dieser Methode baut eine Instanz des **Prepared**, welcher als Rückgabewert zurückgegeben wird.

- Um in dem **Prepared** alle Daten-Models abgedeckt zu haben, wird über die **wTypen** (**TableEnum**) iteriert um aus den Daten-Models (aus **model.generated)** und dem **TableEnums, WModels** zu erstellen. Diese **WModels** hängen dann jeweils ihren Teil des Select-Parts an.

4.3.6 Datenbindung

Die Datenbindung kommt im Server zum Einsatz, wenn ein Command bzw. eine Abfrage im System durchgeführt wurde. Die letzte Station bevor die Anfrage aufgelöst wird, ist das **DBVerhalten**. Das Ergebnis einer solchen Abfrage ist ein **ResultSet**, diese gibt das **DBVerhalten** direkt weiter an die **CenWorkerQueue**, in der es aufbereitet wird. Dieses Aufbereiten ist die Datenbindung.

In dem **ResultSet** liegen die Daten nach einem Select in einer tabellenähnlichen Struktur vor. Die Struktur besteht aus MetaDaten und Nutzdaten. In den MetaDaten sind die Tabellennamen und die Spaltennamen hinterlegt, die den Aufbau der Struktur beschreiben. Die Nutzdaten liegen somit in einer Ebene und können nur über den Tabellennamen den Model-Klassen zugeordnet werden.

Da ein **ResultSet**-Objekt einer Zeile entspricht, wird das Ergebnis auch in einer zeilenorientierten Weise weitergegeben. Jedes **ResultSet**-Objekt wird auf ein **DatenZeile**-Objekt gemappt. Ein **DatenZeilen**-Objekt hält genau ein Objekt des Daten-Models aus dem Paket **model.generated**.

Wie auch in anderen Bereichen des Servers wurde auch hier auf das Design Pattern Wrapper zurückgegriffen, damit in der Wrapper-Klasse die Methode implementiert werden kann, um die Daten des **ResultSet** in eine **DatenZeile** zu übertragen. Damit ein mehrzeiliges Ergebnis zurückgegeben werden kann, werden in die **DatenZeile**-Objekte in dem Objekt **Data** gesammelt (vgl. Abbildung 42).

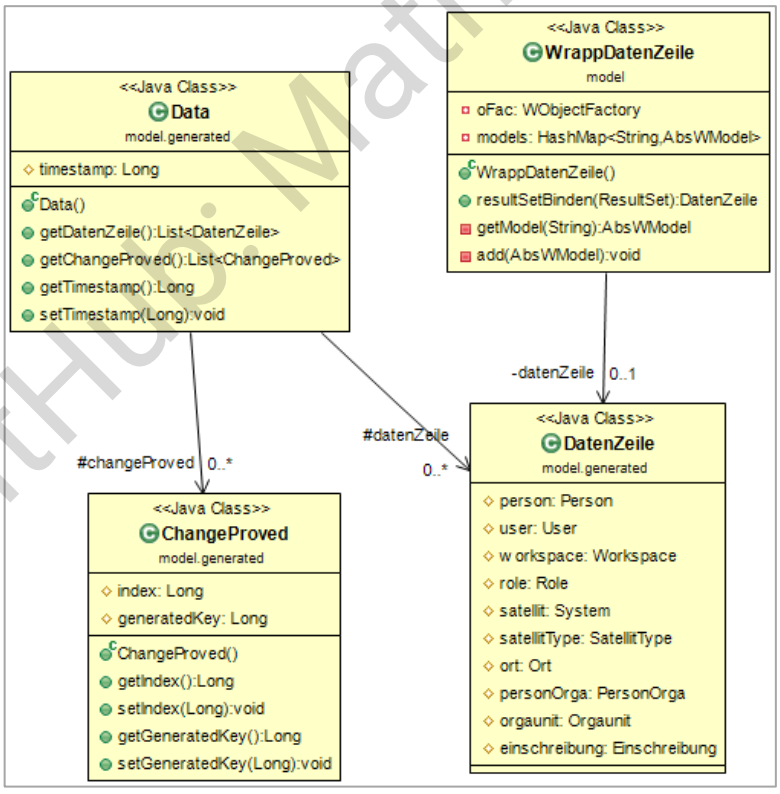

Abbildung 42: Klassendiagramm DatenZeile

Die **WrappDatenZeile** (vgl. Abbildung 42) erstellt in der Methode **resultSetBinden()** mit Hilfe der **WObjetkFactory** WModels, in welche die Daten gespeichert werden. Über die erstellten WModels können dann einfach die Daten in die Daten-Models gespeichert werden. Die WModels halten Model-Link Objekte, die Zugriff auf die Anonymen Inneren Klassen haben, die die Get- und Set-Methoden der Daten-Models ummanteln. Die ModelLink Objekte werden in einer **Map** gespeichert, die als Keys die Namen der verbunden Tabellenspalten tragen. Über die Keys lässt sich direkt die Verbindung zum DB-Model herstellen und die Daten-Models füllen.

Wenn das **ResultSet** nach einem oder mehreren Inserts oder Updates gemappt wird, liegen die Daten auch in einer tabellenähnlichen Struktur vor, allerdings mit der Einschränkung, dass es nur eine Spalte pro Datensatz gibt. Diese eine Spalte ist die **ID**, bzw. der Primärschlüssel der Tabelle. Dadurch, dass das **ResultSet** keine Rückschlüsse auf die veränderte Tabelle gibt, muss im Gegensatz zum Binden des **ResultSets** nach einem Select ein anderes Model gewählt werden.

ChangeProved ist das Model der Wahl, denn in dieser Klasse werden sowohl der **Index** als auch die **ID** abgelegt. Diese Informationen reichen, um ein Insert oder Update zu bestätigen. Um unnötige Abfragen zu vermeiden, wurde an dieser Stelle auf das automatische Generieren eines Select-Statements verzichtet. Wenn ein Setup-Tool oder ein GUI-Client die neu gesetzten Daten in der Datenbank abfragen möchte, muss dazu ein neuer Command abgesetzt werden.

4.3.7 Datenbank-Modell

Das Datenbankmodell wurde parallel zum Datenteil des XSD-Models modelliert. Die Modellierung des Datenbank-Modells geschah in mehreren Schritten unter Zuhilfenahme zweier Modelltypen.

Das ER-Modell (Entity-Relationship-Modell) wurde an die Erweiterte-Chen-Notation angelehnt. Dieses wurde bedingt durch das Programm zum Erstellen der Diagramme, mit der Krähenfuß-Notation / Martin-Notation erweitert. In dieser Verwendung wurde mit einer höheren Abstraktion gearbeitet und auf die Auflistung von Attributen verzichtet.

Das DB-Modell wurde an die UML-Notation für Klassen-Diagramme und IDEF1X angelehnt.

4.3.7.1 ER-Modell

Bei der Entwicklung des Modells war es eine Voraussetzung, dass Kreisschlüsse verhindert werden, um ein automatische Generieren von SQL-Statements zu ermöglichen.

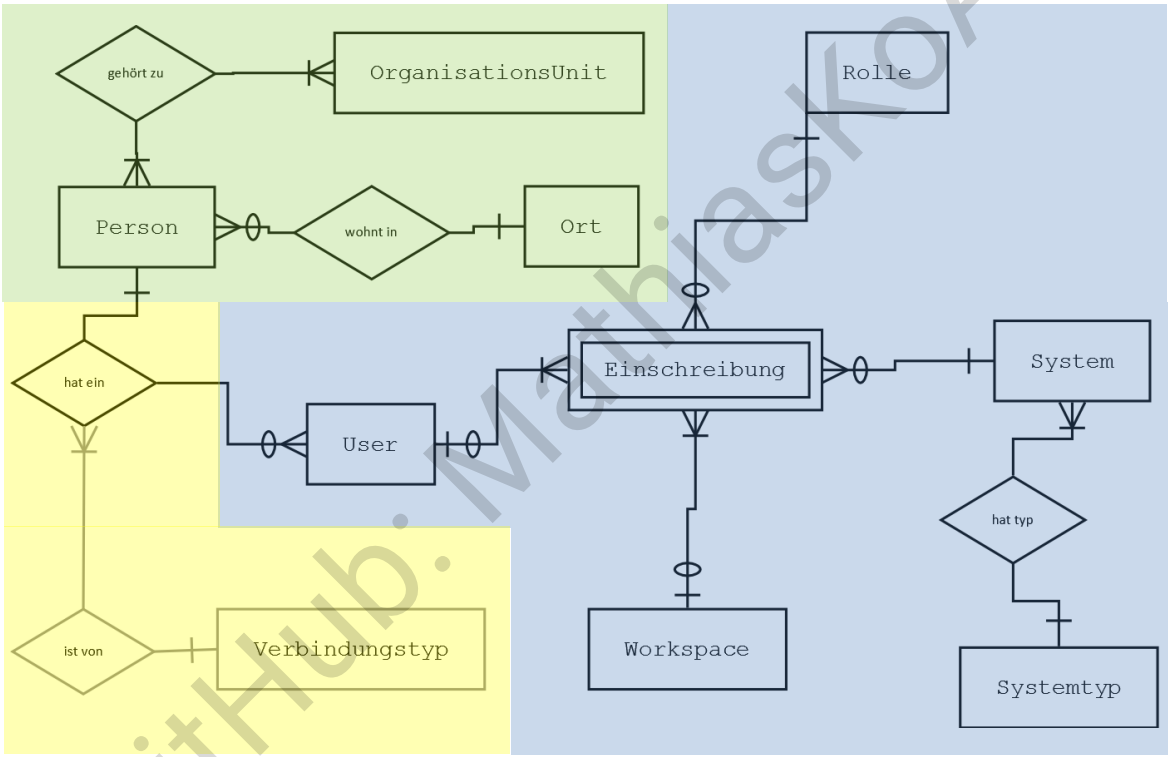

Abbildung 43: ER-Model

Das ER-Model lässt sich logisch in verschiedene Bereiche unterteilen. Diese Bereiche sind in der Abbildung 43 folgendermaßen gefärbt: Systembereich blau, Personenbereich grün und Verbindungsbereich gelb.

- Der Systembereich bildet die Daten aus den Satelliten-Servern sowie die Daten für GUI-Clients und die Kommunikationsinformationen ab. Die Daten vieler Systeme, die eine Userverwaltung haben, lassen sich auf dieses Muster aus User, Arbeitsbereich und Rolle zurückführen.

- Der Personenbereich bildet die Informationen der natürlichen Personen, sowie deren Zugehörigkeit zu OrganisationUnits (OU) ab. Diese Informationen sollen unabhängig vom Rest des Modells geändert werden können.

- Der Verbindungsbereich ist die Schnittstelle zwischen Systembereich und Personenbereich. Der Verbindungstyp ist nötig, um die Bindung zwischen Person und User abzubilden und zu klassifizieren (siehe 10. Anforderung). Diese Klassifizierung soll zwischen automatisch aufgezeigten Verbindungen und deren Stärke, sowie zwischen bestätigten und ausgeschlossenen Verbindungen unterscheiden.

4.3.7.2 DB-Modell

Das DB-Modell in Anlehnung an das UML-Klassendiagramm ist platzsparender, als die erweiterte Chen-Notation mit Attributblasen. Im Gegensatz zu der konzeptionellen Überlegung mit Hilfe des ER-Modells werden im DB-Modell die Tabellen so dargestellt, wie sie in der Datenbank angelegt sind.

