Implementierung und Verifikation eines Report-Browsers für Multidimensionale Datenbanken im Klinikumfeld

Diplomarbeit im Studiengang Medizinische Informatik
Hochschule Heilbronn und Universität Heidelberg

vorgelegt von

Hauke Hund

Betreuer: Prof. Dr. Christian Fegeler
2. Betreuer: Prof. Dr. Dirk Heuzeroth

Mai 2011

Hauke Hund
Neuenheimer Landstraße 5
69120 Heidelberg

Zusammenfassung

Die großen Datenmengen der klinischen Routine stellen für die medizinische Forschung ein großes Potenzial dar. So lassen sich zum Beispiel doppelte Erhebungen vermeiden oder Studienteilnehmer schneller finden. Sollen diese Daten genutzt werden, bedarf es geeigneter Werkzeuge und Prozesse.

Im Rahmen des RWH Projektes der Medizinischen Uniklinik Heidelberg und dem GECKO Institut der Hochschule Heilbronn soll in dieser Arbeit ein Abfragewerkzeug für multidimensionale Datenbanken erstellt und verifiziert werden. Den Schwerpunkt der Arbeit bildet die Wahl einer geeigneten Softwarearchitektur.

Im Anschluss an eine Anforderungsanalyse wird das Abfragewerkzeug mit Hilfe von Java Technologien, wie dem Google Web Toolkit und dem Open Java API for OLAP, erstellt. Die Anforderungen werden mit zwei Anwendungsszenarien verifiziert.

Der RWH Report-Browser konnte mit der festgelegten Architektur implementiert werden. Zum Erstellen von MDX Anfragen an das Data Warehouse wurde ein Anfragegenerator implementiert.

Die Verifikation zeigt, dass der Report-Browser als Plattform für den Zugriff auf klinische Routinedaten geeignet ist. Eine gute Testbarkeit der Architektur konnte nachgewiesen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungen

AD	Active Directory, Verzeichnisdienst von Microsoft
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface, Deutsch: Programmierschnittstelle
BI	Business Intelligence, Deutsch: Geschäftsanalytik
BMBF	Bundesministerium für Bildung und Forschung
CPU	Chest Pain Unit: Kardiologische- und Pneumologische Notfallambulanz der Medizinischen Universitätsklinik Heidelberg
CSV	Comma-Separated Values, Dateiendung von Dateien, welche Daten mittels Trennzeichen (meist Semikola) speichern
DAO	Data Access Object
DOM	Document Object Model
DRG	Diagnosis Related Groups, Deutsch
EPL	Eclipse Public License
GMDS	Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e.V.
GWT	Google Web Toolkit
HOLAP	Hybrid OLAP
hsTnT	Hochsensibles Troponin T, Weiterentwicklung des Blutparameters Troponin T [Kur11]
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure, verschlüsselte Variante des HTTP

i2b2	Informatics for Integrating Biology and the Bedside
ICD-10	International Statistical Classification of Diseases and Related Health Problems, Deutsch: Internationale statistische Klassifikation der Krankheiten und verwandter Gesundheitsprobleme
IDE	Integrated Development Environment
Java EE	Java Enterprise Edition
JDBC	Java Database Connectivity, Java Programmierschnittstelle für relationale Datenbanken
Kardio-MRT	Kardio-Magnetresonanztomographie
KHEntgG	Krankenhausentgeltgesetz
KIS	Krankenhausinformationssystem[1]
LDAP	Lightweight Directory Access Protocol, Übertragungsprotokoll zum Zugriff auf Verzeichnisdienste, wie z.B. einer Benutzerdatenbank
LV EF	Links ventrikuläre Ejektionsfraktion, auch als Auswurffraktion bezeichnet. Verhältnis von links ventrikulären Schlagvolumen zum links ventrikulären Gesamtblutvolumen
MDX	MultiDimensional eXpressions
MOLAP	Multidimensional OLAP
MVC	Model View Controller
MVP	Model View Presenter
MVVM	Model View View-Model
NIH	National Institutes of Health, Behörde des United States Department of Health and Human Services - Gesundheitsministerium der Vereinigten Staaten
OLAP	Online Analytical Processing

[1] Definiert in [Win98]

olap4j	Open Java API for OLAP, Java Programmierschnittstelle für OLAP Datenbanken
OWASP	Open Web Application Security Project
ROLAP	Relational OLAP
RPC	Remote Procedure Call
RWH	Research Warehouse
SGB V	Sozialgesetzbuch V
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
TCP	Transmission Control Protocol
TMF	Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V.
UI	User Interface, Deutsch: Benutzeroberfläche
URL	Uniform Resource Locator, Ort im Internet
XLS	Dateiendung von Dateien des Tabellenkalkulationsprogramms Microsoft Excel
XMLA	XML for Analysis
XSS	Cross-Site Scripting

1 Einleitung

Die Einleitung bildet das erste Kapitel dieser Diplomarbeit. In diesem Textabschnitt wird gezeigt, in welchen Kontext sich diese Arbeit eingliedert; hier wird das Forschungsprojekt RWH kurz beschrieben.

Eine Darstellung der Ziele dieser Arbeit und ihr Aufbau schließen die Einleitung ab.

1.1 Hinführung

Große Datenmengen werden zur Dokumentation von Behandlungen und Untersuchungen im Rahmen der medizinischen Versorgung erhoben. Diese Daten stellen für die Forschung in der Medizin ein großes Potenzial dar. Die Nutzung dieser Routinedaten ist jedoch auch mit großen Herausforderungen verbunden. Durch die Nutzung von Routinedaten lassen sich doppelte Erhebungen vermeiden und die Zeit zum Durchführen einer klinische Studie verkürzen oder auch Studienteilnehmer schneller finden. Andererseits spielen Anforderungen an die Datenqualität und den Datenschutz eine wichtige Rolle [Dug08; PG09; MW02].

Soll also evidenzbasierte Medizin, mit ihrem Anspruch ständig nach der besten Versorgung zu suchen [Sac96], und die Versorgungsforschung, zur Beurteilung der medizinischen Versorgungsrealitäten [Pfa03], mit klinischen Routinedaten unterstützt werden, bedarf es geeigneter Werkzeuge und Prozesse zur Datenverarbeitung, die an die Bedürfnisse der Nutzer angepasst sind. Diese Werkzeuge und Prozesse müssen dazu verhelfen, ständig wechselnde Fragestellungen zu beantworten, sowie gleich bleibende Kennzahlen für wiederkehrende Abfragen bereitzustellen.

Zwar werden in der klinischen Routine große Mengen an Daten erhoben, dies bedeutet jedoch nicht, dass diese rein technisch betrachtet bereits in einer Form vorliegen, die mit Hilfe von geeigneten Abfragewerkzeugen, Forschern zur Verfügung gestellt werden können. Die Integration von Routinedaten und Forschungsdaten, aus den jeweiligen heterogenen Systemlandschaften, bildet eine Herausforderung für die Medizinische Informatik.

1.2 Zusammenhang

Diese Arbeit wird im Rahmen des Research Warehouse (RWH) Projektes erstellt. Das RWH Projekt ist ein Gemeinschaftsprojekt der Abteilung Innere Medizin III der Medizinischen Universitätsklinik Heidelberg[2] und dem GECKO Institut für Medizin, Informatik und Ökonomie an der Hochschule Heilbronn.[3]

Ziel des RWH Projektes ist die Zusammenfassung von klinischen Routinedaten und Forschungsdaten aus unterschiedlichsten Spezial-Datenbankanwendungen der Abteilung Innere Medizin III, um übergreifende Abfragen zu ermöglichen. Anfragen an das Data Warehouse sollen die Versorgungsforschung und somit evidenzbasierte Medizin unterstützen. Über eine Systemintegrationskomponente soll zudem das automatische Auffinden von potenziellen Teilnehmern klinischer Studien möglich gemacht werden. Des Weiteren soll den Anforderungen des Medizin-Controllings Rechnung getragen werden.

Unter den Datenbankanwendungen befinden sich Systeme, die rein zu Forschungszwecken eingesetzt werden sowie Datenbanken, die zur behandlungsbegleitenden Dokumentation genutzt werden. Im Rahmen dieses Projektes werden neue Methoden zur Datenaufbereitung, Analyse und Präsentation evaluiert. Das entwickelte Datenschutzkonzept setzt auf das in der Literatur häufig erwähnte Prinzip des Datentreuhänders zur Pseudonymisierung klinischen Daten [MW02; Pom; Ihl05].

In der Medizinischen Klinik werden für die dortigen Anforderungen individuell entwickelte Datenbankanwendungen eingesetzt. Aus diesen Anwendungen werden mit Hilfe eines ETL-Prozesses[4] zunächst aus den Quellsystemen extrahiert und validiert. Daraufhin erfolgt eine Transformation in das gewünschte

[2] Innere Medizin III: Kardiologie, Angiologie und Pneumologie - Medizinische Universitätsklinik Heidelberg. Siehe auch: `http://www.klinikum.uni-heidelberg.de/med` (besucht am 01.05.2011).

[3] GECKO Institut für Medizin, Informatik und Ökonomie, Hochschule Heilbronn. Siehe auch: `http://www.gecko.hs-heilbronn.de` (besucht am 01.05.2011).

[4] ETL: Extract, Transform, Load. Siehe auch: [BSS00].

Datenformat, bevor die multidimensionale Datenstruktur - das Data Warehouse - angelegt wird.[5] Abbildung 1 zeigt den ETL-Prozess im Rahmen der RWH Basisarchitektur.

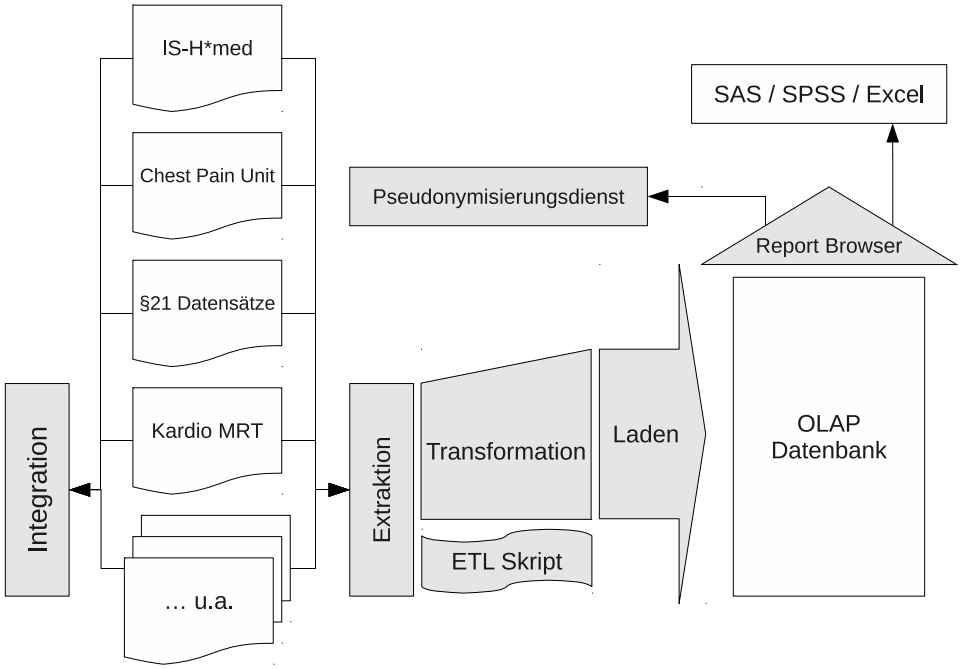


Abbildung 1: RWH Basisarchitektur

In der ersten Version werden Daten aus den Chest Pain Unit (CPU)- und Kardio-Magnetresonanztomographie (Kardio-MRT)-Datenbankanwendungen sowie §21-Datensätze[6] genutzt.

Das hier zu entwickelnde Werkzeug bildet für die Benutzer ein Bindeglied zwischen den Daten des RWH Projektes und Statistiksoftware zur Weiterverarbeitung durch die Biometrie.

[5] Vergleiche: Abschnitt 2.4 (Seite 15).
[6] Datensätze zur Qualitätssicherung, siehe auch §21 Krankenhausentgeltgesetz (KHEntgG).

1.3 Zielsetzung

Ziel dieser Arbeit ist die Durchführung einer Machbarkeitsstudie. In dieser Studie soll die Entwicklung und Verifizierung eines Abfragewerkzeugs für Online Analytical Processing (OLAP) Datenbanken im Klinikumfeld getestet werden.

Ziel des zu erstellenden Report-Browsers ist es, klinischen Benutzer eine Oberfläche zur Verfügung zu stellen, mit der Medizin-Controlling- und medizinische Fragestellungen mit Hilfe der RWH OLAP Datenbank beantwortet werden können.

Im Rahmen der Entwicklung soll eine inhaltliche sowie eine technologische Anforderungsanalyse durchgeführt werden. Die inhaltliche Anforderungsanalyse umfasst Beschreibung und Priorisierung von gewünschten Funktionen der Anwendung. Die technologische Anforderungsanalyse beinhaltet die Auswahl von Software-Technologien zur eigentlichen Entwicklung anhand der inhaltlichen Anforderungen.
Einen zentralen Punkt dieser Arbeit stellt die Erstellung einer geeigneten Software-Architektur dar. Diese Architektur soll es ermöglichen, die inhaltlichen und technologischen Anforderungen an den Report-Browser zu erfüllen und zudem Möglichkeiten zu bieten, zukünftige Anforderungen im Rahmen der Architektur umzusetzen.

Mit kaufmännischen Fragestellungen aus dem Bereich des DRG-Managements und mit medizinischen Fragestellungen aus dem Bereich der Kardiologie wird der Report-Browser verifiziert.

1.4 Aufbau der Arbeit

Diese Arbeit gliedert sich in sechs Teile. Auf die Einleitung folgen:

- **Kapitel 2 - Grundlagen**
  Hier wird ein Überblick über das Forschungsumfeld, in das sich diese Arbeit eingliedert, gegeben. Die technologischen Grundlagen werden beschrieben; zu diesen gehören multidimensionale Datenbanken wie auch die Programmbibliotheken olap4j und GWT.

- **Kapitel 3 - Entwurf**
  Im dritten Kapitel ist die Entwicklungsmethodik des Report-Browsers dargestellt und welche funktionalen und nicht-funktionalen Anforderungen an diese Software gestellt werden. Die entworfene Softwarearchitektur sowie Sicherheitsrisiken und Szenarien zur inhaltlichen Verifikation werden hier dargelegt.

- **Kapitel 4 - Implementierung**
  In Abschnitt 4 werden wichtige Details der Implementierung des Report-Browser aufgezeigt. Einen wichtigen Teil diese Kapitels macht die Beschreibung der Benutzerführung aus.

- **Kapitel 5 - Verifizierung**
  Hier werden die Ergebnisse der Verifikation des Entwurfs und der Zielsetzung diese Arbeit beschrieben. Die dargestellten Ergebnisse machen Aussagen über die Testbarkeit der Softwarearchitektur und die Qualität des Werkzeugs - im Hinblick auf medizinische Forschung - möglich.

- **Kapitel 6 - Diskussion**
  Im letzten Kapitel dieser Arbeit wird sowohl ein technologische- als auch

eine inhaltliche Bewertung der Arbeit vorgenommen. Die Diskussion stellt Maßnahmen dar, die vor einer produktiven Nutzung des Report-Browsers durchgeführt werden müssen, und gibt einen Hinweis auf das Potenzial der Anwendung.

2 Grundlagen

Dieses Kapitel liefert einen Überblick über das Forschungsumfeld, in das sich diese Arbeit eingliedert. Es wird dabei die Frage beantwortet, wie in der Medizin Wissen generiert wird und welche Rolle die Versorgungsforschung bei der Verifizierung spielt.

Technologische Grundlagen, also das Prinzip der Multidimensionalen Datenbanken, ihre Abfragesprache und der Zugriff aus Java sowie das Framework Google Web Toolkit, sind ebenso Teil dieses Abschnittes.

2.1 Primärforschung in der Medizin

Erkenntnisgewinn in der Medizin folgt den Regeln der Wissenschaft. Zu diesem Schluss kommt Johannes Köbberling in seinem Aufsatz „Der Begriff der Wissenschaft in der Medizin“ [Köb98] Er begründet seinen Wissenschaftsbegriff auf der Fehlbarkeitslehre von Karl Popper.[7] Die „Bereitschaft zur ständigen kritischen Überprüfung und gegebenenfalls Verwerfung der Hypothesen“ [ebd.] sei das zentrale Element wissenschaftlicher Medizin.

Ein wichtiges Mittel für die Falsifikation in der Medizin stellt die Primärforschung mit ihren unterschiedlichen Studientypen bereit. Hier werden Hypothesen getestet, angenommen oder verworfen. Die medizinische Primärforschung unterscheidet Grundlangenforschung (auch als experimentelle Forschung bezeichnet), klinische Forschung und epidemiologische Forschung. Die Grundlagenforschung umfasst „Tierversuche, Zellversuche, biochemische, genetische und physiologische Untersuchungen [... und] Studien zu Arzneimittel- und Materialeigenschaften“ [Röh09, S. 263f] sowie Untersuchungen zu Diagnostischen Tests (also bildgebenden Verfahren, Blutuntersuchungen und genetische Analysen). Klinische Forschung unterscheidet experimentelle und beobachtende Studien. Inhalte dieser Studien sind meist die Beurteilung von neuen Behandlungsmethoden. Während in experimentellen Studien die Behandlung durch das Studienprotokoll vorgegeben wird und somit aktiv in den Behandlungsablauf eingegriffen wird, entscheidet der Arzt in beobachtenden Studien jedoch nach medizinischen Kriterien oder Patientenwunsch. [ebd.] Epidemiologische Studien betrachten zumeist die „Verteilung und zeitliche Veränderung der Häufigkeiten von Krankheiten sowie deren Ursachen“ [ebd., S. 266].

Zur Sicherstellung von interner Validität (d.h. Eindeutigkeit der Ergebnisse) wird in Studien aller drei Teilbereichen versucht, Beobachtungsgleichheit, Behandlungsgleichheit, Strukturgleichheit und standardisierte Versuchsbedingungen zu erreichen. Zur Reduktion der Komplexität wird die zu untersuchende

[7] Karl Raimund Popper - Philosoph: * 1902 Wien - † 1994 London.

Patientengruppe mit Hilfe von Einschluss- und Ausschlusskriterien klar definiert [Reb09, S. 131].

Die externe Validität solcher Studien (d.h. Generalisierbarkeit der Ergebnisse) ist jedoch meist nicht überprüfbar und nur heuristisch begründbar [Kie10]. Es lassen sich diese „Laborbedingungen [...] nicht immer auf den klinischen Alltag [...] übertragen“ [Röh09].

Einen Ausweg aus diesem Dilemma bildet die Versorgungsforschung: Diese sei zwar „keine wissenschaftliche Disziplin ‚sui generis‘“ so Schmacke [Sch07], aber „ein interdisziplinärer Ansatz, der inhaltlich und methodisch auf die Effizienzsteigerung der medizinischen Versorgung unter Alltagsbedingungen abzielt.“

2.2 Versorgungsforschung

Die Versorgungsforschung beschäftigt sich mit den Versorgungsrealitäten im Gesundheitswesen. Während die Generalisierbarkeit von Ergebnissen der experimentellen Forschung auf Grund von geringer Variabilität in den gewählten Beobachtungseinheiten häufig nicht angenommen werden kann, wird in der Versorgungsforschung bewusst der Versorgungsalltag zur Beurteilung der Wirksamkeit einer Methode gewählt [Röh09]. Die Versorgungsforschung ist ein Teilgebiet der Gesundheitssystemforschung und untersucht das Gesundheitssystem auf der Mikroebene, also die konkrete Gesundheitsversorgung in Krankenhäusern, Praxen und sonstigen Einrichtungen.