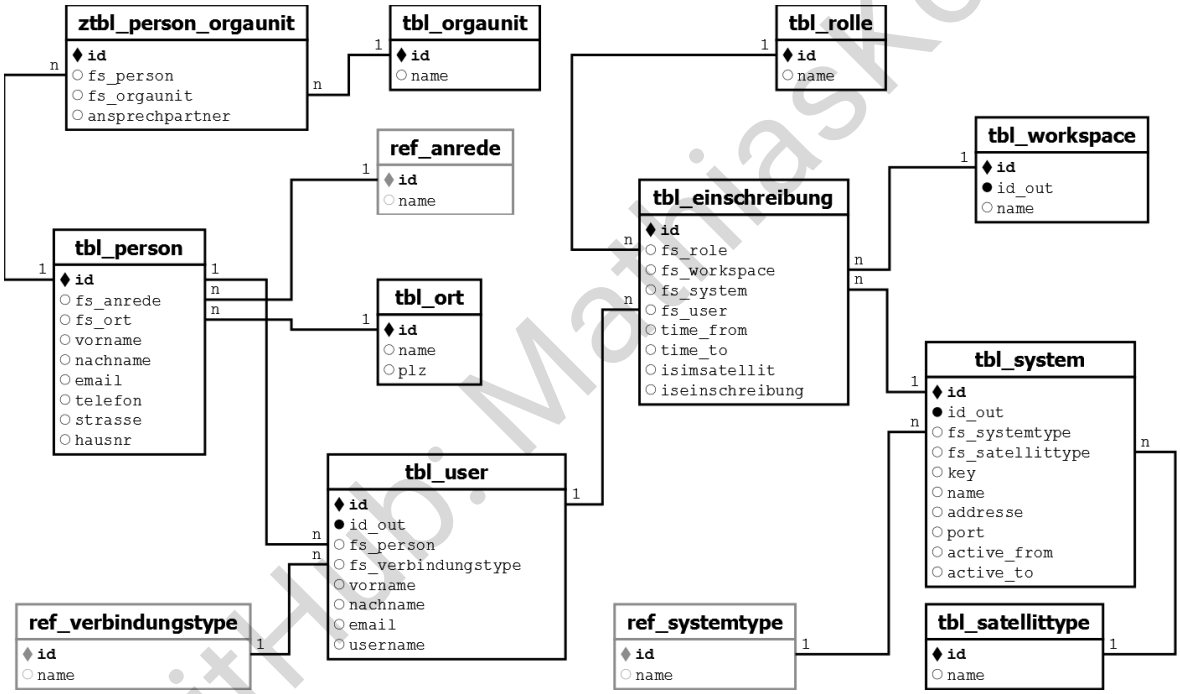

Abbildung 44: DB-Modell

Bei der Normalisierung und Betrachtung der Attribute wurden folgende Tabellen, zur Vervollständigung, ergänzt:

- **ref_anrede** um die Anrede referenziert abzubilden

- **tbl_satellittype** um den Typ des Satelliten zu vereinheitlichen

- **ztbl_person_orgaunit** um die Verbindung zwischen Person und Organisationseinheit zu modellieren

4.3.7.2.1 Konvention und Muster

Beim Erstellen stand der Leitsatz „Konvention statt Konfiguration“ im Fokus, was viele wiederkehrende Muster und damit Vereinfachung mit sich trägt.

- Alle Attribute in der Datenbank werden klein geschrieben.
- Die Tabellen haben synthetische IDs als Primärschlüssel, was das Migrieren in andere Datenbank erleichtert und die Verbindung mit Fremdschlüsseln simpler gestaltet.
- Der Primärschlüssel einer Tabelle wird immer als Attribut **`id`** angegeben und ist ganzzahlig nummerisch.
- Die Attribute, die als Fremdschlüssel fungieren, erhalten den Präfix **`fs_`**.
- Zwischentabellen, welche beim Auflösen von M-zu-N-Verbindungen entstehen, bekommen statt einem zusammengesetzten, einen eigenen Primärschlüssel. Der Name dieser Tabellen setzt sich aus den Tabellennamen der referenzierten Tabellen zusammen.
- Tabellen die Daten halten, welche auch in anderen Systemen gespeichert werden, tragen das Attribut **`id_out`** was die ID des anderen Systems wiederspiegelt.

Bei der Umsetzung von Entitätstypen in Tabellen wurden die Tabellennamen mit Präfixe erweitert um die Übersichtlichkeit zu gewährleisten.

- tbl_ für Tabellen mit sich änderndem Inhalt
- ztbl_ für Tabellen, die aus aufgelösten M-zu-N-Verbindungen entstanden sind
- ref_ für Tabellen dessen Inhalt sich nicht ändert und als Referenz dienen

4.3.7.2.2 Denormalisierung

Unter Denormalisierung versteht man einen Schritt der Rücknahme von Normalisierung, nach dem die Normalisierung in gewünschter Weise abgeschlossen ist. Denormalisierung wird verwendet, um die Performance des Systems zu verbessern oder spezielle Sachverhalte zu vereinfachen (vgl. [11]).

In dieser Arbeit sind nachfolgende Stellen denomalisiert:

- Die Tabelle **`ref_verbindungstype`** ist denormal mit **`tbl_user`** verbunden, da die Verbindung nicht zwingend von dem Primärschlüssel des Users abhängt. Damit ist die **`tbl_user`** nicht in der Zweiten-Normalform. Der Verbindungstyp würde normalisiert nur die Verbindung von Person zu User beschreiben und müsste in einer Zwischentabelle gelöst werden. Gleichzeitig müsste man in der Zwischentabelle erzwingen, dass keine M-zu-N-Verbindung ersteht. Abbildung 61, im Anhang, zeit die alternative Modellierung.
- Die Tabelle **`tbl_satellittype`** ist denormal mit **`tbl_system`** verbunden, da Systeme wie der Central-Server und GUI-Clients keinen Satellitentyp haben können. Der Satellitentyp ist nicht funktional von dem Primärschlüssel jedes Systems abhängig. Da diese Abhängigkeit nicht gegeben ist, ist dadurch die Tabelle **`tbl_system`** nicht in der Zweiten-Normalform. Abbildung 60, im Anhang, zeigt die alternative Modellierung, welche in Zweiter-Normalform ist.

Da der Satellitentyp durch ein Fremdschlüssel mit der Tabelle **tbl_system** verbunden ist, sorgt diese Verbindung, in der gewählten Modellierung dafür, dass keine Redundanzen auftreten. Das Konstrukt so zu belassen ist performanter, als die Trennung zwischen den Systemen durch Vererbung und das Spezialisieren der einzelnen Systeme.

4.3.7.2.3 Sonderfall gemeinsame Kontaktdaten

Die Tabellen **tbl_user** und **tbl_person** haben zum Teil übereinstimmende Attribute. Diese Attribute könnten thorethisch in eine neue Tabelle **tbl_kontaktdaten** ausnormalisiert werden. Diese Daten haben eine andere Zugehörigkeit, denn eine natürliche Person kann andere Kontaktdaten haben, als ein User, der eine Rolle spielt. Die mögliche Ausnormalisierung würde zu einem Kreisschluss führen, der so im Link-Model (vgl. 4.3.5.4) nicht vorgesehen ist.

4.3.7.2.4 Sternschema

Das Sternschema ist eine Denormalisierung, welche die Bündelung der Verbindungen (oder möglicher Zwischentabellen) an eine zentrale Stelle zu Gunsten von schnelleren Abfragen erklärt. Für diese schnellen Abfragen werden Redundanz und mehr Aufwand bei Schreibvorgängen in Kauf genommen. Das Sternschema wird häufig an den Stellen eingesetzt, welche den Kern des Teilmodells binden sollen.

Die Tabelle **tbl_einschreibung** ist im Sternschema und damit nicht im strengsten Sinne normalisiert. Die Einschreibung bildet nicht den Kern des Gesamtmodells, jedoch des Modellteils für den Systembereich. In diesem Modell bilden Einschreibungen die zentralen Anwendungsfälle: „Wer ist wo (Workspace), in welchem System, mit welcher Rolle eingeschrieben?“

4.3.7.2.5 Speziell MySQL

Für die Speicherung von IDs wird im Normalfall von dem Datentyp „BIG INT“ ausgegangen. Es werden bei den Schlüsseln größere Werte auf Grund von wachsenden Einträgen erwartet. Für die Speicherung von IDs der Referenztabellen wird von dem Datentyp „INT“ ausgegangen um die Kompatibilität mit Enums in Java aufrecht zu erhalten.

Für das Speichern von Userdaten als Zeichenkette ist der Typ VARCHAR zu empfehlen, da er dynamisch den Speicherbedarf der Zelle anpasst (vgl. [12]). Ein Beispiel dafür ist folgendes: Für das Feld „Name“ ist ein eine Länge von 20 Zeichen vorgesehen. Wenn in dem Feld nur ein Name mit vier Buchstaben eingetragen ist, würde ein Feld mit Typ CHAR 20 Byte, ein Feld vom Typ VARCHAR jedoch nur fünf Byte belegen. Der Nachteil von VARCHAR gegenüber CHAR ist, dass wenn die Zelle leer bleibt, VARCHAR ein Byte Speicher belegt, CHAR aber nichts.

Als Zeichensatz in der Datenbank wird das Charset „latin1“ vorausgesetzt. Die Konvertierung in den Zeichensatz von Java wird automatisch durch den JDBC-Connector durchgeführt (vgl. [13]).

Die genaue Beschreibung der Datenbanktabellen befindet sich im Anhang (vgl. 8.3).

5 Sicherheit

Sicherheit ist im Softwaresektor ein heikles und wichtiges Thema. In dieser Arbeit wird die Sicherheit im Sinne der Datensicherheit betrachtet, welche teilweise gegenüber der Ausfallsicherheit priorisiert wird, um die Privatsphäre der Halter der Benutzerkonten nicht zu gefährden. Im Rahmen der Arbeit möchte ich mich nur auf die Datensicherheit des Systems beziehen, die man als Softwareentwickler direkt beeinflussen kann.

5.1 Risiken

Sicherheit ist nicht die Abwesenheit von Risiken, sondern die Abwesenheit von unakzeptablen Risiken (vgl. [14] S.440). Im folgenden Abschnitt werden klassische Risiken und deren Gegenmaßnahme erläutert.

5.1.1 SQL-Injektion

SQL-Injektion ist das Einschleusen von SQL in eine Software, die mit SQL arbeitet, um dem System zu schaden oder versteckte Daten auszulesen. Die klassischen Gegenmaßnahmen sind das „Escapen von Spezialchars“ und das Verwenden von Prepared-Statements. Generell ist das Prepared-Statement zu bevorzugen, da es die Grundlade für die Angriffsmöglichkeit beseitigt. Die SQL-Injektion funktioniert nur, weil das System, in letzter Instanz das DBMS, nicht zwischen Daten und Anweisungen trennen kann, da beide Teile in einem String zusammen abgesendet werden. Durch das Prepared-Statement wird genau diese Trennung geschaffen. Das Prepared-Statement wird als komplettes Command mit Platzhaltern abgesendet, so ist dem DBMS bekannt, welcher Teil des Gesamt-Statements das Command ist. Die Daten werden nachgesendet und als solche erkannt, selbst wenn in den Daten SQL-Statements oder Teile von SQL-Statements versteckt sind.

5.1.2 Code-Injektion

Code-Injektion ist ein Oberbegriff für das Injekten (oder Injizierten) von Codefragmenten in eine bestehende Software. Generell könnte man SQL-Injektion auch als Code-Injektion bezeichnen, jedoch wird SQL selten als Code bezeichnet. Code-Injektion ist das Einbringen von ausführbarem Code in eine Anwendung, meist mit der Intention dem System zu schaden. Code-Injektion wird häufig in Script-Sprachen durchgeführt, da der Code, der ausgeführt wird, nicht kompiliert werden muss und teilweise durch Interpretation von anderem Code erst erstellt wird.

In Java ist Code-Injektion schwieriger und seltener als bei Skriptsprachen, deshalb gehe ich hier nicht näher auf Code-Injektion in Java ein. Durch die Trennung von GUI-Client und Central-Server ist dies noch schwieriger, da man den Effekt nicht in dem GUI-Client sondern in dem Central-Server durchführen muss, um Schaden anzurichten. Da in Satelliten-Clients Script-Sprachen verwendet werden können, gehe ich hier darauf ein.

Man kann Code-Injektion direkt oder indirekt ausführen. Direkt durch Eingeben des Codes in die Felder des XML-Strings, was dann durch den jeweiligen Empfänger abgefangen werden muss. Da der Empfänger die XSD-Datei hat, die Auskunft darüber gibt, wie das XML aussehen darf, ist klar, wo sich Daten befinden. Diese Daten müssen im weiteren Verlauf im Programm entsprechend behandelt werden. Indirekt durch XML in dem XML, quasi durch XML-Injektion. XML-Injektion wird verhindert, da der Sender und Empfänger mit Hilfe der XSD-Datei das XML prüft.

5.1.3 Sniffing

Sniffing ist das Abfangen oder Abhören von Datenverkehr im Netzwerk. Sniffen ist dann besonders ertragreich, wenn die Netzwerkkommunikation nicht verschlüsselt und in einem Broadcastnetzwerk geschieht. Gegenmaßnahmen sind Verschlüsseln und Zugang einschränken.

In dem Gesamtsystem wird der Netzwerkverkehr von allen Teilnehmern von Endpunkt zu Endpunkt verschlüsselt. Die Verschlüsselung ist zudem für jeden Teilnehmer anders und wird pro Message-Paar neu vermittelt. Ein Risiko der Verschlüsselung ist, dass der Start-Schlüssel pro Systempaarung gleich und somit angreifbar ist.