„Aus systemtheoretischer Sicht ist Versorgungsforschung ein Forschungsgebiet, welches untersucht, wie das Versorgungssystem und seine kulturellen, sozialen, personalen, technischen, ökonomischen und organisatorischen Eigenheiten und Rahmenbedingungen Input, Output und Outcome dieses Systems beeinflussen.“ [Pfa03]

In der Versorgungsforschung werden die Teilgebiete: Grundlagenforschung (Erklärung des Ist-Zustandes) und Anwendungsforschung (Entwicklung von Versorgungskonzepten) unterschieden.

Ziel der Versorgungsforschung ist es neben der Beschreibung des Gesundheitssystems und der Entwicklung von neuen Konzepten für mehr Transparenz im Gesundheitswesen zu sorgen. Um dies zu ermöglichen, ist es nötig,F große Mengen an Daten über das System auszuwerten und zu beurteilen. „Ohne die (Sekundär-)Nutzung bereits vorhandener Informationen wäre dieser Bedarf mit Primärerhebungen kaum zu vertretbaren Kosten [...] zu stillen.“ [SI05, S. 11]

Die Bereitstellung von primär zur Dokumentation und Behandlung erhobenen Daten zur Sekundärnutzung in der Versorgungsforschung sowie prospektiver oder retrospektiver Experimentalforschung ist Gegenstand des RWH Projektes.

2.3 Themenbezogene Arbeiten

Diese Arbeit gliedert sich in ein derzeit sehr aktives Forschungsumfeld. Der dieses Jahr zum zehnten Mal stattfindende Kongress für Versorgungsforschung[8] sowie die Projektgruppe: „Nutzung von Elektronischen Krankenakten für die klinische Forschung“ [PDS10] der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e.V. (GMDS) sind Zeugen dieser Aktivitäten. Ebenso Projekte rund um das derzeit wichtigste Softwareprojekte in diesem Bereich, das US Amerikanische Projekt Informatics for Integrating Biology and the Bedside (i2b2)[9] [Mur07; Mur10]. Diverse Projekte in den USA und auch in Europa (Frankreich, Italien, Deutschland - Göttingen [Mur09, Folie 45]

[8] 10. Deutscher Kongress für Versorgungsforschung - Deutsches Netzwerk Versorgungsforschung e.V.. Siehe auch: `http://dkvf2011.de` (besucht am 01.05.2011).

[9] Gefördert durch den NIH Common Fund, des US Gesundheitsministeriums. Siehe auch: `http://commonfund.nih.gov/bioinformatics` (besucht am 01.05.2011).

und Erlangen [Gan10]) setzten auf die im i2b2 Projekt entwickelte Opensource Software.

Im Journal „Forum der Medizin_Dokumentation und Medizin_Informatik (mdi)“ beschreibt Prokosch die aktuellen „Single-Source-Aktivitäten in Deutschland“ [Pro10]. Mit „Single-Source“ wird eine Methode beschrieben, klinische Routinedaten aus der Dokumentation für die medizinische Forschung verfügbar zu machen und somit doppelt Erhebungen zu vermeiden. Drei Projekte der Universitätskrankenhäuser Münster, Essen und Erlangen werden ebenso in diesem Heft beschrieben [ebd., ff].

Ganslandt et al. berichten aus Erlangen von „Beispielprojekte[n] zur Nutzung von Daten aus der elektronischen Krankenakte für die klinische und translationale Tumorforschung“, [Gan10] welche erfolgreich mit Hilfe der i2b2 Software realisiert wurden. Sie machen jedoch deutlich, dass meist nicht die technische Realisierbarkeit problematisch ist, sondern dass, „bevor Daten aus der elektronischen Krankenakte für Forschungszwecke weiterverwendet werden, [...] es unbedingt notwendig [ist], den Kontext, in dem Daten für die Krankenversorgung erhoben werden, genau zu analysieren und zu prüfen, ob dies tatsächlich auch den Merkmalsdefinitionen entspricht, die für das jeweilige Forschungsprojekt definiert wurden“. [ebd.]

Einen guten Überblick über die Sekundärnutzung von Routinedaten im Gesundheitswesen liefern Swart und Ihle (Hrsg.) in ihrer Papersammlung: „Routinedaten im Gesundheitswesen“ [SI05]. Die Arbeiten reichen von gesetzlichen Rahmenbedingungen über die Darstellung von Methoden bis hin zu diversen Fallbeispielen.

Um den Single-Source Aktivitäten im deutschsprachigen Raum eine Plattform zum Austausch zu geben, wurde vor zwei Jahren (2009) von der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e.V. (GMDS) die bereits erwähnte Projektgruppe „Nutzung von Elektronischen Krankenakten für die klinische Forschung“ ins Leben gerufen. Wichtige Inhalte der Arbeitsgruppe sind laut Eigendarstellung: Wiederverwendung von Da-

ten aus dem Krankenhausinformationssystem (KIS) bzw. einer elektronischen Krankenakte, IT-Unterstützung beim Management von Biomaterialdatenbanken und Patientenrekrutierung für klinische Studien mit Hilfe von Daten aus der elektronischen Krankenakte.

Laut einer Pressemitteilung der Universität Münster scheitere jede dritte Studie an einer zu geringen Zahl an Probanden [Nus10]. Dieses ließe sich möglicherweise ändern, indem Routinedaten automatisch nach passenden Patienten ausgewertet werden, so Ohmann et al. [OK07]. Sie beschreiben den Prozess der Patientenrekrutierung und mögliche technische Probleme beim automatischen auswerten elektronischer Krankenakten.

Dugas et al. [Dug08] eigen am Universitätsklinikum Münster eine automatische Auswertung von elektronischen Krankenakten und eine Benachrichtigung per E-Mail über potenzielle Studienteilnehmer. Das in der Arbeit vorgestellte System wurde für Studien der Akuten myeloischen Leukämie implementiert.

Ein Gemeinschaftsprojekt in diesem Bereich der Universitäten Münster, Erlangen-Nürnberg, Düsseldorf, Heidelberg, Gießen und dem Verein, Technologie und Methodenplattform für die vernetzte medizinische Forschung (TMF e.V.) wird derzeit vom Bundesministerium für Bildung und Forschung (BMBF) gefördert. In diesem Verbundprojekt: „KIS-basierte Unterstützung der Patientenrekrutierung in klinischen Studien“ [BMBF10], sollen IT-Unterstützungen zur Rekrutierung von Patienten für klinische Studien in unterschiedlichen KIS Architekturen prototypisch implementiert und evaluiert werden.

Nur wenige Arbeiten wurden bisher im Bereich „Anwendungen von OLAP Methoden in der klinischen Forschung“ publiziert. Eine Arbeit stammt aus Slowenien. Hirstovski et al. [HRM00] berichten 2000 von einem OLAP Data Warehouse für ambulante Patientendaten auf nationaler Ebene. Sie beschreiben die Gesundheitswissenschaften als geeignete Domäne für die Nutzung von OLAP und wählten diese Methodik vor allem wegen ihrer Möglichkeiten zur interaktiven Suche und Analyse. In einer weiteren Arbeit wurde von Wu 2006 ein Data Warehouse ebenfalls aus dem ambulanten Bereich beschrieben [Wu06].

Explizite Arbeiten zu Abfragewerkzeugen für klinische Forschungsdatenbanken konnten bisher nicht gefunden werden. Erwähnenswert ist jedoch die i2b2 Workbench [Mur07; Mur10]. Dieses Werkzeug dient dem i2b2 Projekt als Abfragewerkzeug und wird als „benutzerfreundliche[...] Query-Oberfläche zum Recherchieren von Patientenkohorten für neu zu konzipierende Forschungsprojekte“ bezeichnet [Gan10].

2.4 Multidimensionale Datenbanken

Die Analyse multidimensionaler Daten reicht zurück in die 1960er Jahre und wurde von E.F. Codd das erste Mal 1960 als Online Analytical Processing (OLAP) definiert [JF10, S. 7]. OLAP beschreibt eine Methode zur Verarbeitung von Daten mit einem multiplen Dimensionsmodell. Multidimensionale Analysen finden vor allem in Bereichen der Business Intelligence (BI) Verwendung, jedoch wurden auch Einsätze in anderen Bereichen z.B. Agrarwirtschaft [CSL04; Abd09] und Medizin [HRM00; Wu06] beschrieben.

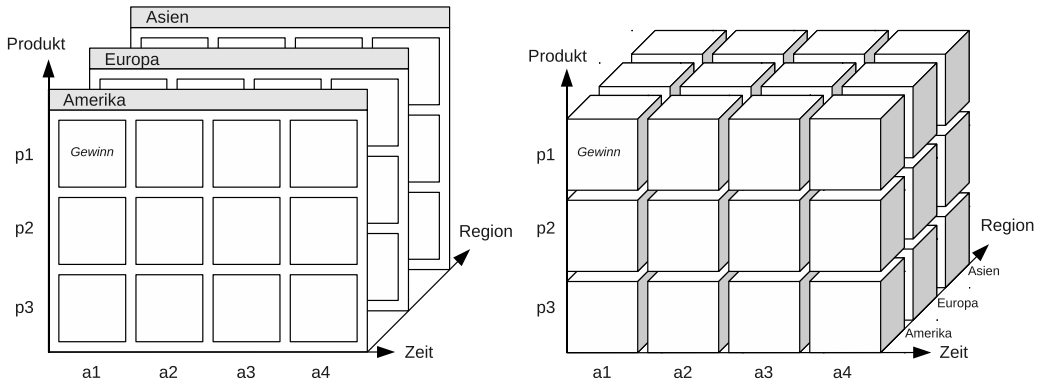


Abbildung 2: Von Tabellen zum Würfel

Sollen zum Beispiel (Siehe Abbildung 2) Gewinne einzelner Produkte (p1 - p3) für unterschiedliche Jahre (a1 - a4) anhand der Regionen Amerika, Europa und Asien aufgeschlüsselt werden, sind dazu in einem relationalen Modell drei Tabellen nötig. Multidimensonale Datenbanken bieten die Möglichkeit, diese

Daten in einem Würfel (mit n Dimensionen) abzuspeichern. Produkt, Zeit und Region werden als Dimensionen bezeichnet, Messwerte, wie z.B. Gewinn oder die Anzahl verkaufter Produkte, werden als Maßzahlen (Eng. Measures) bezeichnet.

2.4.1 Aufbau

Daten werden in multidimensionalen Datenbanken als so genannte Measures (Eng.) abgespeichert und sind in der Regel numerische Werte. Die Messwerte bilden Punkte im Dimensionsmodell. Die einzelnen Dimensionen werden genutzt, um die Messwerte zu beschreiben oder zu gruppieren (i.d.R. Texte). Dimensionen (Eng. Dimensions) enthalten Hierarchien (Eng. Hierarchies), welche wiederum unterschiedliche Detailstufen (Eng. Level) beinhalten. So ist die Zeit-Dimension typischerweise mit einer Hierarchie `Datum`, welche die Detailstufen: `Jahr`, `Semester`, `Quartal`, `Monat` und `Tag` enthält, versehen. Eine Hierarchie, die zum Beispiel das Geschäftsjahr abbildet wäre hier ebenfalls denkbar.

OLAP unterscheidet grundsätzlich drei Arten, Daten zu speichern [Wiki11b]:

- **Multidimensional OLAP (MOLAP)**
  MOLAP bildet das klassische OLAP. Daten werden hier vom Datenbanksystem multidimensional gespeichert. Diese Art der Speicherung erfordert die Aufbereitung von relationalen Quelldaten. MOLAP gilt als speicherintensive und schnellste OLAP Variante.
- **Relational OLAP (ROLAP)**
  In der relationalen OLAP Variante werden Daten in relationalen Strukturen gespeichert. Sollen auf diesen OLAP Abfragen bearbeitet werden, müssen Datenwürfel zur Laufzeit berechnet werden. ROLAP priorisiert den Speicherverbrauch gegenüber der Ausführungsgeschwindigkeit.
- **Hybrid OLAP (HOLAP)**
  HOLAP stellt einen Kompromiss zwischen MOLAP und ROLAP dar.

Teile der Daten werden relational, andere multidimensional abgespeichert.

2.4.2 Grundoperationen

2.4.2.1 Slicing

Unter Slicing (Schneiden) versteht man das Eingrenzen einer Dimension auf eine Teilmenge aller möglichen Einträge. In Abbildung 3 wird zum Beispiel aus der Zeit-Dimension nur das Jahr 2007 gewählt.

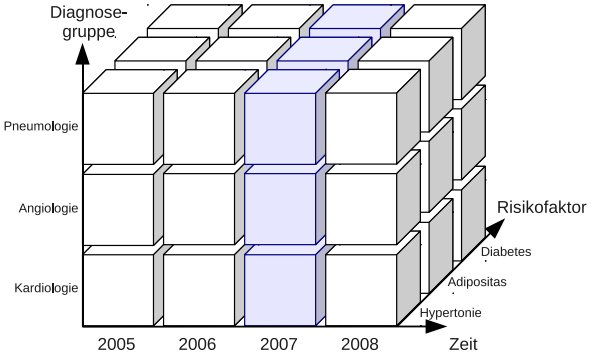


Abbildung 3: Slicing

2.4.2.2 Dicing

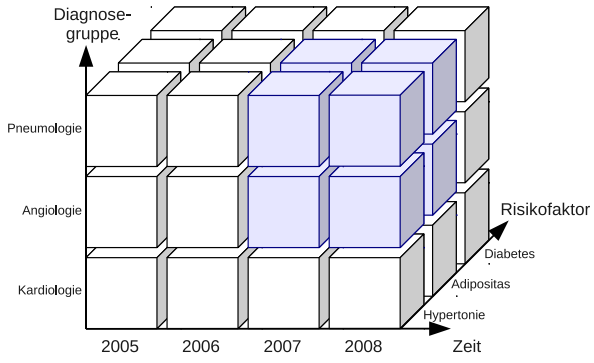


Abbildung 4: Dicing

Dicing bezeichnet die Bildung einer Teilergebnismenge mittels Eingrenzung von mehr als einer Dimension. Im Beispiel (Abbildung 4) werde aus allen drei Dimensionen Teilmengen gewählt. Dicing kann somit auch als hintereinander Ausführen mehrerer Slicing-Vorgänge verstanden werden.

2.4.2.3 Pivoting

Mit Pivoting wird die Drehung eines OLAP-Würfels derart beschrieben, dass eine zuvor nicht sichtbare Dimension in der Ergebnismenge sichtbar wird. In Abbildung 5 werden durch drehen des Würfels die Dimensionen `y` und `z`, anstelle von `x` und `y` in der zweidimensionalen Ergebnispräsentation, sichtbar.

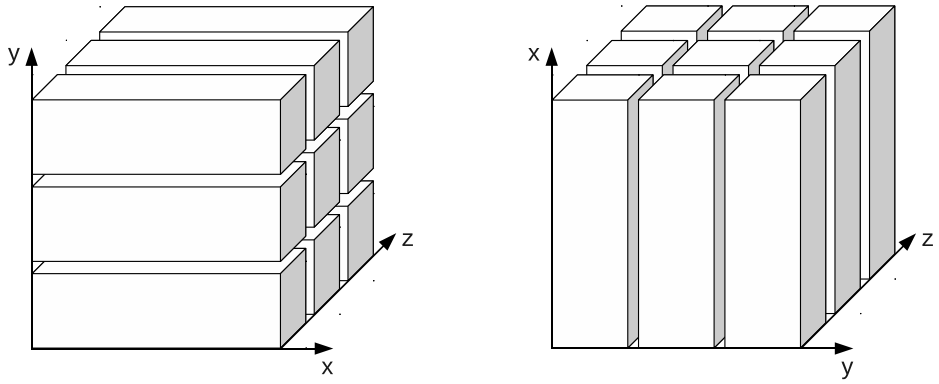


Abbildung 5: Pivoting

2.4.2.4 Drill-Down / Roll-Up

Drill-Down und Roll-Up bezeichnen das Verändern der Detailtiefe der dargestellten Daten. Im Beispiel (Abbildung 6) wird die Auflösung der Dimension Zeit verändert. Zeit-Dimensionen beinhalten typischerweise die Detailstufen Jahr, Semester, Quartal, Monat und Tag. Drill-Down bezeichnet dabei einen Vorgang der mehr Details zeigt, also eine höhere Auflösung der Daten. Beim Roll-Up werden die Daten auf einer niedrigeren Detailebene, meist durch Summierung, aggregiert.

2.4.3 Abfragesprache

Anfragen an OLAP Datenbanken werden mit MultiDimensional eXpressions (MDX) geschrieben. MDX wurde 1997 von Microsoft zum ersten Mal definiert

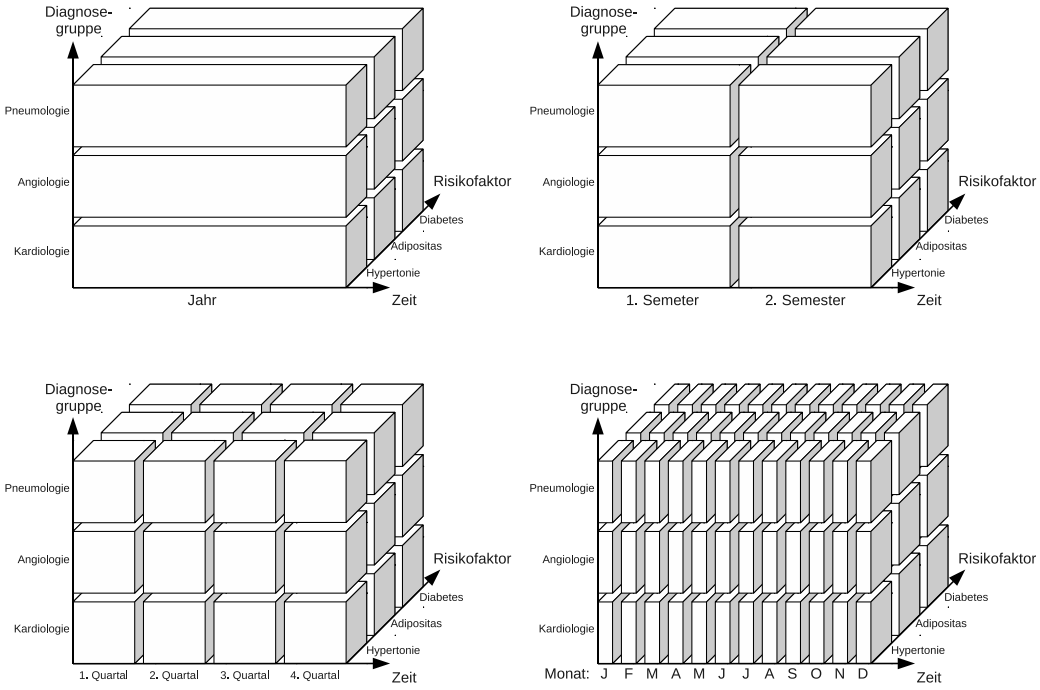


Abbildung 6: Drill-Down / Roll-Up

und wurde zum Quasi-Standard [JF10, S. 219]. MDX ist an die Structured Query Language (SQL) angelehnt und folgt ebenso dem Aufbau:

`SELECT ... FROM ... WHERE.`

2.4.3.1 Select

Der `SELECT` Teil einer MDX Anfrage enthält die Bezeichnungen der Achsen in der Ergebnismenge. Die folgende Anfrage (nur `SELECT` Teil) nutzt zwei Achsen und das Ergebnis kann in einer Tabelle dargestellt werden. Bezeichner von Dimensionen folgen dem Muster: „Dimension.Hierarchie.Level“, eckige Klammern werden genutzt, um Namen eindeutig zu machen.

```
SELECT [Dimension 1].[Hierarchie 1].[Level 1] ON ROWS,
    [Dimension 2].[Hierarchie 1].[Level 1] ON COLUMNS
```

Für die ersten fünf Ergebnisachsen existieren Namen (`COLUMNS`, `ROWS`, `PAGES`, `SECTIONS` und `CHAPTERS`). Unterstützt werden bis zu 128 Achsen, welche im `SELECT` Teil der Abfrage mit `ON 1`, `ON 2` usw. genutzt werden können [MSDN11].