Der Zugang ist nur aus dem lokalen Netzwerk möglich, und somit nicht über WLAN oder andere Netzwerke erreichbar, was Broadcast-Netzwerke ausschließen sollte.

5.1.4 Man-in-the-Middle-Angriff

In dem MitMA kontrolliert der Angreifer das Netzwerk. Alle Daten (oder ein großer Teil) zwischen beiden Opfern werden über den Angreifer versendet. Der Angreifer kann somit die Daten vollständig mitlesen und verändern. Die Gegenmaßnahme gegen einen solchen Angreifer ist Verschlüsselung, Netzwerküberwachung und ein zweiter Kanal.

Zur Verschlüsselung ist dieselbe Aussage zu treffen wie in 5.1.3 Sniffing. Netzwerküberwachung kann durch die Software nicht geleistet werden und liegt bei den Administratoren des Netzwerkes. Ein zweiter Kanal für die Authentifizierung und Autorisierung von Vorgängen ist denkbar, aber in diesem System noch nicht umgesetzt. Als zweiter Kanal würde sich SMS oder Email anbieten, wobei SMS zu bevorzugen ist, da durch den MitMA auch die Emails mitgehört oder abgefangen werden könnten.

5.1.5 Bekannter Schlüssel

Der Schlüssel der angewandten Verschlüsselung kann „gebrochen“ oder „geklaut“ werden.

Ein Schlüssel einer unzureichenden Komplexität kann, wenn man eine verschlüsselte Nachricht hat und den Algorithmus kennt, ausprobiert und „gebrochen“ werden. Dieses Bruteforce-Schlüssel-Brechen kostet Zeit und Strom. Als Gegenmaßnahme bietet sich ein langer komplexer Schlüssel und häufiges Wechseln an.

Ein Schlüssel kann geklaut werden, indem der Angreifer Zugriff auf die Einstellungsdatei des anzugreifenden Teilsystems bekommt und die Einstellungsdatei entschlüsseln kann. Wenn man Sniffing mit „Schlüssel Brechen“ kombiniert, kann der Angreifer generell die Nachrichten abfangen, einen Schlüssel knacken und wenn der erste Schlüssel geknackt ist, die anderen Schlüssel, die übertragen werden, klauen.

Als Gegenmaßnahme bieten sich eine starke Verschlüsselung mit starkem Schlüssel, das Hinzufügen von SessionIds und SystemIds an. Die starke Verschlüsselung erhöht den Aufwand der nötig ist, bis der Schlüssel gebrochen ist. Die SessionId sorgt dafür, dass der Schlüssel nicht doppelt verwendet wird und die SystemId sorgt dafür, dass ein System eindeutig ist. Jede Nachricht bekommt eine SessionId, wenn eine SessionId doppelt ankommt, wird der Schlüssel ungültig und die Verbindung unterbrochen. Das doppelte Ankommen einer Nachricht sollte durch TCP/IP unterbunden sein und kann nur auf einen Fehler im System, Fehler im Netzwerk oder einen Angriff deuten. Alle diese Fälle sind zu unterbinden. Dieselbe Idee, wie hinter der SessionId, steckt hinter der SystemId. Jedes System kann sich nur einmal mit dem Central-Server verbinden, bzw. der Central-Server kann sich

nur einmal mit einem Satelliten-Client verbinden. Zusätzlich kennen die Satelliten-Clients die IP des Central-Servers und lassen nur Verbindungen von der IP-Adresse des Central-Servers mit der richtigen SystemId zu. Wenn sich am Central-Server doch zwei Systeme mit derselben SystemId anmelden, wird die Verbindung unterbrochen und der Schlüssel gelöscht.

5.2 Verschlüsselung und Login

Verschlüsselung ist laut 5.1 Risiken eine wichtige Maßnahme und im Gesamtsystem ein großer Bestandteil. Es werden die Einstellungen als setup.cry und die Netzwerkübertragungen verschlüsselt.

5.2.1 Verschlüsselte Einstellungen

Um das unbefugte Lesen von Einstellungen zu unterbinden und damit das Risiko zu verringern, dass Schlüssel und andere kritische Daten ausgelesen werden sind die Einstellungen verschlüsselt.

Die Einstellungen werden nach den Vorgaben, die in Kapitel 4.2 Einstellungen festgelegt wurden in verschlüsselte XML-Dateien gespeichert. Die Verschlüsselung geschieht durch symmetrische Verschlüsselungsverfahren. Die Einstellungsdateien werden durch das Setup-Tool und zusammen mit dem Central-Server erstellt als setup.cry-Datei gespeichert.

Die Einstellungen für den Central-Server, werden mit dem Setup-Tool direkt festgelegt. Für die Einstellungen der Client-Systeme werden, die entsprechenden Daten zusätzlich in der Datenbank des Central-Servers gespeichert. Durch das Speichern der Daten in dem Central-Server wird zum einen die Verschlüsselung und die Identifizierung der Client-Systeme ermöglicht und zum anderen wird die Benutzbarkeit erhöt, da sich die Einstellungen als Grundlage für erneutes Ausstellen der Einstellungen verwenden lassen.

Die setup.cry Datei wird in dem Ordner „config“ zum jeweiligen System hinterlegt. Wichtig ist, dass das Verschlüsselungsverfahren des Erstellers und des Lesers übereinstimmen. Das Plugin zur Verschlüsselung muss in den lesenden Systemen unter „plugin/crypt/default“ liegen. Der Speicherort des Plugins ist wichtig, damit das System ohne, die Einstellungen geladen zu haben, das Plugin benutzen kann, um die Einstellungen zu entschlüsseln.

5.2.2 Verschlüsselte Übertragung

Die Verschlüsselung geschieht über symmetrisch verschlüsseltes XML mit vorangestellter systemId zur Identifizierung des sendenden Systems. Die **SystemId** ist im Gesamtsystem als long abgespeichert, damit ist die größte erreichbare Zahl (unter Java 7) 9223372036854775807 und somit als 19 stelliger String speicherbar.

```
<xs:complexType name="SessionSet">
      <xs:attribute name="nextSessionId" type="xs:string"/>
      <xs:attribute name="nextCryptKey" type="xs:string"/>
</xs:complexType>
```

Abbildung 45: Ausschnitt der XSD-Datei SessionSet

Jede festgelegte Verbindung hat einen im Central-Server und im Client hinterlegten **`cryptKey`** und **`SessionId`**. Der hinterlegte Key ist eindeutig der **`systemId`** zugeordnet. Der Key für die erste Verbindung ist der hinterlegte **`cryptKey`**, danach liefert der Central-Server bei jeder Verbindung ein neues **`SessionSet`**, wie in Abbildung 45 zusehen, bestehend aus **`cryptKey`** und **`sessionId`** aus.

5.2.3 Login Verfahren

Das Login Verfahren besteht aus zwei bis drei Schritten, die aufeinander aufbauen. Ziel ist es die Einstellungen geschützt zu halten, das System und, wenn vorhanden, den User zu identifizieren.

- Start-Login

  Der erste Schritt bei allen Teilsystemen ist, nach dem die Einstellungen getätigt sind, das Starten. Das Starten aller Systeme erfordert die Eingabe des Start-Passwortes, mit dem die Einstellungen geladen werden und entschlüsselt werden.

- System-Login

  Mit Hilfe der Einstellungen, die durch den Start-Login entschlüsselt wurden, meldet sich der Client automatisch beim Central-Server an.

- User-Login

  Der User-Login existiert nur beim GUI-Client und beim Setup-Tool. Nach dem System-Login wird der User-Login freigegeben, der den User identifiziert. Für die Identifizierung des Users gibt dieser Username und Passwort an.

6 Weiterentwicklung

Die Weiterentwicklung, Anpassung und Wartung von Softwaresystemen sind für den langfristigen Betrieb eines Software-Systems essentiell. Um die Anpassung und Weiterentwicklung der Software zu ermöglichen, gibt es zwei Möglichkeiten. Entweder man verändert das Kernsystem oder die Plugins. Spätestens, wenn es um die Anpassung an eine neue Runtime Environment geht, muss das Kernsystem verändert werden. Kleinere Anpassungen dagegen, die nicht durch die Alterung des Systems hervorgerufen sind, können mit Plugins gelöst werden; Zum Beispiel ist das Wechseln des Cryptsystems oder des DBMS möglich ohne das Kernsystem zu verändern.

Zur Weiterentwicklung gehört im nächsten Schritt auch die Entwicklung der Satelliten und GUI-Clients. Dafür gibt es zwei Möglichkeiten. Entweder auf der Basis des Servers ein Gerüst bauen, das Login, Sessionhandling und das Parsen von XML auf Java-Objekte übernimmt oder ein neues System erstllen, welches nur die Schnittstellen zum Server bedient. Die Entscheidung, welche der beiden Varianten für die Entwicklung der beiden Clients verwendet wird, liegt in der Zukunft. Es sollte berücksichtigt werden, dass die Wiederverwendbarkeit von Erzeugnissen einen großen Vorteil bieten kann. Der größte Vorteil liegt darin, die Entwicklung parallel durchzuführen und Synergien zu nutzen, um Module nicht doppelt implementieren zu müssen.

6.1 Wiederverwendbarkeit

Wiederverwendbarkeit ist eines der Ziele der modernen Softwareentwicklung. Wie oben in Kapitel 6. Weiterentwicklung erwähnt wurde, kann man ein Gerüst auf Basis von Teilen des Server bauen, um die Implementation der Clients zu vereinfachen.

Die Pakete des Servers, welche sich auch für die Entwicklung der Clients eignen, sind die folgenden:

Pakete die mit minimalen Änderungen verwendet werden können:

- **server** ist ein Paket, das bereits aus einem anderen Projekt übernommen wurde und hält die Funktionalität eines Socket-Clients und Servers.
- Das Paket **model.generated** ist aus der XSD-generierbar und stellt somit einen festen Bestandteil der Kommunikation dar. Über diese Klassen ist die komplette Sprache zwischen Client und Server abbildbar.
- **crypt** ist ein Paket und als Cryptsystem ein Plugin, welches auf den Systemen, die miteinander kommunizieren, verwendet werden muss. Da unabhängig von der Rolle im System beide Seiten ver- und entschlüsseln, ist dieses Paket oder Plugin übertragbar.
- Das Paket **tools** hält Klassen, die durch viele Pakete im Server verwendet werden. Unter anderem befindet sich dort die Definition des Queue-Modells und der JaxbMarschaller.

Folgende Pakete sind nur in Kombination verwendbar und stellen Anforderungen an das System, um genutzt zu werden.

- Das Paket **db** bietet eine Schnittstelle der Anbindung einer Datenbank mit Verbindung an das `Papared-System`. Wenn man dieses Paket einsetzen möchte muss man das `Prepared-System` einsetzen. Des Weiteren hält dieses Paket eine eigene Queue. Wenn das Queue-Modell sich nicht für den zu implementierenden Client geeignet sein sollte, müsste das ebenfalls geändert werden.

- **db.mySql** ist ein Paket bzw. ein Plugin, welches an das DBMS MySQL angepasst ist. Wenn der Client ein Dritt-System anbindet, welches über eine MySQL-Datenbank verfügt, kann dieser Teil des Servers als Paket oder Plugin verwendet werden.

- Das Paket **model.join** bildet die Relationsstruktur des Datenbank-Modells ab. In diesem Fall bildet es die Datenbank-Struktur des Servers ab. Wenn das anzubindende Dritt-System über eine Relationale-Datenbank verfügt, die den Voraussetzungen im Kapitel 4.3.5.1 genügt, kann dieses Model verwendet werden, muss aber auf die Datenbank des Dritt-System angepasst werden. (mehr Information dazu im Kapitel: 4.3.5)

- **model** ist ein Paket, was sich stark an dem WModel orientiert, welches sich in dem Paket befindet. Wenn man das Paket **model** nutzen möchte, sollte man das Modell WModel auf das anzubindende Dritt-System anwenden. Das WModel ist nur in Verbindung mit dem `model.generated` und dem `model.join` anwendbar. (mehr Information dazu im Kapitel: 4.3.5)

Die Clients sind in Satelliten-Client und GUI-Client zu unterteilen. Die Clients können voraussichtlich die Pakete `server`, `model.generated`, `crypt` und `tools` verwenden. Der GUI-Client ist mit diesen Pakten gut ausgestattet, denn er braucht nur Messages bauen, empfangen und darstellen. Der Satelliten-Client kann darüber hinaus, je nach angebunden Dritt-System, die Pakete `db`, `db.mySql`, `model.join` und `model` zusätzlich verwenden.