Sollen andere Dimensionen des OLAP-Würfels dargestellt werden (vergl. 2.4.2.3, Seite 18), muss die Zuordnung von Dimensionen und Achsen geändert werden.

Maßzahlen können entweder nur auf einer Achse oder mit Hilfe des Where Teils als Inhalt der Ergebniszellen ausgewählt werden.

Drill-Down und Roll-Up lassen sich realisieren, in dem andere Level auf den Achsen gewählt werden. (vergl. 2.4.2.4, Seite 18) Zum Beispiel:

```
SELECT [Zeit].[Datum].[Monat] ON ROWS
```

und

```
SELECT [Zeit].[Datum].[Quartal] ON ROWS
```

2.4.3.2 From

Mit Hilfe einer Angabe im `FROM` Teil lässt sich ein OLAP-Würfel auswählen.

2.4.3.3 Where

Der `WHERE` Teil der Anfrage dient zum Eingrenzen der Ergebnismenge (vergl. 2.4.2.1 und 2.4.2.2, Seite 17) oder zur Auswahl einer Maßzahl für die Werte der Ergebniszellen.

Im folgenden Beispiel (nach [JF10, S. 232f]) wird die Zeit-Dimension mit der Auflösung „Monate“ auf der ersten Achse, die Produkt-Dimension mit der Kategorie „Fahrräder“ sowie die Maßzahl „Versandkosten“ für die Region „Amerika“ gewählt:

```
SELECT [Zeit].[Datum].[Monat] ON COLUMNS,
    [Produkt].[Kategorie].[Fahrräder] ON ROWS
FROM [Datenwürfel 1]
WHERE ([Maßzahlen].[Versandkosten],
    [Geographie].[Kontinent].[Amerika])
```

2.5 Open Java API for OLAP (olap4j)

Die Open Java API for OLAP (olap4j) Bibliothek dient der einfachen Integration von OLAP Datenbank in Java Anwendungen. olap4j wird als Opensource Projekt (Eclipse Public License (EPL) Lizenz) auf sourceforge[10] entwickelt und liegt zum Zeitpunkt der Implementierung des Report-Browsers in Version 0.9.8 (Beta) vor.

Ziel des olap4j API ist es, Software unabhängig von der Implementierung der OLAP Datenbank bzw. dem OLAP Datenbank Produkt zu machen. Dies wird erreicht, indem, wie auch beim Java Database Connectivity (JDBC) API, produktspezifische Details innerhalb des jeweiligen Treibers implementiert werden [HK11]. Ein XML for Analysis (XMLA) Treiber zur Kommunikation mit XMLA fähigen OLAP Datenbanken wird durch die Autoren des olap4j APIs bereitgestellt. Das XMLA Protokoll basiert auf dem Datentransfer Standard Simple Object Access Protocol (SOAP) und wird von den OLAP Datenbankprodukten von Microsoft, Oracle, SAS und SAP unterstützt [XMLA].

Das olap4j API ist als Erweiterung des JDBC APIs implementiert; auf diese Art lassen sich bekannte Connection-Pooling Bibliotheken wie Jakarta Commons DBCP[11] und c3p0[12] einsetzen.

[10] olap4j: `http://sourceforge.net/projects/olap4j` (besucht am 01.05.2011).
[11] DBCP: `http://jakarta.apache.org/commons/dbcp` (besucht am 01.05.2011).
[12] c3p0: `http://sourceforge.net/projects/c3p0` (besucht am 01.05.2011).

Zur Laufzeit bietet die Bibliothek Zugriff auf das OLAP Dimensionsmodell und die verfügbaren Datenwürfel eines Servers. Listing 1 zeigt, wie mit der olap4j Bibliothek eine Verbindung zu einem Server aufgebaut und alle verfügbaren Datenwürfel aufgelistet werden können. In diesem Beispiel wird der olap4j XMLA Treiber verwendet, um eine Verbindung zu einem Microsoft SQL Server aufzubauen.

```
1  import java.sql.*;
2  import org.olap4j.*;
3
4  Class.forName("org.olap4j.driver.xmla.XmlaOlap4jDriver");
5
6  Connection con = DriverManager.getConnection("jdbc:xmla:Server=" +
7    "http://localhost/olap/msmdpump.dll;Catalog=DB_Name");
8  OlapConnection olapConnection =
9    ((OlapWrapper) con).unwrap(OlapConnection.class);
10
11 for (Catalog catalog : olapConnection.getCatalogs())
12   System.out.println(catalog.getName());
```

Listing 1: Server Verbindung und Auflistung aller Datenwürfel. Nach [HK11]

Datenbank Anfragen in der MDX Anfrage Sprache können mit Hilfe der Java Klassen im package `org.olap4j.mdx` programmatisch erstellt werden. Ein Beispiel liefert Listing 2.

Das Beispiel[13] konstruiert folgende Anfrage:

```
SELECT
[Masszahl] ON ROWS
FROM [Datenwuerfel]
```

[13] Umlaute wurden auf Grund von Einschränkungen im Textsatzprogramm entfernt, sind jedoch grundsätzlich innerhalb von MDX möglich.

```
1  import java.util.ArrayList;
2  import org.olap4j.*;
3  import org.olap4j.mdx.*;
4
5  SelectNode query = new SelectNode();
6
7  query.setFrom(new IdentifierNode(
8  new IdentifierNode.NameSegment("Datenwuerfel")));
9  query.getAxisList().add(
10   new AxisNode(null, false, Axis.ROWS,
11     new ArrayList<IdentifierNode>(), new IdentifierNode(
12       new IdentifierNode.NameSegment("Masszahl"))));
13
14 System.out.println(query.toString());
```

Listing 2: Erzeugen einer MDX Query mit olap4j. Nach [ebd.]

2.6 Google Web Toolkit

Das Google Web Toolkit (GWT)[14] ist eine Sammlung von Softwareentwicklungswerkzeugen und Programmierschnittstellen zur Entwicklung von AJAX[15] basierten Webanwendungen. Mit GWT entwickelte Webanwendungen werden in Java geschrieben. Der Client-Code wird durch das Toolkit in optimierten JavaScript Code cross-kompiliert [GU10]. GWT liegt zum Zeitpunkt der Implementierung des Report-Browsers in Version 2.1.1 vor. Es werden grundsätzlich alle wichtigen Webbrowser[16] unterstützt, diverse Einschränkungen gelten für Opera.

Die Entwicklung von Webanwendungen mit Hilfe von GWT kann in bekannten Integrated Development Environment (IDE) Anwendungen wie z.B. Eclipse

[14] GWT: `http://code.google.com/webtoolkit` (besucht am 01.05.2011).
[15] Asynchronous JavaScript and XML (AJAX) vergl. [Gar05].
[16] FireFox, Google Chrome, Internet Explorer, Safari, Opera.

oder Netbeans erfolgen. Alle Funktionen, die das Entwickeln von Java Anwendungen erleichtern, wie zum Beispiel Refactoring-Tools, Debugger und Autocompletion, stehen somit auch beim Erstellen von GWT Anwendungen zur Verfügung.

Das Google Web Toolkit unterscheidet zwei Arten, das entwickelte Programm auszuführen:

- **Development:**
  Im Entwicklungsmodus wird der Java Programmcode von GWT als Bytecode ausgeführt. Um die Anwendung in einem Browser zu starten, ist ein Development Mode Plugin nötig, welches einen JavaScript Emulator bildet und den Programmcode von der GWT Laufzeitumgebung ausführen lässt. Dieser Modus erlaubt es, GWT Anwendungen zu debuggen. Änderungen am Programmcode können durch erneutes Laden (aktualisieren) der Seite im Webbrowser nachvollzogen werden.

- **Deployment:**
  In dieser Variante wird JavaScript Code im Webbrowser direkt ausgeführt. Dieser Modus stellt den Produktionsmodus von GWT Anwendungen dar. Um Änderungen am Programmcode nachvollziehen zu können, muss das Programm erneut kompiliert werden. Dies ist je nach Rechenleistung zeitaufwändig, da normalerweise mindestens sechs Varianten des Programms erstellt werden. Startet eine GWT Anwendung in diesem Modus, wird die für den speziellen Webbrowser nötige Variante heruntergeladen und gestartet.[17]

Listing 3 zeigt ein simples „Hello World“ Programm mit GWT. Die dazu passende Modul Konfigurationsdatei sowie Startseite sind in Anlage A.1 enthalten. Die Konfigurationsdatei wird genutzt, um den GWT Compiler zu konfigurieren. Die Startseite enthält die Referenz zum JavaScript Code. Die hier referen-

[17] GWT Anwendungen lassen sich Internationalisieren. Ist eine Lokalisierung für die Sprache des Benutzers vorhanden, wird eine entsprechende Programm-Variante gestartet. Die Anzahl der zu kompilierenden Varianten erhöht sich um den Faktor der Anzahl der unterstützten Sprachen.

zierte (`...nocache.js`) JavaScript-Datei entscheidet über die zu startende Programm-Variante im Produktionsmodus.

```
1  package de.rwh.test;
2
3  import com.google.gwt.core.client.EntryPoint;
4  import com.google.gwt.user.client.Window;
5
6  public class HelloWorld implements EntryPoint
7  {
8    @Override
9    public void onModuleLoad()
10   {
11     Window.alert("Hello World");
12   }
13 }
```

Listing 3: GWT: Hello World

Wie klassische Java Enterprise Edition (Java EE) Webanwendungen sind GWT Anwendungen ebenfalls anfällig für diverse Angriffsszenarien. Das Open Web Application Security Project (OWASP) bewertet, auf ihrer zuletzt 2010 erschienenen Rangliste der Top 10 Risiken [OWASP10], Cross-Site Scripting (XSS) Angriffe als Angriffe mit dem zweit höchsten Sicherheitsrisiko für Webanwendungen. Das Risiko eines XSS Angriffes lässt sich in GWT Anwendungen mit Hilfe der Klassen im package `com.google.gwt.safehtml.shared` einfach reduzieren. So können z.B. mit Hilfe der Methode `sanitize(String html)` der Klasse `SimpleHtmlSanitizer` alle Hypertext Markup Language (HTML)-Tags bis auf `<b>`, `<em>`, `<i>`, `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`, `<ul>`, `<ol>`, `<li>` und `<hr>` in einem Text durch „escapen“ entfertn werden.