6.2 Commands

Zur Verdeutlichung der Commands, die durch die XSD bereits generell beschrieben sind, stelle ich drei Grund-Commands und ein Login vor und wie diese in Java und XML aussehen könnten.

6.2.1 Message Rumpf

Bevor die Commands und der Login erstellt werden können, muss der Rumpf der Message stehen. Im Abbildung 46 sieht man eine Möglichkeit den Rumpf einer Message zu erstellen.

```
ObjectFactory oFac = new ObjectFactory();
Message mes = oFac.createMessage();

mes.setSessionId(this.lastSessionId);
mes.setFromSenderId(this.mySenderId);

mes.setSystem(oFac.createSystem());
mes.getSystem().setIdOut(1l);
```

Abbildung 46: Quelltext zum Erstellen des Message-Rumpfs

Zunächst wird über die **ObjectFactory** ein Objekt vom Typ **Message** erstellt. Dieses Objekt wird mit den weiteren Informationen ergänzt; die letzte **SessionId**, die eigene **SenderId** und das Ziel in Form von dem Angegeben **System**.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message fromSenderId="542" sessionId="64ad6a4f616">
      <System idOut="1" />
</Message>
```

Abbildung 47: XML-Ausschnitt Message-Rumpf

Wie das daraus entstehende XML aussehen würde, ist in Abbildung 47 zusehen. Dieser XML-Ausschnitt ist jedoch laut definiertem XSD nicht vollständig und benötigt noch ein **Command**, **Login**, **Error** oder **Data**.

6.2.2 Login

Für das Erstellen des Logins ist der Rumpf der Message wie in 6.2.1 beschrieben notwendig. Es wird davon ausgegangen, dass diese bereits korrekt erstellt und über die Variable **mes** verwendbar ist.

```
mes.setLogin(oFac.createLogin());
mes.getLogin().setUser(oFac.createUser());

mes.getLogin().getUser().setId(this.getMyUserId());
mes.getLogin().getUser().setPasswort(this.getMyUserPasswort());
```

Abbildung 48: Quelltext zum Erstellen des Login

Der **Login** wird erstellt, indem der **Message** ein **Login**-Objekt hinzugefügt wird. In Abbildung 48 ist ein User-Login zu sehen. Für den User-Login wird dem **Login**-Objekt ein **User**-Objekt gegeben, der zur Identifizierung die **Id** und zur Verifizierung ein **Passwort** enthält. Das XML zum User-Login ist in Abbildung 49 zusehen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message fromSenderId="542" sessionId="64ad6a4f616">
	<System idOut="1" />
	<Login>
		<User id="9329" passwort="sichersPasswort;)" />
	</Login>
</Message>
```

Abbildung 49: XML eines User-Login

6.2.3 Get Command

Wie bei dem Login in Kapitel 6.2.2 zuvor ist der Message-Rumpf die Ausgangslage und der Command wird hinzugegeben.

```
	mes.setCommand(oFac.createCommand());
	mes.getCommand().setCommandType(CommandType.SELECT);
	mes.getCommand().setGet(oFac.createGet());

	mes.getCommand().getGet().setPerson(oFac.createPerson());
	mes.getCommand().getGet().getPerson().setVorname("Peter");
	mes.getCommand().getGet().getPerson().setNachname("Tester");

	mes.getCommand().getGet().setUser(oFac.createUser());
	mes.getCommand().getGet().getUser().setUsername("PeTester");
```

Abbildung 50: Quelltext zum Erstellen eines Get Commands

Der Get Command wird erstellt, indem der **Message** ein **Command**-Objekt hinzugegeben wird und dieses Command-Objekt als **Select** bzw. **Get**, über den **CommandType**, typisiert wird. Danach wird dem Command ein **Get**-Objekt hinzugefügt, in dem das **Get** formuliert wird.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message fromSenderId="542" sessionId="64ad6a4f616">
	<System idOut="1" />
	<Command commandType="SELECT">
		<Get>
			<Person vorname="Peter" nachname="Tester" />
			<User username="PeTester" />
		</Get>
	</Command>
</Message>
```

Abbildung 51: XML eines Get Command

Das Formulieren des **Get** wird durch Setzen von Daten-Objekten durchgeführt. In diesem Fall durch das Objekt **Person** und **User**, mit jeweiligen Attributen, die hier als Beispiel gesetzt wurden. Mehre Daten-Objekte in einem **Select** unterliegen einem impliziten Join. Durch den Join wird eine **Person** mit **Vorname** „Peter“, **Nachname** „Tester“ einem **User** mit **Usernamen** „PeTester“ gesucht. Das aus dem **Get** generiete XML in Abbildung 51 zusehen.

6.2.4 Add Command

Wie in den Kapiteln zuvor, ist der Message Rumpf die Ausgangslage des Commands.

```
mes.setCommand(oFac.createCommand());
mes.getCommand().setCommandType(CommandType.INSERT);

One add = oFac.createOne();
add.setPerson(oFac.createPerson());
add.getPerson().setAnrede(Anrede.FRAU);
add.getPerson().setNachname("Mustermann");
add.getPerson().setVorname("Kristina");

mes.getCommand().getAdd().add(add);
```

Abbildung 52: Quelltext zum Erstellen eines Add Commands

Der Add Command wird erstellt, indem man der **Message** ein **Command**-Objekt mit dem **CommandType INSERT** und einem oder mehrere **One**-Objekte hinzufügt. Das **Command** hält eine Liste in welche die **One**-Objekte hinzufügt werden. Deswegen wird in Abbildung 52 das **One** in einer eigenen Variable instanziiert und nach der Konfiguration der Liste hinzugefügt. Die gesetzten Attribute sind nur ein Teil der möglichen Attribute, die man bei einer neuen Person anlegen sollte und dienen als Beispiel. In Abbildung 53 ist der XML-Ausschnitt des der Add Commands zu sehen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message fromSenderId="542" sessionId="64ad6a4f616">
      <System idOut="1" />
      <Command commandType="INSERT">
            <Add>
                  <Person anrede="Frau" vorname="Kristina"
                  nachname="Mustermann" />
            </Add>
      </Command>
</Message>
```

Abbildung 53: XML-Ausschnitt eines Add Commands

6.2.5 Edit Command

Wie zuvor, ist auch für den Edit Command die Grundlange der Message-Rumpf.

```
mes.setCommand(oFac.createCommand());
mes.getCommand().setCommandType(CommandType.UPDATE);

Edit edit = oFac.createEdit();
edit.setSet(oFac.createOne());
edit.getSet().setPerson(oFac.createPerson());
edit.getSet().getPerson().setEmail("Muster@email.de");

edit.setWhere(oFac.createOne());
edit.getWhere().setPerson(oFac.createPerson());
edit.getWhere().getPerson().setId(83939l);

mes.getCommand().getEdit().add(edit);
```

Abbildung 54: Quelltext zum Erstellen eines Edit Command

Für das Edit Command muss der **CommandType** auf **UPDATE** gesetzt werden und ein oder mehre **Edit**-Objekte gesetzt werden. In einem **Command** können mehre **Edit**-Objekte gesetzt werden, deswegen wird in Abbildung 54 das **Edit**-Objekt in einer eigenen Variable vorbereitet und dann der Liste hinzugefügt. Ein **Edit**-Objekt besteht aus **Set**-Part und **Get**-Part, die jeweils **One**-Objekte sind. In dem **One**-Objekt des **Set**-Parts wird die Änderung definiert und in **One**-Objekt des **Where**-Parts die Identifizierung des zu ändernden Eintrags. Wenn der Eintrag mit einer **Id** zu identifizieren ist, ist diese Variante zu bevorzugen. Ein Beispiel für das XML eines Edit Commands ist in Abbildung 55 zu sehen.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Message fromSenderId="542" sessionId="64ad6a4f616">
	<System idOut="1" />
	<Command commandType="UPDATE">
		<Edit>
			<Set>
				<Person email="Muster@email.de" />
			</Set>
			<Where>
				<Person id="83939" />
			</Where>
		</Edit>
	</Command>
</Message>
```

Abbildung 55: XML eines Add Commands

7 Schlussbemerkung

In dieser Bachelorarbeit wurde ein System zur zentralen Benutzerverwaltung von dezentralen Benutzerdatenbanken mit Fokus auf Wartbar-, Erweiterbarkeit und Sicherheit entwickelt.

Im ersten Teil der Arbeit wurden Stakeholder, Anwendungsfälle und Anforderungen des Gesamtsystems gesammelt und konkretisiert. Es stellte sich schnell heraus, dass eine Teilung in vier Systeme sinnvoll ist. Die Anwendungsfälle und Anforderungen wurden somit für diese vier Systeme separat formuliert.

Im zweiten Teil der Arbeit schloss sich eine Technologiewahl an, in dem der technologische Rahmen definiert wurde. Basierend auf der Anforderungsanalyse und der Technologiewahl wurde der Softwareentwurf „top-down“ durchgeführt und beschrieben. Bei der Granulierung hat sich herausgestellt, dass das Beschreiben des Softwareentwurfs aller vier Bestandteile des Gesamtsystems zu umfangreich ist. Mit dieser Erkenntnis wurde der Central-Server, der das „Herz-Stück“ des Gesamtsystems darstellt, in den Fokus gerückt und die anderen Systeme nicht genauer beschrieben.

Auf den Softwareentwurf folgte eine Betrachtung der Sicherheit, in der Risiken und deren Maßnahmen beschrieben wurden. Ein Großteil dieser Maßnahmen findet sich in dem Softwareentwurf bereits wieder oder lässt sich mit geeignetem Aufwand nachrüsten.

Um die Weiterentwicklung und die Wiederverwendbarkeit der Bestandteile des Central-Servers zu beschreiben, schloss sich der Betrachtung der Sicherheit ein entsprechendes Kapitel zur Wiederverwendbarkeit an. In diesem Kapitel wurde die Wiederverwendbarkeit der Bestandteile auf Paketgröße beschrieben und Commads der auf XML basierenden „Kommunikations-Sprache“ beispielhaft erklärt. Viele der durch die modellgetriebene Entwicklung entstanden Pakete lassen sich auch für die Clients wiederverwenden.

7.1 Weiterentwicklung

Bei der nachträglichen kritischen Betrachtung eines Systems fallen immer Möglichkeiten zur Weiterentwicklung auf. Genauso stellen sich auch in dieser Arbeit Möglichkeiten zu weiteren Betrachtung heraus.

Möglichkeiten in Bereich der Sicherheit

- Entwicklung der Anbindung unterschiedlicher Verschlüsselungsverfahren pro Client. Um dies zu ermöglichen sind kleine Änderungen im Kern-System des Central-Servers nötig. Die Änderung würde jedoch die Verschlüsselung der Kommunikation stärken, da Angreifer sowohl den Schlüssel als auch das Verfahren herausfinden müssten.

- Dem vorherigen Punkt angeschlossen ist das Entwickeln eines anderen Verschlüsselungsverfahrens. Dies ist durch Austausch des Cryptsystems noch vor Inbetriebnahme möglich.

Möglichkeiten im Bereich der kontinuierlichen Entwicklung

- Entwicklung eines Installationsverfahrens, das selbständig Datenbanktabellen anlegt. Dies ist durch Erweiterung des WModels und JModels möglich, da nötige Informationen bereits in den Models enthalten sind.

- Für die weitere Entwicklung bietet es sich an, wie in der Arbeit bereits erwähnt, ein Referenz Design für GUI-Client und Satelliten-Client zu implementieren.

- Während der Betrachtung der Weiterentwicklung ist das Paket Model aufgefallen, welches das größte unter den Paketen im Central-Server ist. Hier ist eine genauere Aufteilung, für die Übersicht und die Wiederverwendbarkeit, von Vorteil.

- Ein Punkt für spätere Weiterentwicklung ist, dass im Paket „db“ mehr Funktionalität ausgelagert werden könnte, um die Zahl der Interfaces und den Aufwand, ein anderes DBMS anzubinden, verringert wird.

- Der kritischste Punkt ist, dass das Löschen von Daten in der Datenbank über XML noch nicht korrekt möglich ist. Dies ist durch kleine Eingriffe in den Central-Server und die XSD-Datei möglich, da sich ein DELETE Command genauso abbilden lässt wie ein INSERT Command.

7.2 Ausblick

Im Rückblick auf den Verlauf der Bachelorarbeit lässt sich sagen, dass man Erweiterbarkeit, Wartbarkeit und potentielle Sicherheit durch Einsatz von leicht erhöhter Entwicklungszeit erkauft hat. Diese eingesetzte Zeit sollte sich jedoch positiv auf die Lebensdauer des Gesamtsystems und Flexibilität im Produktionszeitraum auswirken.

Als nächster Schritt ist der Delete Command einzupflegen und der Central-Server zu Ende zu implementieren. Danach oder parallel ist ein Referenz-Model für jeweils den GUI-Client und den Satelliten-Client zu implementieren. Für die Entwicklung des Satelliten-Clients lohnt es sich den Moodle-Client dem BSCW-Client vorzuziehen, da die Datenbank-Anbindung genutzt werden kann.

7.3 Fazit

Im Rahmen dieser Bachelorarbeit wurde der Softwareentwurf nicht für alle Teile des Gesamtsystems abgeschlossen. Der Fokus wurde auf den Central-Server verlagert, der den Hauptteil des Systems ausmacht, und dieser konkret beschrieben. Trotz Verlagerung sind, durch die modellnahe Entwicklung und Kapselung, Bestandteile erstellt worden, die bei der Entwicklung der anderen System helfen werden.

Als Autor hätte ich noch gerne mehr Diagramme zum dynamischen Verhalten des Systems erstellt, sowie die anderen System-Teile beschrieben. Diese Bachelorarbeit bietet trotz der noch bevorstehenden Arbeit eine gute Grundlage, um das Gesamtsystem vollständig zu implementieren und in den Betrieb zu gehen.

Ich freue mich darauf, auch nach der Bachelorarbeit dieses Software-System vollständig zu implementieren.

8 Anhang

8.1 XSD-Message

Die folgenden XSD-Diagramme wurden aus dem XSD-Source generiert.

8.1.1 XSD-Source

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="de.mathias.kohs.cusms"
elementFormDefault="qualified" attributeFormDefault="unqualified">

    <xs:simpleType name="ErrorType">
        <xs:restriction base="xs:string">
            <xs:enumeration value="ServerError"/>
            <xs:enumeration value="CommandBroken"/>
            <xs:enumeration value="NoSystemLogin"/>
            <xs:enumeration value="NoUserLogin"/>
            <xs:enumeration value="CommandAttributeLost"/>
            <xs:enumeration value="XmlBroken"/>
            <xs:enumeration value="DataToOld"/>
            <xs:enumeration value="VersionToOld"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="SystemType" final="restriction">
        <xs:restriction base="xs:string">
            <xs:enumeration value="GUI" />
            <xs:enumeration value="CENTRAL" />
            <xs:enumeration value="SATELLITE" />
        </xs:restriction>
    </xs:simpleType>

    <xs:simpleType name="CommandType" final="restriction">
        <xs:restriction base="xs:string">
            <xs:enumeration value="SELECT" />
            <xs:enumeration value="UPDATE" />
            <xs:enumeration value="INSERT" />
        </xs:restriction>
    </xs:simpleType>
```

```
<xs:simpleType name="Anrede" final="restriction">
	<xs:restriction base="xs:string">
		<xs:enumeration value="Herr" />