3 Entwurf

Der Abschnitt 3 zeigt, wie der Report-Browser entwickelt wird und welche Anforderungen an die Software gestellt werden. Ebenso wird beschrieben, welche architektonischen Entwurfsmuster genutzt werden um den Browser zu implementieren.

Sicherheitsrisiken, die für diese Software gelten, werden dargestellt, bevor dieses Kapitel mit einer Erläuterung der Verifikationsszenarien für den Report-Browser abgeschlossen wird.

3.1 Entwicklungsmethodik

Die Entwicklung des RWH Report-Browsers erfolgt im Rahmen dieser Arbeit in Interationen. Das heißt, die Software wird in Teile anhand einer zuvor festgelegten Priorisierung der Anforderungen implementiert. Anforderungen an die Software werden in einer vorangestellten Anforderungsanalyse definiert. Die Verifizierung der Anwendung erfolgt während der Implementierung mit Hilfe von Softwaretests und nach Abschluss der Implementierung mit Hilfe zweier Anwendungsfälle aus der Medizin (siehe 3.7 Seite 40).

Auf die Erstellung klassischer Anforderungsanalyse-Dokumente, wie z.B. dem Lasten- und Pflichtenheft, wird verzichtet, da die technische Machbarkeit erst im Rahmen dieser Arbeit festgestellt wird.

3.2 Anforderungsanalyse

Anforderungen an Software lassen sich grundsätzlich in zwei Gruppen aufteilen: zum einen funktionale Anforderungen, das heißt Anforderungen an das Verhalten einer Software sowie deren eigentlichen Funktionsumfang, zum anderen nicht-funktionale Anforderungen also Anforderungen an die Struktur und Architektur der Anwendung sowie Anforderungen an die Testbarkeit, Sicherheit, Verfügbarkeit und Interoperabilität.

Im Folgenden werden funktionale und nicht-funktionale Anforderungen an den RWH Report-Browser beschrieben und eine Priorisierung vorgenommen. Daraus ergibt sich der in 3.2.3 dargestellte Implementierungsplan.

3.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen des Report-Browsers wurden mittels Expertengesprächen innerhalb der Medizinischen Universitätsklinik definiert.

Multidimensionale Anfragen an das Research Warehouse (RWH)
Der Report-Browser dient als Plattform für klinisch wissenschaftliche- und Medizin-Controlling-Anfragen an das Research Warehouse (RWH) der Abteilung Innere Medizin III. Der Report-Browser stellt somit den zentralen Anlaufpunkt für Nutzer des RWH dar und soll an die Bedürfnisse und Fähigkeiten der zu erwartenden Benutzergruppe angepasst sein.

Darstellung von Anfrage-Ergebnissen
Der Report-Browser soll es ermöglichen Ergebnisse von Anfragen während der Erstellung der Anfrage einzusehen, um somit die Ergebnismenge leichter einschränken zu können.

Abspeichern von Anfragen
Um Anfragen einfach wiederholen zu können, sollen Anfragen an das RWH im Report-Browser abgespeichert werden können.

Verändern von gespeicherten Anfragen
Da sich Fragestellungen mit fortschreitender Zeit verändern können und um zeitversetztes Arbeiten an Anfragen zu ermöglichen, soll der Browser Funktionen zum Verändern bereits gespeicherter Anfragen anbieten.

Export von Anfrage-Ergebnissen
Viele Fragestellungen lassen sich anhand von einzelnen Kennzahlen beantworten. Diese Kennzahlen werden meist mit Hilfe von Statistik-Werkzeugen aus Datenmengen berechnet. Der Report-Browser soll es daher erlauben, Daten aus dem RWH in andere Werkzeuge zu exportieren. Dabei sollen die Datei-Formate CSV und XLS (Microsoft Excel) unterstützt werden.

Kollaboratives Erstellen und Bearbeiten von Anfragen
Der Report-Browser soll Gruppen Zugriff auf Anfragen und deren Ergebnisse ermöglichen. Dabei sollen Lese- und Schreib-Rechte für das Verändern von Anfragen vergeben werden können.

Öffentliche Anfragen zum Darstellen von Controlling-Kennzahlen
Um Controlling-Kennzahlen transparenter zu machen, sollen im Report-Brow-

ser Anfragen als öffentlich gekennzeichnet werden und somit die Ergebnisse von allen Benutzer eingesehen werden können.

Ergebnissdarstellung mit Drill-Down / Roll-Up Funktionalität
Funktionen zum einfachen Navigieren in Ergebnismengen mittels Drill-Down oder Roll-Up sollen implementiert werden.

3.2.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen an den Report-Browser wurden zum Teil ebenfalls mit Hilfe von Experteninterviews spezifiziert, ergeben sich jedoch auch zu anderen Teilen aus den technischen Gegebenheiten innerhalb der Abteilung der Medizinischen Universitätsklinik sowie aus der Rahmenarchitektur des RWH Projektes.

Abteilungsweite Verfügbarkeit
Zugriff auf den Report-Browser soll mindestens aus der Abteilung Innere Medizin III ermöglicht werden. Ein aufwendiger Installationsprozess sei dabei zu vermeiden.

Nur autorisierter Zugriff
Der Zugriff auf den Report-Browser und somit auf die Daten des RWH soll nur autorisierten Benutzern ermöglicht werden. Zur Authentifizierung der Benutzer soll das im Klinikum etablierte Active Directory (AD) integriert werden.

Interoperabilität mit Excel / Statistik-Programmen SAS und SPSS
Der Report-Browser soll als Softwarewerkzeug in den Datenauswertungsprozess von der Erfassung bis zur Auswertung mittels Statistik-Werkzeugen integriert werden. Eine Exportfunktionalität soll daher vorgesehen werden.

Maximale Testbarkeit der Anwendung mit Hilfe von Softwaretests
Um eine hohe Softwarequalität nachhaltig zu ermöglichen, sollen möglichst alle Komponenten des Report-Browsers automatisch durch Softwaretest getestet werden können.

Unabhängigkeit von den Datenstrukturen des OLAP Würfels
Damit der Report-Browser auch auf anderen OLAP Würfeln außerhalb des RWH Projektes eingesetzt werden könnte, soll der Browser Unabhängig von den Datenstrukturen des RWH OLAP Würfels implementiert werden und nicht direkt von diesen Strukturen abhängen.

Performante Ausführung von Abfragen
Bei der Ausführung von Anfragen an das RWH soll darauf geachtet werden, dass diese in angemessener Zeit ausgeführt werden und die Ausführungsgeschwindigkeit nicht durch die Report Anwendung verlangsamt wird.

3.2.3 Implementierungsplan

Nicht alle Anforderungen werden im Rahmen dieser Arbeit umgesetzt. Die folgende Liste liefert eine priorisierte Liste der umzusetzenden Funktionen und Anforderungen.

1. Konfiguration der Entwicklungsumgebung (Eclipse, Subversion), Konfiguration des Maven-Build Systems, Implementierung der Basisarchitektur sowie grundlegender UI-Elemente.

2. Integration des OLAP Servers und Technologischer „Durchstich“.

3. UI-Elemente zum Ausführen von nicht-generierten MDX-Anfragen und zur Darstellung von Anfrage-Ergebnissen.

4. UI-Elemente zur Generierung von MDX-Anfragen anhand der OLAP Metadaten und Implementierung des MDX-Anfragegenerator Algorithmus.

5. UI-Elemente und Datenbank Strukturen zum Abspeichern, Laden und Verändern von Anfragen.

6. Ergebnis Export im CSV- und XLS-Format.

3.3 Architektur

Der Report-Browser wird mit Hilfe einer Drei-Schichten Architektur erstellt. Auf der untersten Schicht befinden sich der SQL-Server mit einer relationalen Datenbank zum Abspeichern von Anfragen an das Data Warehouse und der OLAP-Datenbank Server, mit dem eigentlichen Data Warehouse sowie das Benutzerverzeichnis (Lightweight Directory Access Protocol (LDAP) / AD) [Abb. 7 re.]. Auf der mittleren Schicht befindet sich die Java EE Web-Server Applikation [Abb. 7 m.], auf der oberen Schicht der mit Hilfe des GWT Frameworks realisierte Web-Client [Abb. 7 li.].

Die Kommunikation zwischen Web-Client und Web-Server erfolgt via Hypertext Transfer Protocol (HTTP), Web-Server und SQL-Datenbank kommunizieren via Transmission Control Protocol (TCP), mit dem OLAP-Server wird eine HTTP Verbindung mittels XMLA Protokoll aufgebaut, die Kommunikation mit dem AD erfolgt per LDAP über TCP.

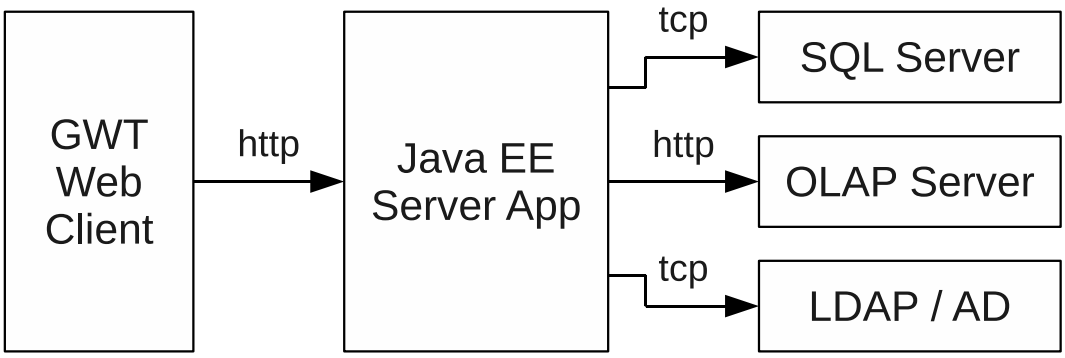


Abbildung 7: Drei-Schichten-Architektur des Report-Browsers

3.3.1 Client Architektur

Auf Grund der guten Entkopplung von einzelnen Modulen und Klassen [Rya09] sowie auf Grund der guten Testbarkeit von grafischen Oberflächen mit GWT [Dan10] wird die Oberfläche des Report-Browsers mit Hilfe des Model View

Presenter (MVP) Entwicklungsmusters realisiert [Pot96]. Alternativen zu diesem Muster bilden die Pattern Model View Controller (MVC) [Gam95, S. 4] und Model View View-Model (MVVM) [Smi09].