		<xs:enumeration value="Frau" />
	</xs:restriction>
</xs:simpleType>

<xs:simpleType name="VerbindungsType" final="restriction">
	<xs:restriction base="xs:string">
		<xs:enumeration value="BIND" />
		<xs:enumeration value="SEHR_STARK" />
		<xs:enumeration value="STARK" />
		<xs:enumeration value="MITTEL" />
		<xs:enumeration value="SCHWACH" />
		<xs:enumeration value="NONE" />
	</xs:restriction>
</xs:simpleType>

<xs:complexType name="SatellitType">
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="name" type="xs:string"/>
</xs:complexType>

<xs:complexType name="System">
	<xs:sequence>
		<xs:element name="satellitType" type="SatellitType"/>
	</xs:sequence>
	<xs:attribute name="systemType" type="SystemType"/>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="idOut" type="xs:long"/>
	<xs:attribute name="name" type="xs:string"/>
	<xs:attribute name="key" type="xs:base64Binary"/>
	<xs:attribute name="addresse" type="xs:string"/>
	<xs:attribute name="port" type="xs:long"/>
	<xs:attribute name="activeFrom" type="xs:long"/>
	<xs:attribute name="activeTo" type="xs:long"/>
</xs:complexType>
```

```
<xs:complexType name="Orgaunit">
	<xs:attribute name="name" type="xs:string"/>
	<xs:attribute name="id" type="xs:long"/>
</xs:complexType>

<xs:complexType name="Role">
	<xs:attribute name="name" type="xs:string"/>
	<xs:attribute name="id" type="xs:long"/>
</xs:complexType>

<xs:complexType name="Ort">
	<xs:attribute name="name" type="xs:string"/>
	<xs:attribute name="plz" type="xs:string"/>
	<xs:attribute name="id" type="xs:long"/>
</xs:complexType>

<xs:complexType name="Workspace">
	<xs:attribute name="name" type="xs:string"/>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="idOut" type="xs:long"/>
</xs:complexType>

<xs:complexType name="User">
	<xs:sequence>
		<xs:element name="Einschreibung" type="Einschreibung" maxOccurs="unbounded" minOccurs="0"/>
		<xs:element name="VerbindungsType" type="VerbindungsType" maxOccurs="1" minOccurs="0"/>

	</xs:sequence>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="idOut" type="xs:long"/>
	<xs:attribute name="username" type="xs:string"/>
	<xs:attribute name="vorname" type="xs:string"/>
	<xs:attribute name="nachname" type="xs:string"/>
	<xs:attribute name="passwort" type="xs:string"/>
	<xs:attribute name="email" type="xs:string"/>
</xs:complexType>
```

```
<xs:complexType name="Person">
	<xs:sequence>
		<xs:element name="User" type="User" maxOccurs="unbounded" minOccurs="0"/>
		<xs:element name="Ort" type="Ort" maxOccurs="1" minOccurs="0"/>
	</xs:sequence>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="anrede" type="Anrede"/>
	<xs:attribute name="vorname" type="xs:string"/>
	<xs:attribute name="nachname" type="xs:string"/>
	<xs:attribute name="email" type="xs:string"/>
	<xs:attribute name="telefon" type="xs:string"/>
	<xs:attribute name="strasse" type="xs:string"/>
	<xs:attribute name="hausnr" type="xs:string"/>
</xs:complexType>

<xs:complexType name="Einschreibung">
	<xs:sequence>
		<xs:element name="Role" type="Role"/>
		<xs:element name="Workspace" type="Workspace"/>
		<xs:element name="Satellit" type="System"/>
		<!-- xs:element name="User" type="User"/> -->
	</xs:sequence>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="timeFrom" type="xs:long"/>
	<xs:attribute name="timeTo" type="xs:long"/>
	<xs:attribute name="imSatellit" type="xs:boolean"/>
	<xs:attribute name="einschreibung" type="xs:boolean"/>
</xs:complexType>
```

```
<xs:complexType name="PersonOrga">
	<xs:sequence>
		<xs:element name="Person" type="Person"/>
		<xs:element name="OrgaUnit" type="Orgaunit"/>
	</xs:sequence>
	<xs:attribute name="id" type="xs:long"/>
	<xs:attribute name="ansprechpartner" type="xs:boolean"/>
</xs:complexType>

<xs:complexType name="Login">
	<xs:choice>
		<xs:element name="User" type="User"/>
		<xs:element name="System" type="System"/>
	</xs:choice>
</xs:complexType>

<xs:complexType name="SessionSet">
	<xs:attribute name="nextSessionId" type="xs:string"/>
	<xs:attribute name="nextCryptKey" type="xs:string"/>
</xs:complexType>

<xs:complexType name="Error">
	<xs:attribute name="ErrorType" type="ErrorType"/>
	<xs:attribute name="text" type="xs:string"/>
</xs:complexType>
```

```
<xs:complexType name="Get">
	<xs:sequence>
		<xs:element name="Person" type="Person" maxOccurs="1" minOccurs="0"/>
		<xs:element name="User" type="User" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Workspace" type="Workspace" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Role" type="Role" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Satellit" type="System" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Ort" type="Ort" maxOccurs="1" minOccurs="0"/>
		<xs:element name="PersonOrga" type="PersonOrga" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Orgaunit" type="Orgaunit" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Einschreibung" type="Einschreibung" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Where" type="Where" maxOccurs="unbounded" minOccurs="1"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="Edit">
	<xs:sequence>
		<xs:element name="Set" type="One" maxOccurs="1" minOccurs="1"/>
		<xs:element name="Where" type="One" maxOccurs="1" minOccurs="1"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="One">
	<xs:choice>
		<xs:element name="Person" type="Person"/>
		<xs:element name="User" type="User"/>
		<xs:element name="Workspace" type="Workspace"/>
		<xs:element name="Role" type="Role"/>
		<xs:element name="Satellit" type="System"/>
		<xs:element name="Ort" type="Ort"/>
		<xs:element name="PersonOrga" type="PersonOrga"/>
		<xs:element name="Orgaunit" type="Orgaunit"/>
		<xs:element name="Einschreibung" type="Einschreibung"/>
	</xs:choice>
</xs:complexType
```

```
<xs:complexType name="Where">
    <xs:sequence>
        <xs:element name="Person" type="Person" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="User" type="User" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Workspace" type="Workspace" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Role" type="Role" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Satellit" type="System" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Ort" type="Ort" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="PersonOrga" type="PersonOrga" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Orgaunit" type="Orgaunit" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Einschreibung" type="Einschreibung" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="Command">
    <xs:choice>
        <xs:element name="Get" type="Get" maxOccurs="1" minOccurs="0"/>
        <xs:element name="Edit" type="Edit" maxOccurs="unbounded" minOccurs="0"/>
        <xs:element name="Add" type="One" maxOccurs="unbounded" minOccurs="0"/>
    </xs:choice>
    <xs:attribute name="commandType" type="CommandType"/>
</xs:complexType>
```

```
<xs:complexType name="DatenZeile">
	<xs:choice>
		<xs:element name="Person" type="Person" maxOccurs="1" minOccurs="0"/>
		<xs:element name="User" type="User" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Workspace" type="Workspace" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Role" type="Role" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Satellit" type="System" maxOccurs="1" minOccurs="0"/>
		<xs:element name="SatellitType" type="SatellitType" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Ort" type="Ort" maxOccurs="1" minOccurs="0"/>
		<xs:element name="PersonOrga" type="PersonOrga" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Orgaunit" type="Orgaunit" maxOccurs="1" minOccurs="0"/>
		<xs:element name="Einschreibung" type="Einschreibung" maxOccurs="1" minOccurs="0"/>
	</xs:choice>
</xs:complexType>

<xs:complexType name="ChangeProved">
	<xs:attribute name="index" type="xs:long"/>
	<xs:attribute name="generatedKey" type="xs:long"/>
</xs:complexType>

<xs:complexType name="Data">
	<xs:choice>
		<xs:element name="DatenZeile" type="DatenZeile" maxOccurs="unbounded" minOccurs="0"/>
		<xs:element name="ChangeProved" type="ChangeProved" maxOccurs="unbounded" minOccurs="0"/>
	</xs:choice>
	<xs:attribute name="timestamp" type="xs:long"/>
</xs:complexType>
```

```xml
<xs:element name="Message">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="System" type="System" maxOccurs="1" minOccurs="1"/> <!-- TODO CHECKEN
maxOccurs -->
            <xs:element name="SessionSet" type="SessionSet" maxOccurs="1" minOccurs="0"/>
                <xs:choice>
                    <xs:element name="Login" type="Login"/>
                    <xs:element name="Data" type="Data"/>
                    <xs:element name="Command" type="Command"/>
                    <xs:element name="Error" type="Error"/>
                </xs:choice>
            </xs:sequence>
            <xs:attribute name="fromSenderId" type="xs:long"/>
            <xs:attribute name="sessionId" type="xs:string"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

8.1.2 XSD-Diagramme

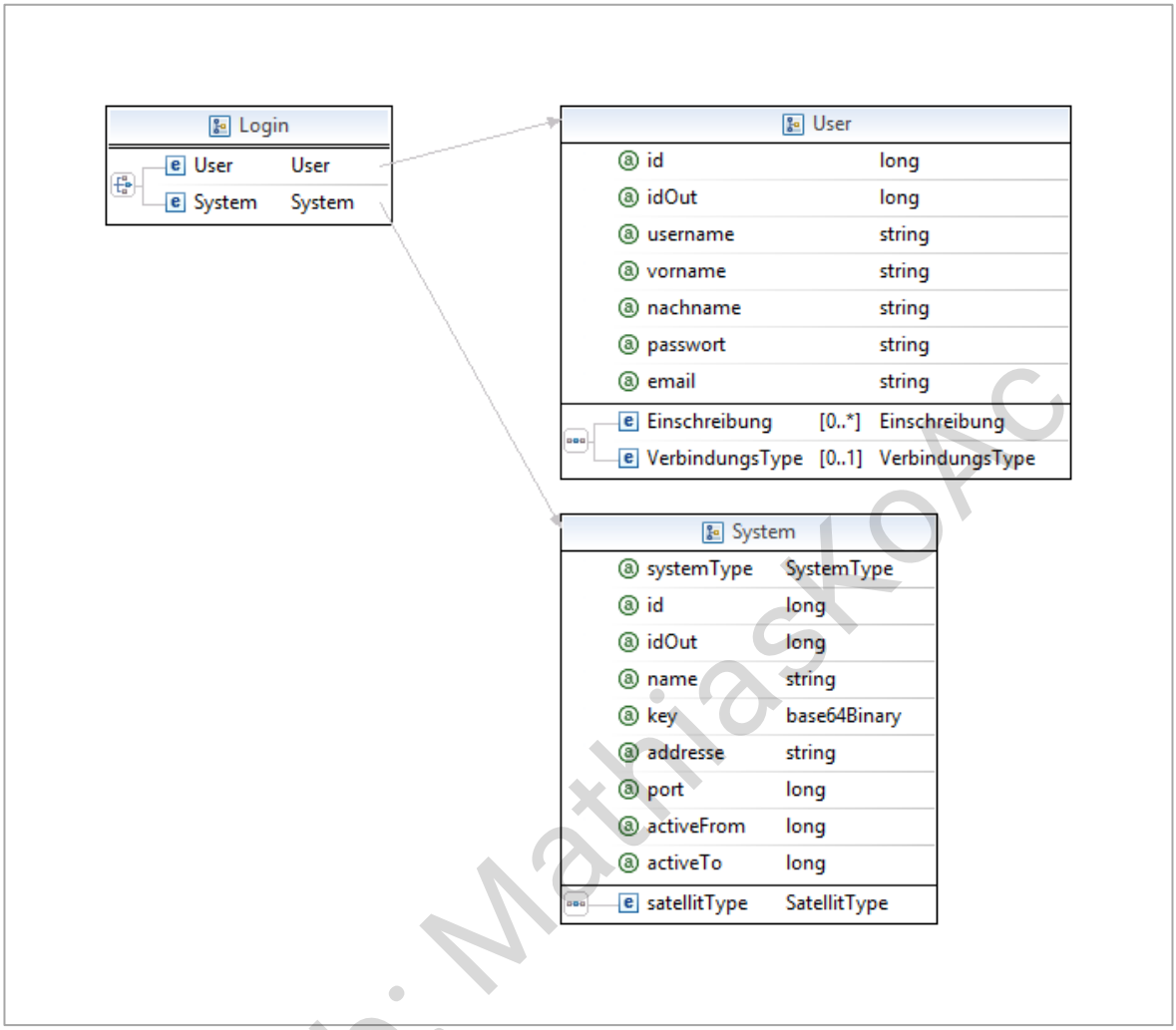

Abbildung 56: XSD-Diagramm Message Login

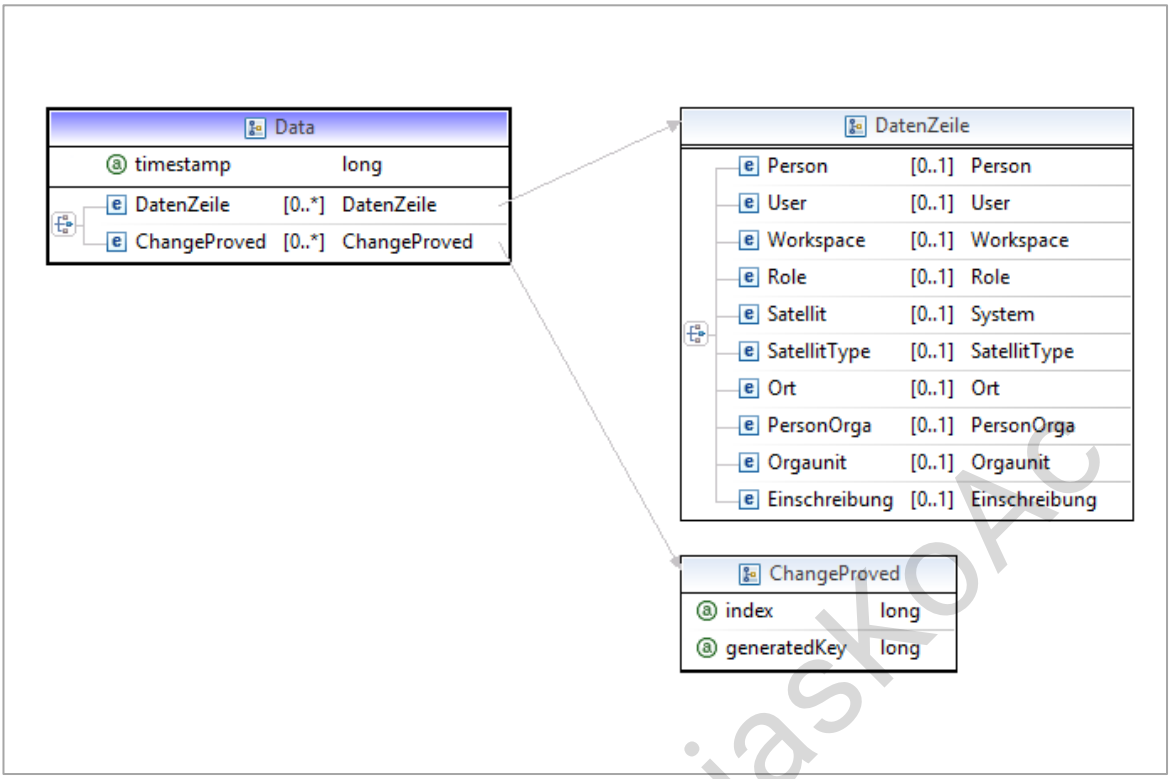

Abbildung 57: XSD-Diagramm Message Data

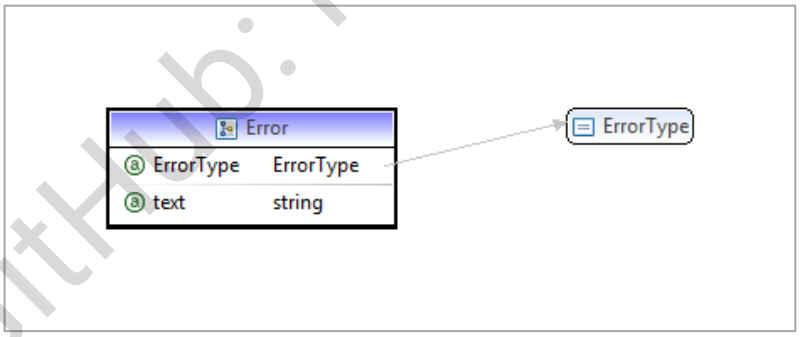

Abbildung 58: XSD-Diagramm Message Error

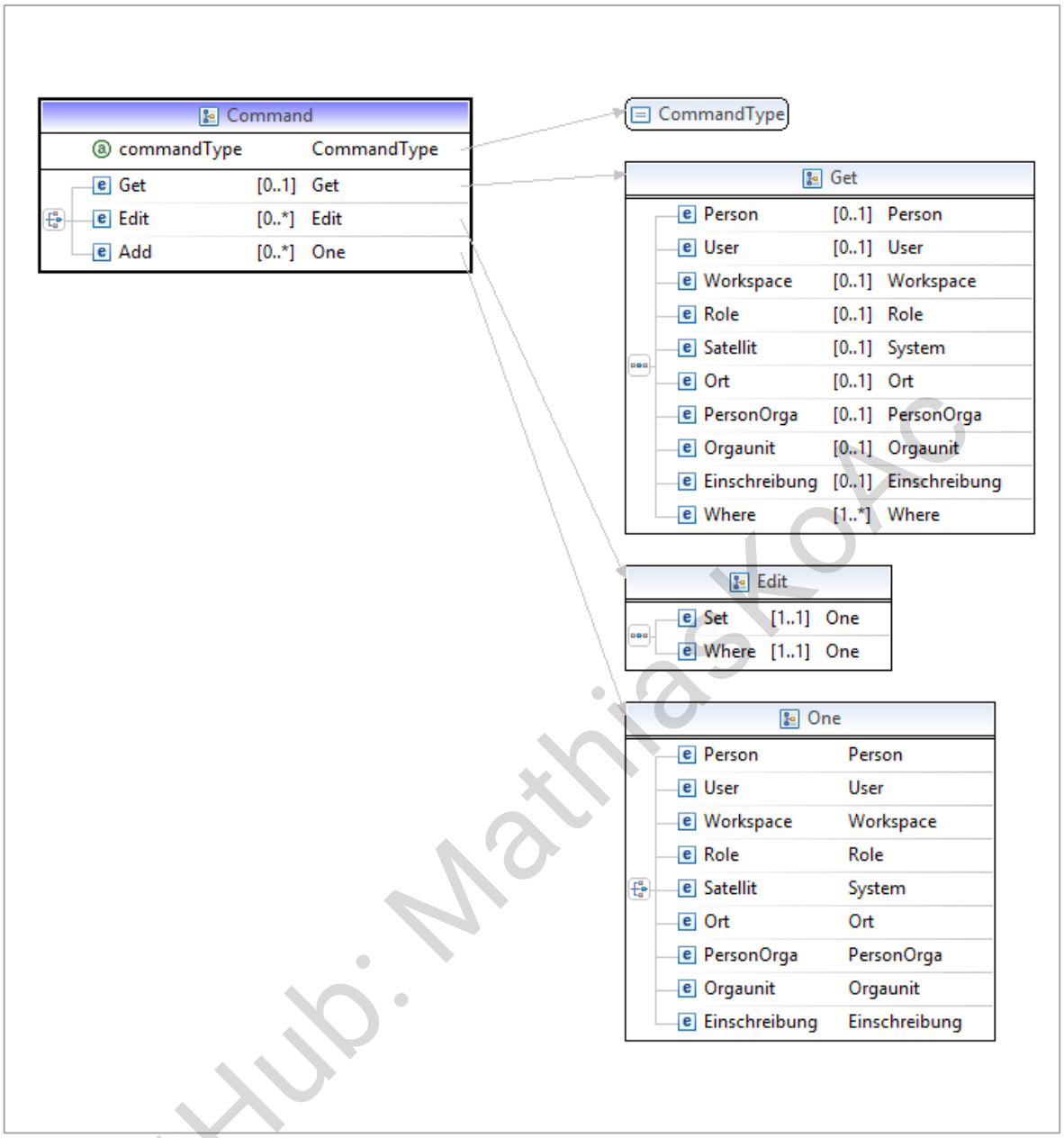

Abbildung 59: XSD-Diagramm Message Command

8.2 Alternativen des Datenbank-Modells

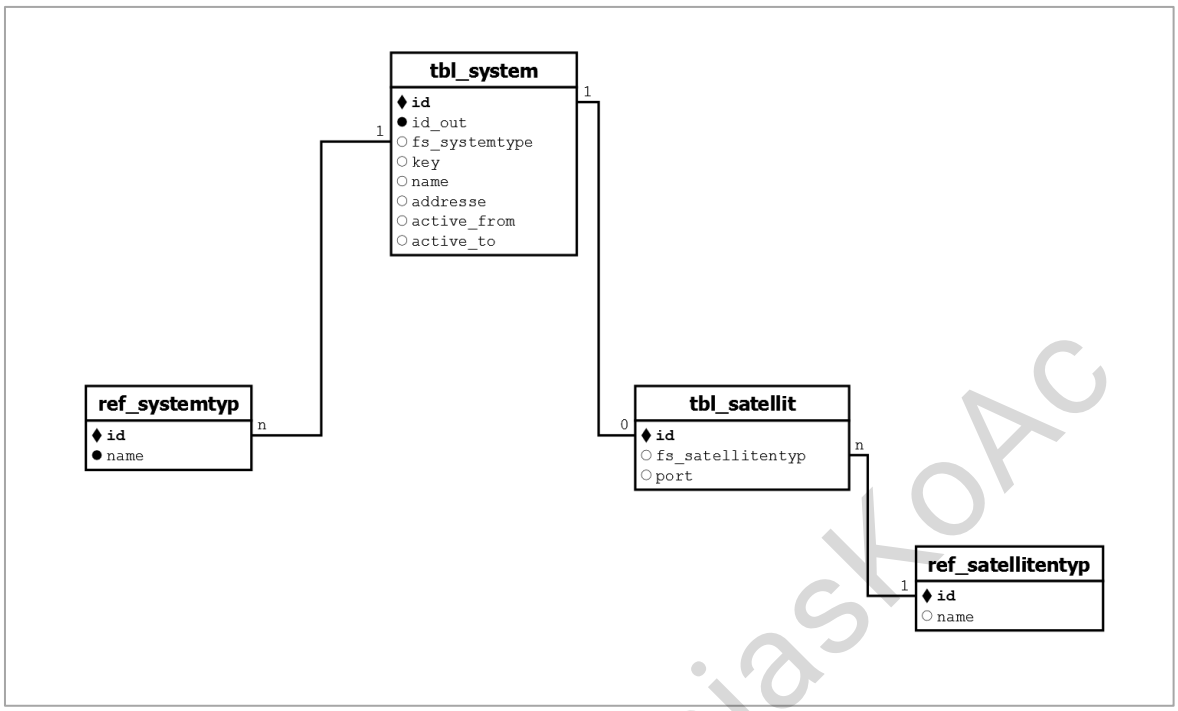

Abbildung 60: DB-Modell alternative Normalisierung von System

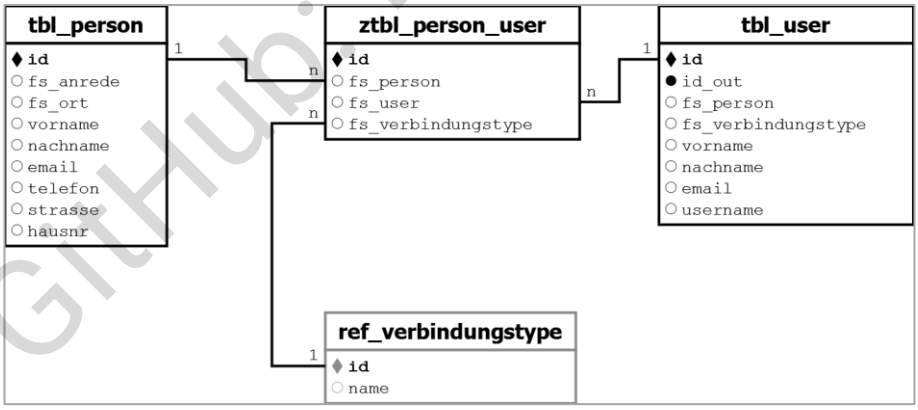

Abbildung 61: DB-Modell alternative Normalisierung Person User

8.3 Beschreibung der Datenbank Tabellen

Tabelle		
Name der Tabelle	tbl_rolle	
Verbindung in das ER-Diagramm	Entität Rolle	
Type im ER-Diagramm	Echte Entität	
Tabellentyp	Tabelle	
Kurzbeschreibung	Entspricht einer Rolle im Satellitensystem.	
Beispiel	User „A“ hat die Rolle „Administrator“ im Arbeitsbereich „Vorlesung Analysis2“.	
Verbindungen	- 1 zu N mit tbl_einschreibung	
Indirekte Verbindungen	- M zu N mit tbl_user - M zu N mit tbl_satellit - M zu N mit tbl_arbeitsbereich	
Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
name	varchar	

Tabelle 15: Beschreibung Entität Rolle

Tabelle		
Name der Tabelle	ref_systemtyp	
Verbindung in das ER-Diagramm	Entität Systemtyp	
Type im ER-Diagramm	Echte Entität	
Tabellentyp	Referenztabelle	
Kurzbeschreibung	System gehören dem Systemtyp Central-Server, GUI-Client oder Satellit an.	
Beispiel	„Hydrology-Moodle2.3“ ist ein System von dem Systemtyp Satellit.	
Verbindungen	- 1 zu N mit tbl_system	
Indirekte Verbindungen	- 1 zu N mit tbl_einschreibung	
Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
name	varchar	

Tabelle 16: Beschreibung Entität Systemtyp

Tabelle		
Name der Tabelle	tbl_satellitentyp	
Verbindung in das ER-Diagramm	(Entität Satellitentyp nicht im ER-Model)	
Type im ER-Diagramm	--	
Tabellentyp	Tabelle	
Kurzbeschreibung	Satelliten gehören einem Typ an, dieser Typ sagt aus, für welche Arten von Satellitenserver, diese Satelliten geeignet sind.	
Beispiel	Der Satellit der auf die Adresse 192.168.1.15 Port 2234 ist für die Verbindung mit einem Moodle2.3 zuständig, ist von dem Satellitentyp „Moodle2.3“.	
Verbindungen	- 1 zu N mit tbl_satellit	
Indirekte Verbindungen	- 1 zu N mit tbl_einschreibung	
Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
name	varchar	

Tabelle 17: Beschreibung Entität Satellitentyp

Tabelle	
Name der Tabelle	ref_verbindungstyp
Verbindung in das ER-Diagramm	Entität Verbindungstyp
Type im ER-Diagramm	Echte Entität
Tabellentyp	Referenztabelle
Kurzbeschreibung	Gibt die Art und Stärke der Verbindung zwischen Person und User an.
Beispiel	Automatisch generierte Verbindungen können den Typ, „schwach“, „mittel“, „stark“, „sehr stark“ haben. Eine durch den Endanwender bestätigte Verbindung hat den Typ „bestätigt“, eine abgelehnte Verbindung „ausgeschlossen“.
Verbindungen	- 1 zu N mit tbl_user
Indirekte Verbindungen	- 1 zu N mit tbl_einschreibung - M zu M mit tbl_person

Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
name	varchar	

Tabelle 18: Beschreibung Entität Verbindungstyp

Tabelle	
Name der Tabelle	tbl_ort
Verbindung in das ER-Diagramm	Entität Ort
Type im ER-Diagramm	Echte Entität
Tabellentyp	Tabelle
Kurzbeschreibung	Gibt den Ort an, als Verbindung von Name und Postleitzahl.
Beispiel	„Peter“ „Lustig“ wohnt im Ort mit dem Namen „Aachen“ und der plz 52074.
Verbindungen	- 1 zu N mit tbl_person
Indirekte Verbindungen	- 1 zu N mit tbl_user - 1 zu N mit ztbl_person_orgaunit

Attribute		
Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
name	varchar	

Tabelle 19: Beschreibung Entität Ort

Tabelle		
Name der Tabelle	tbl_workspace	
Verbindung in das ER-Diagramm	Entität Workspace	
Type im ER-Diagramm	Echte Entität	
Tabellentyp	Tabelle	
Kurzbeschreibung	Entspricht einem Workspace im Satellitensystem.	
Beispiel	Arbeitsbereich in einem BSCW-Server Kursinstanz in einem Moodle-Server	
Verbindungen	- 1 zu N mit tbl_einschreibung	
Indirekte Verbindungen	- M zu N mit tbl_user - M zu N mit tbl_satellit - M zu N mit ref_rolle	
Attribute		
Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
id_out	bigint	Externer Schlüssel
name	varchar	

Tabelle 20: Beschreibung Entität Workspace

Tabelle	
Name der Tabelle	tbl_einschreibung
Verbindung in das ER-Diagramm	schwache Entität Einschreibung
Type im ER-Diagramm	schwache Entität
Tabellentyp	Tabelle
Kurzbeschreibung	Ermöglicht das Einschreiben von User in einen Workspace mit einer Rolle auf einen Satelliten-Server.
Beispiel	User „PeterLustig“ ist in den Workspace „Vorlesung-Hydrologie1“ mit der Rolle „Teilnehmer“ auf dem Satelliten-Server „Hydrology-Moodle“ eingeschrieben.
Verbindungen	- N zu 1 mit ref_rolle - N zu 1 mit tbl_system - N zu 1 mit tbl_user - N zu 1 mit tbl_workspace
Indirekte Verbindungen	Nicht sinnvoll, da schwache Entität.

Attribute

Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
fs_rolle	int	Fremdschlüssel
fs_system	int	Fremdschlüssel
fs_workspace	bigint	Fremdschlüssel
fs_user	bigint	Fremdschlüssel
time_from	int	
time_to	int	
isimsatellit	tinyint / boolean	
iseinschreibung	tinyint / boolean	

Tabelle 21: Beschreibung Entität Einschreibung

Tabelle		
Name der Tabelle	tbl_orgaunit	
Verbindung in das ER-Diagramm	Enität OrganisationsUnit	
Type im ER-Diagramm	Echte Entität	
Tabellentyp	Tabelle	
Kurzbeschreibung	OrgaUntis bzw. Organisations-Einheiten sind Einheiten in die Personen eingetragen werden können, um diese unabhängig von der Struktur in den Satelliten zu verwalten (organisieren).	
Beispiel	In der OrgaUnit „AB-Control“ ist die Person „Peter“ „Lousberg“ eingetragen, obwohl in den Einschreibungen nichts Derartiges erkennbar ist.	
Verbindungen	- 1 zu N mit ztbl_person_orgaunit	
Indirekte Verbindungen	- M zu N mit tbl_person	
Attribute		
Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
name	varchar	

Tabelle 22: Beschreibung Entität OrganisationsUnit

Tabelle	
Name der Tabelle	tbl_person
Verbindung in das ER-Diagramm	Entität Person
Type im ER-Diagramm	Echte Entität
Tabellentyp	Tabelle
Kurzbeschreibung	Entspricht einer natürlichen Person in der realen Welt, die Userkonten haben kann und zu besseren Übersicht der Endbenutzer in OragnisationsUnits eingetragen werden kann. Die Tabelle gibt als denormalisiert, da die Kontaktinformationen nicht ausgelagert sind.
Beispiel	„Peter“ „Lousberg“ ist eine natürliche Persion und hat ein Useraccount mit Usernamen „PeterLustig“. „Peter“ ist ein Student und ein Mitarbeiter des Institutes „AB-Control“, also wurde er in die OrgranisationsUnit „AB-Control“ und „Student“ eingetragen.
Verbindungen	- 1 zu N mit tbl_user - 1 zu N mit ztbl_person_orgaunit - N zu 1 mit tbl_ort
Indirekte Verbindungen	- 1 zu N mit tbl_einschreibung - M zu N mit tbl_orgaunit

Attribute		
Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
fs_anrede	int	Fremdschlüssel
fs_ort	bigint	Fremdschlüssel