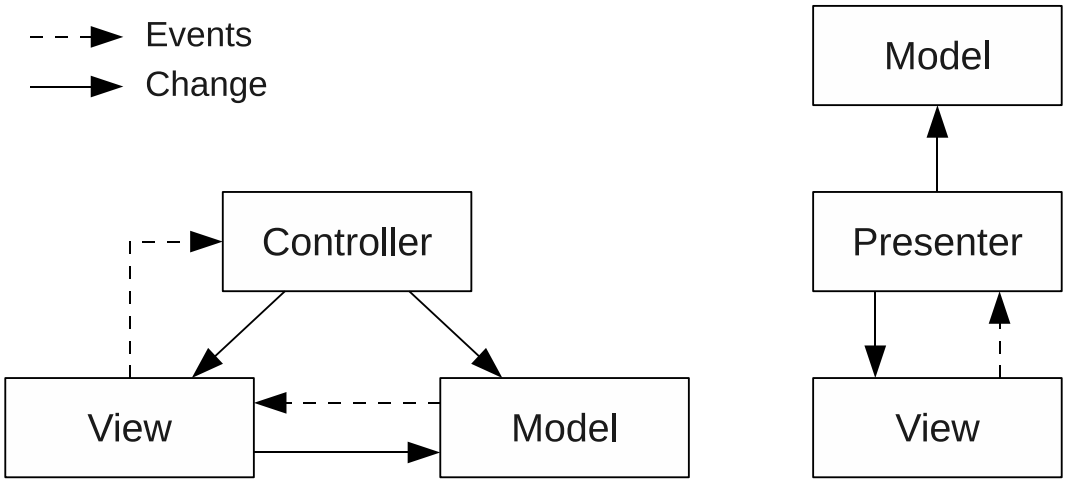


Abbildung 8: Gegenüberstellung: MVC und MVP. Nach [Rya09]

Während das MVVM Pattern hauptsächlich in Anwendungen rund um das Windows Presentation Foundation Framework Verwendung findet, wird das MVC Pattern wegen der größeren Abhängigkeit zwischen den einzelnen Klassen (siehe Abb. 8) nicht gewählt.

Im MVP Pattern übernimmt der Presenter folgende Aufgaben: Verändern des Models und Anzeigen des Models im View. Das View stellt in diesem Pattern ein Informationssenke dar, welche Änderungen durch den Benutzer per Events dem Presenter mitteilt. Das Model besitzt als Informationscontainer keine weiteren Aufgaben.

Im Rahmen des Report-Browser werden für alle Views Schnittstellen (Interfaces) definiert, da für Softwaretests das Document Object Model (DOM) des Browsers nicht zur Verfügung steht. Hier müssen die Views durch „Dummy“-Objekte, sogenannte „Mocks“, ersetzt werden.

Neben dem MVP Muster wird das Eventbus Pattern, nach [Rya09], zum Entkoppeln von einzelnen Modulen der grafischen Oberfläche verwendet.

3.3.2 Server Architektur

Der Serverteil des Report-Browsers wird logisch mit Hilfe des Fassaden-Entwurfsmusters [Gam95, S. 185] vom übrigen Teil der Anwendung getrennt.[18] Um einen definierten Punkt für die Implementierung von Filter- und Zugriffsregeln zu ermöglichen, wird die Fassade als Zuständigkeitskette (Cain of Reposibility Pattern) realisiert [ebd., S. 223]. Abbildung 9 zeigt, wie die Fassade vor die übrigen Klassen des Servers geschaltet ist.

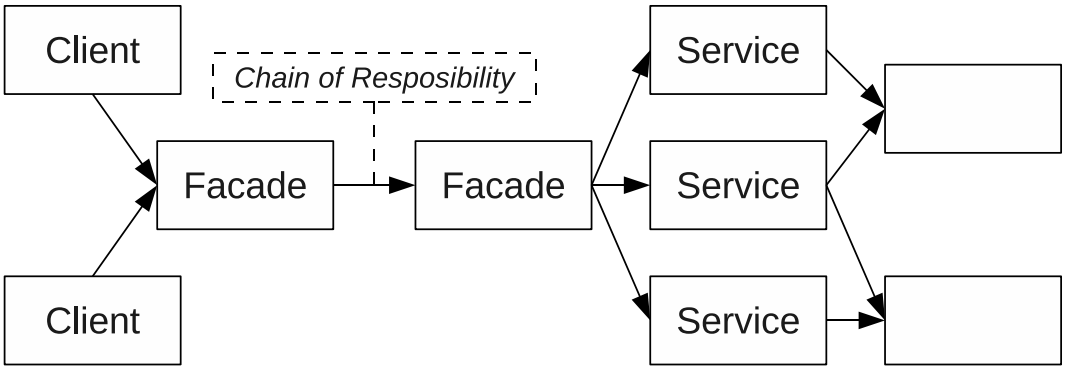


Abbildung 9: Facade- und Chain of Responsibility Pattern. Nach [ebd.]

Der Zugriff auf die relationale Datenbank wird über Klassen, die nach dem Data Access Object (DAO) Muster [SDN01] entwickelt werden, realisiert. Des weiteren wird ein Service zum An- und Abmelden des aktuellen Benutzer, aufbauend auf das Spring Security Framework[19] sowie ein Service für den Zugriff des OLAP-Servers erstellt.

3.3.3 Client-Server Kommunikation

Zur Übertragung von Daten zwischen GWT-Web-Client und der Java EE-Applikation wird auf den Remote Procedure Call (RPC) Mechanismus des

[18] Vergl. Reverse Proxy [Arc09].
[19] Spring Security: `http://static.springsource.org/spring-security/site` (besucht am 01.05.2011).

Google Web Toolkits aufgesetzt [GwtRpc]. Abbildung 10, aus [ebd.] zeigt, welche Klassen von GWT generiert werden (grün), und welche Klassen, mit einer konkreten Implementierung, selbst geschrieben werden müssen (blau).

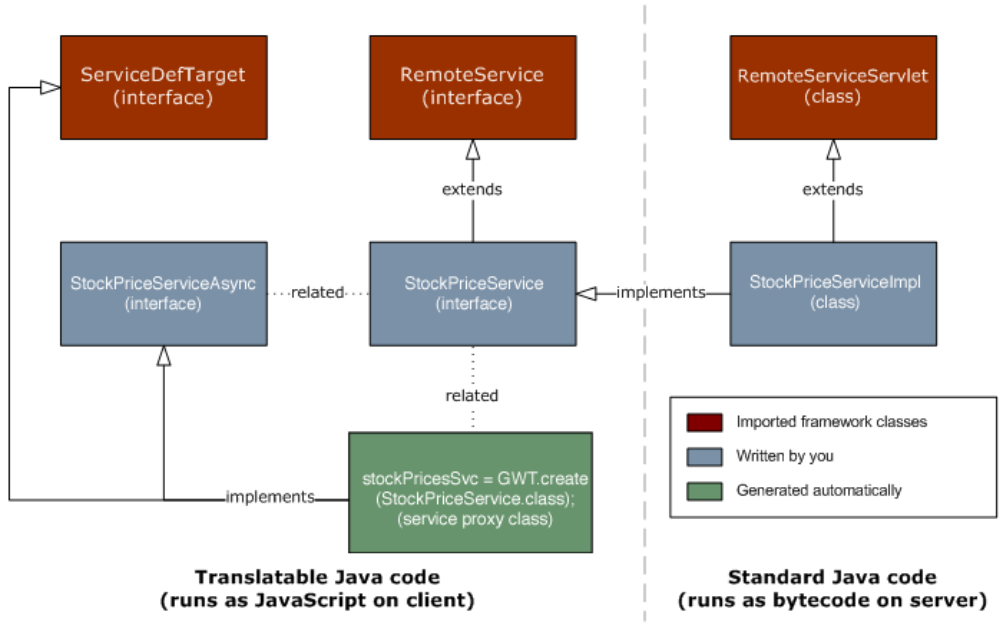


Abbildung 10: Klassen des GWT RPC Systems. Aus [ebd.]

Der GWT RPC Mechanismus dient der Übertragung von beliebigen serialisierbaren Java-Objekten.[20]

Neben den serverseitigen Domänen-Klassen werden Transfer-Klassen [Fow02, S. 401] realisiert, so kann die Implementierungen von Client und Server möglichst unabhängig von einander gehalten werden. Eine Verknüpfung dieser beiden spezialisierten Modelle wird durch Mapper-Klassen (im Quelltext als Converter bezeichnet) hergestellt [ebd., S. 473].

Der GWT RPC Mechanismus wird erweitert, in dem ein Command Pattern [Gam95, S. 233] zum Entkoppeln von Aufrufen von Funktionen der Applika-

[20] Java Marker Interface Serializable: `http://download.oracle.com/javase/6/docs/api/java/io/Serializable.html` (besucht am 01.05.2011).

tionsfassade aus der grafischen Oberfläche implementiert wird. Abbildung 11 zeigt das klassische Command Pattern. Der Client führt indirekt eine Funktion der Klasse Receiver aus, in dem ein konkretes Command zur Ausführung an den Invoker (Aufrufer) übergeben wird.