vorname	varchar	
nachname	varchar	
email	varchar	
telefon	int	
straße	varchar	
hausnr	varchar	

Tabelle 23: Beschreibung Entität Person

Tabelle	
Name der Tabelle	tbl_system
Verbindung in das ER-Diagramm	Enität System
Type im ER-Diagramm	Echte Entität
Tabellentyp	Tabelle
Kurzbeschreibung	Entspricht einem GUI-Client oder der Instanz eines Satelliten-Clients, der als Kopplung an ein Satelliten-Sever genutzt wird.
Beispiel	Der Satelliten-Client für die Verbindung zu dem Satelliten-Server Hydrology-Moodle 2.3.
Verbindungen	- 1 zu N mit tbl_einschreibung - N zu 1 mit tbl_satellitentyp - N zu 1 mit ref_systemtyp - N zu 1 mit ref_anrede
Indirekte Verbindungen	- M zu N mit tbl_user - M zu N mit tbl_arbeitsbereich - M zu N mit tbl_rolle - M zu N mit ref_user

Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
fs_satellitentyp	int	Fremdschlüssel
fs_systemtyp	int	Fremdschlüssel
name	varchar	
addresse	varchar	
port	int	
key	bigint	
time_from	int	
time_to	int	

Tabelle 24: Beschreibung Entität System

Tabelle	
Name der Tabelle	tbl_user
Verbindung in das ER-Diagramm	Entität User
Type im ER-Diagramm	Echte Entität
Tabellentyp	Tabelle
Kurzbeschreibung	Entspricht einem Userkonto auf einem Satelliten-Server, der mit einer natürlichen Person verbunden ist.
Beispiel	Die Natürliche Person „Peter“ „Lousberg“ hat ein Userkonto auf der Moodle-Instanz „Hdyrology-Moodle“ mit dem Username „PeterLustig“.
Verbindungen	- 1 zu N mit tbl_einschreibung - N zu 1 mit ref_verbindungstyp - N zu 1 mit tbl_person
Indirekte Verbindungen	- M zu N mit tbl_arbeitsbereich - M zu N mit tbl_system - M zu N mit tbl_rolle

Attribute		
Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
fs_person	bigint	Fremdschlüssel
fs_verbindungstyp	int	Fremdschlüssel
vorname	varchar	
nachname	varchar	
email	varchar	
username	varchar	

Tabelle 25: Beschreibung Entität User

Tabelle	
Name der Tabelle	ztbl_person_orgaunit
Verbindung in das ER-Diagramm	Relation „gehört zu“ zwischen Person und OrganisationsUnit
Type im ER-Diagramm	Relation
Tabellentyp	Zwischentabelle
Kurzbeschreibung	Ermöglicht eine M zu N Beziehung zwischen Person und OrganistationsUnit.
Beispiel	Person „Peter“ „Lousberg“ gehört zu Orgaunit „AB-Controll“ und zu „Student“. Person „Max“ „Muster“ gehört zu Orgaunit „Student“.
Verbindungen	- N zu 1 mit tbl_person - N zu 1 mit tbl_orgaunit
Indirekte Verbindungen	Nicht Sinnvoll, da Zwischentabelle.
Attribute	

Feld	Datentyp	Schlüssel
id	bigint	Primärschlüssel
fs_person	bigint	Fremdschlüssel
fs_orgaunit	bitint	Fremdschlüssel
ansprechpartner	tinyint / boolean	

Tabelle 26: Beschreibung Relation Person OrganisationsUnit

Tabelle		
Name der Tabelle	ref_anrede	
Verbindung in das ER-Diagramm	(nicht im ER-Model)	
Type im ER-Diagramm	--	
Tabellentyp	Referenztabelle	
Kurzbeschreibung	Entspricht der Anrede	
Beispiel	„Herr“ „Peter“ „Müller“	
Verbindungen	- 1 zu N mit tbl_person	
Indirekte Verbindungen	- 1 zu N mit ztbl_person_orgaunit - 1 zu N mit tbl_user	
Attribute		
Feld	Datentyp	Schlüssel
id	int	Primärschlüssel
name	varchar	

Tabelle 27: Beschreibung Entität Anrede

8.4 GUI-Client Oberflächenvorschlag

Während der Entwicklung des Systems wurde ein Oberflächenvorschlag für den GUI-Client erstellt, dieser wurde jedoch auf Grund des Fokus auf den Central-Server und dessen Erweiterbarkeit aus Gründen vernachlässigt.

Abbildung 63: Skizze Oberflächenvorschlag GUI-Client

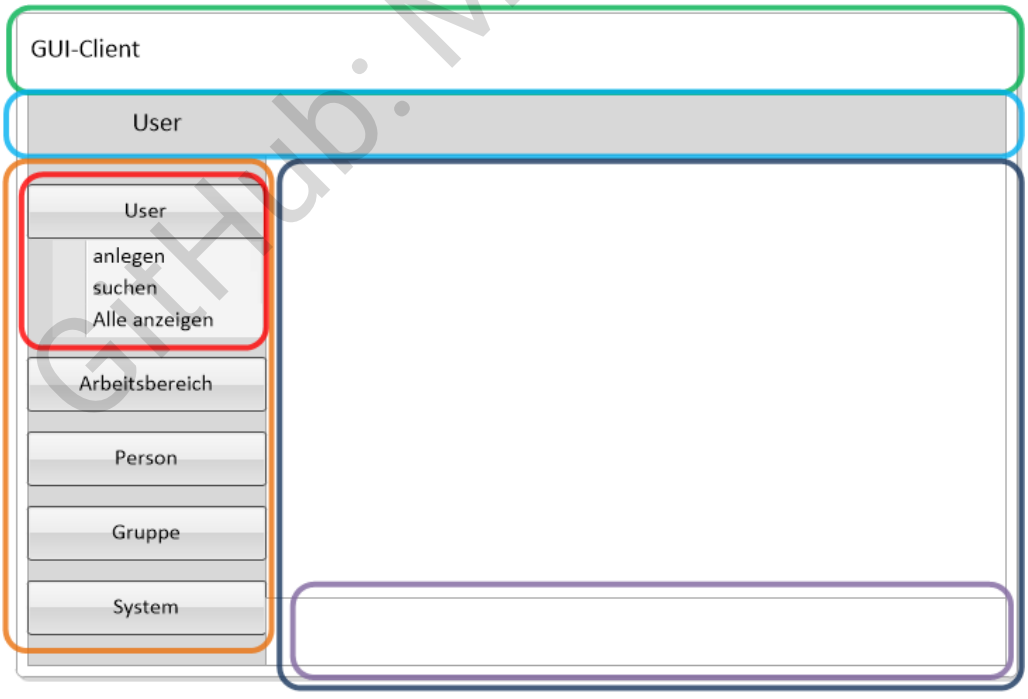

Abbildung 62: Skizze Oberflächenvorschlag GUI-Client mit markierten Bereichen

Fenstermenu des Desktopfensters Ein Standard-Fenstermenü mit betriebssystemtypischen Aktionen (minimieren, verkleinern, schließen)
Anzeige der aktuellen Navigation Durch die Beschriftung wird der aktuell ausgewählte Bereich angezeigt.
Contentbereich, für Formulare und Listen
Dynamischer Navigationsbereich abhängig von dem geladen Inhalt, um Logisch verbundene Aktionen anzubieten
Navigationsbereich mit allen Bereichen der Navigation Durch Klick auf einem Navigationsknoten wird dieser aktiviert und die Subnavigation geöffnet.
Aktiver Navigationsknoten (User), mit Subnavigation in Form einer Dropdownliste

Abbildung 64: Beschreibung zur Skizze des Oberflächenvorschlags

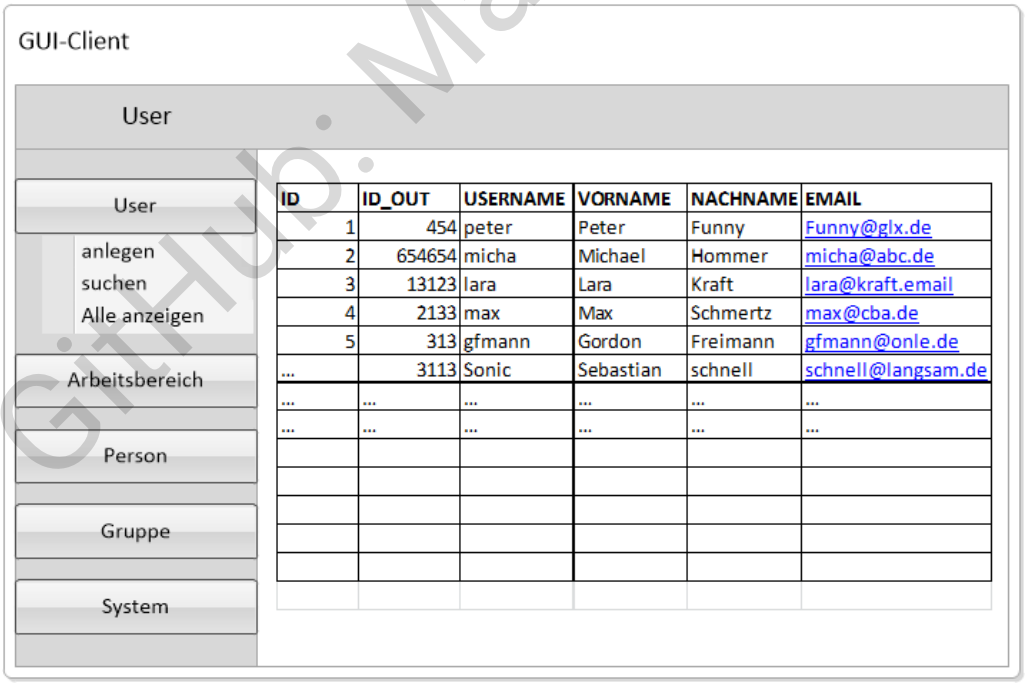

Abbildung 65: Skizze Oberflächenvorschlag GUI-Client mit Inhalt in Tabellenform

9 Werkzeuge

Name	Generelle Aufgabe	Einsatz in der Arbeit
Eclipse 4.4.0 Java	Entwicklungsumgebung	- Implementierung von Java - Schreiben von XSD-Dateien - Generieren von XSD-Diagra. - Kontrolle von Genierieten Elementen in Java - Schreiben von XML
Umlet 12.0	Zeichentool für UML-Diagramme	- Zeichnen von Klassen-Diagra. - Zeichnen von Paket-Diagra. - Zeichnen von Ordner-Strukt. - Zeichnen von Kommunikations-Dia. - Zeichnen von Sequenz-Diagra.
Dia 0.97.2	Zeichentool für Diagramme	- Zeichnen von Er-Diagra. - Zeichnen von DB-Diagra.
Objekt Aid UML Explorer	Zeichentool und Generator für Diagramme als Plugin für Eclipse	- Generieren von Klassen-Diagra. - Zeichnen von Klassen-Diagra.
Microsoft Office Word 2010	Textverarbeitungswerkzeug	- Schreiben der Bachelorarbeit - Einbetten von erzeugten Diagra.
Microsoft Visio 2010	Zeichentool für Diagramme und grafischen Oberflächen	- Zeichen des Oberflächenvorschlags des GUI-Clients
JDK 1.7.0_xy (64bit)	Java Development Kit und Laufzeitumgebung	- Kompilieren von Java - Ausführen von Java - Generieren von Java aus XSD

10 Literaturverzeichnis

[1] S. Kleuker, Grundkurs Software-Engineering, Vieweg + Teubner, 2009.

[2] C. Rupp und S. die, Requirementsengineering und Management, München: Carl Hanser Verlag, 2007.

[3] D. M. Sosnoski, „iBX: Binding XML to Java Code," Sosnoski, [Online]. Available: http://jibx.sourceforge.net/. [Zugriff am 10 11 2014].

[4] „XMLBeans," Apache, [Online]. Available: https://xmlbeans.apache.org/. [Zugriff am 14 12 2014].

[5] „Project JAXB," [Online]. Available: https://jaxb.java.net/. [Zugriff am 14 12 2014].

[6] C. Rupp, S. Queins und d. SOPHISTen, UML2 glasklar, München: Carl Hanser Verlag, 2012.

[7] M. Kohs, C. Schmitz, J. Hittorf und M. Häsel, *RUNDENBASIERTES NETZWERKSPIEL,* Aachen: FH Aachen Campus Jülich, 2011.

[8] E. Freeman, E. Freeman, K. Sierra und B. Bates, Entwurfsmuster von Kopf bis Fuß, Köln: O'Reilly Verlag GmbH & Co. KG, 2006.

[9] C. Ullenboom, „Java ist auch eine Insel 9.11 Variablen mit volatile kennzeichnen," [Online]. Available: http://dev.cs.uni-magdeburg.de/java/Books/javainsel3/javainsel_090010.htm. [Zugriff am 26 11 2014].

[10] H. Helmke, F. Höppner und R. Isernhagen, Einführung in die Software-Entwicklung, München Wien: Carl Hanser Verlag, 2007.

[11] G. R. Bernhard Lahres, „Praxisbuch Objektorientierung," 2006. [Online]. Available: http://openbook.galileocomputing.de/oo/oo_04_strukturvonooprogrammen_02_003.htm. [Zugriff am 03 11 2014].

[12] MySql, „The CHAR and VARCHAR Types," Oracle Corp., [Online]. Available: https://dev.mysql.com/doc/refman/5.6/en/char.html. [Zugriff am 03 11 2014].

[13] MySql, „Using Character Sets and Unicode," Oracle Corp., [Online]. Available: https://dev.mysql.com/doc/connector-j/en/connector-j-reference-charsets.html. [Zugriff am 03 11 2014].

[14] P. D.-i. P. Liggesmeyer, Software-Qualität Testen, Analysieren und Verifizieren von Software, Heidelberg: Spektrum Akademischer Verlag, 2009.

[15] MySql, „Character Sets and Collations," Oracle Corp., [Online]. Available: https://dev.mysql.com/doc/refman/5.6/en/charset-charsets.html. [Zugriff am 03 11 2014].

[16] „Wikipedia Sternschema,“ [Online]. Available: https://de.wikipedia.org/wiki/Sternschema. [Zugriff am 05 11 2014].

[17] H. Michael, Modellierung von Business-Intelligence-Systemen, Heidelberg: dpunkt.verlag, 2014.

[18] F. Geisler, Datenbanken Grundlagen und Design, Austria: mitp-verlag, 2009.

[19] MySql, „Data Type Storage Requirements,“ Oracle Corp., [Online]. Available: https://dev.mysql.com/doc/refman/5.6/en/storage-requirements.html. [Zugriff am 12 11 2014].