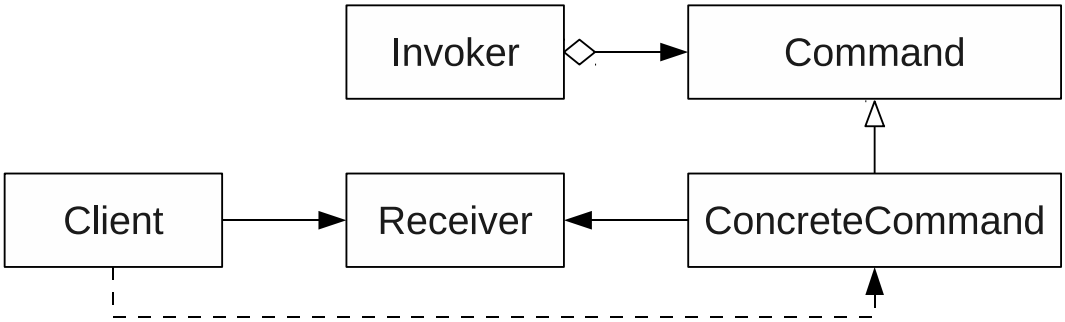


Abbildung 11: Command Pattern. Nach [ebd., S. 236]

Die Rolle des Aufrufers übernimmt im Report-Browser eine Klasse des Servers in einem serverseitigen Ausführungsschritt und eine Klasse des Clients in einem anschließenden clientseitigen Ausführungsschritt.

3.4 Softwaretests

Die integrität des Report-Browser wird mit automatischen Software-Tests sichergestellt. Die Tests werden in den Maven-Build Prozess integriert und bei jedem Build der Software automatisch ausgeführt. Die Test werden mit Hilfe des Testing-Frameworks JUnit[21] implementiert, hinzu kommen die Mocking-Funktionalitäten[22] des Frameworks EasyMock[23]. Tests der Datenbank Zugriffs-Klassen werden mit Hilfe der Hauptspeicher Datenbank HyperSQL[24] realisiert.

[21] JUnit: `http://junit.org` (besucht am 01.05.2011).
[22] Mock-Objekte bezeichnen Dummy-Objekte welche bei Modultests als Platzhalte für nicht mit getestete Abhängigkeiten genutzt werden.
[23] EasyMock: `http://easymock.org` (besucht am 01.05.2011).
[24] HyperSQL: `http://hsqldb.org` (besucht am 01.05.2011).

Die im Rahmen dieser Arbeit realisierten Komponententests fokussieren auf das Prüfen von isolierten Modulen (unit testing). Integrationstests lassen sich jedoch ebenso mit Hilfe der oben beschriebenen Technologien implementieren.

Da neben dem Server auch die grafische Benutzeroberfläche in Java geschrieben ist, lässt sich das Verhalten der Anwendung sowohl serverseitig als auch clientseitig testen. Automatisierte Integrationstests sind ebenso möglich, werden jedoch im Rahmen dieser Arbeit nicht realisiert.

3.5 Ausführungsgeschwindigkeit

Die Ausführungsgeschwindigkeit (Performance) von Web-Applikationen ist abhängig von diversen Faktoren und beeinflusst das Qualitätsempfinden der Anwendung und der dargestellten Daten, so Sears et al. [SJB97]. Ein wichtiger Faktor ist die Geschwindigkeit des Netzwerkes [Lei08]. Neben diesem spielen Zeiten zum Parsen und Ausführen von JavaScript, das Zeichnen und Layouten von Elementen der Benutzeroberfläche als Elemente des DOM, sowie die Ausführungsgeschwindigkeit der Server-Anwendung eine wichtige Rolle.

Um Aussagen über die Performance der Anwendung machen zu können, ist es wichtig, diese Werte zu messen. Nur mit Hilfe von konkreten Messwerten lassen sich Schwachstellen aufdecken, beziehungsweise feststellen, ob sich Änderungen positiv auswirken.

Das von Google entwickelte Werkzeug Speedtracer (ein Plugin des Google Webbrowsers Chrome) [25] bietet eine gute Möglichkeit, die oben genannten Parameter zu messen. Neben diesem auf Linux und Windows zur Verfügung stehenden Werkzeug ist mit dem Firefox Plugin Firebug ein Werkzeug gegeben, das ähnliche Funktionalitäten aufweist.[26]

[25] Speedtracer: `http://code.google.com/webtoolkit/speedtracer` (besucht am 01.05.2011).
[26] Firebug: `http://getfirebug.com` (besucht am 01.05.2011).

Messergebnisse der Ausführungsgeschwindigkeiten von Anfragen im Report-Browser finden sich im Abschnitt 5.4 (Seite 71).

3.6 Angriffsszenarien

Drei Angriffsszenarien stehen in Fokus dieser Arbeit. Es handelt sich dabei um die ersten drei Szenarien aus der Top 10 Liste des Open Web Application Security Project (OWASP), welche zuletzt 2010 veröffentlicht wurde [OWASP10].

3.6.1 Injection

Injection bezeichnet das Einschleusen von illegalen, nich-validierten Werten. Anfällig für diese Art von Sicherheitslücke sind SQL-Datenbanken, LDAP-Anwendungen[27] und diverse andere Anwendungen, in denen Benutzereingaben zum Ausführen von Befehlen genutzt werden.

Das folgende Beispiel aus [ebd.] zeigt, wie nicht nicht-validierte Parameter zum Zugriff auf alle Accounts genutzt werden können.

SQL:
```
String query = "SELECT * FROM accounts WHERE
    custID='" + request.getParameter("id") +"'";
```

Aufruf: `http://example.com/app/accountView?id=' or 1'='1`

Ausgeführter SQL Befehl:
```
SELECT * FROM accounts WHERE custID='' or 1'='1
```

[27] Lightweight Directory Access Protocol (LDAP).

Im Rahmen der Report-Browser Software werden sowohl die relationale- als auch die OLAP-Datenbank des Microsoft SQL-Servers eingesetzt. Beide Technologien sind grundsätzlich anfällig für Injection Angriffe.

3.6.2 Cross-Site Scripting (XSS)

Mit Cross-Site Scripting (XSS) werden Angriffe beschrieben, die dazu dienen, Programmcode oder Inhalte in Webseiten einzuschleusen. Diese Angriffe dienen meist dem Ausspähen von Vertrauenswürdigen Inhalten oder dem Beeinflussen des Benutzerverhaltens. Ausgenutzt werden hierbei Lücken im Interpreter des Webbrowsers. Quelle dieser Angriffe sind entweder direkte Benutzereingaben (Eingabefelder, URL's, Cookies) oder Werte, die in Datenbanken abgelegt sind und ungefiltert in Webseiten eingebunden werden [ebd.].

Das Beispiel hier zeigt, wie nicht validierte Benutzereingaben genutzt werden können, um JavaScript auszuführen. Nach [Wiki11a]

Webseite:
```
String page = "<div>Suchbegriff: <i>
      " + request.getParameter("item") + "</i></div>"
```

Aufruf:
```
http://example.com/app/searchView?item=<script
    type="text/javascript">alert("XSS");</script>
```

Ergebnis:
```
<div>Suchbegriff: <i><script type="text/
      javascript">alert("XSS");</script></i></div>
```

Im Webbrowser wird in diesem Beispiel somit JavaScript Code ungewollt ausgeführt.

Der Report-Browser wird als Webapplikation entwickelt und ist demnach grundsätzlich anfällig für XSS-Angriffe.

3.6.3 Broken Authentication and Session Management

In der Angriffsgruppe Broken Authentication and Session Management sind diverse Schwachstellen zusammengefasst, werden diese ausgenutzt ist es möglich illegalerweise an Informationen zu gelangen oder Befehle auf Datenbanken etc. auszuführen, ohne die dazu nötigen Berechtigungen zu besitzen.

Ein Beispiel in [OWASP10] zeigt, dass bei einem falsch implementierten Session Management über die URL[28] eine Session übernommen werden kann; falls man zum Beispiel den Link zu einem Produkt an jemanden weitergibt.

URL: `http://example.com/sale/saleitems;`
`jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?`
`dest=Hawaii`

3.7 Verifikationsszenarien

Mit Hilfe zweier Anwendungsfälle wird die Funktionsweise des Report-Browsers verifiziert. Diese Szenarien bilden einen Teil der zu erwartenden Anwendungsfälle für den Report-Browser und das RWH.

Das erste Szenario kommt aus dem Bereich des DRG-Managements.[29], während das an eine Arbeit aus der Medizinischen Universitätsklinik Heidelberg angelehnt ist.

3.7.1 Verweildauer

Thomas Grobe beschreibt in seiner Arbeit „Stationäre Versorgung - Krankenhausbehandlungen“ (2005), wie klinische Routinedaten für das Controlling der

[28] Uniform Resource Locator (URL).
[29] Diagnosis Related Groups (DRG).

Krankenkassen[30] in anderen Kontexten verwendet werden können [Gro05]. Grobe zeigt beispielhaft eine Auswertung, welche die durchschnittlichen Krankenhaustage für die Kapitel des ICD-10[31] für Frauen und Männer darstellt. Diese Auswertung [ebd., S. 96] ist Vorbild für dieses Szenario. In diesem Anwendungsfall werden §21 KHEntgG Daten zur Auswertung genutzt, welche ähnliche Inhalte abbilden, wie die von Grobe genutzten §301 SGB V Datensätze.

3.7.2 Kardiales Troponin

Ziel des RWHs ist es unter anderem, unterschiedliche Datenquellen gemeinsam auswertbar zu machen. Das hier beschriebene Szenario soll dies überprüfen. Von Steen et al. wurden 2007 in der Arbeit „Relative Role of NT-pro BNP and Cardiac Troponin T at 96 hours for Estimation of Infarct Size and Left Ventricular Function After Acute Myocardial Infarction“ die prognostischen Eigenschaften des kardialen Blutwertes Troponin T für die Vorhersage der Auswirkungen eines akkuten Myokardinfarkts auf die zu erwartende Pumpfunktion des Herzens beschrieben [Ste07].

Die Korrelation zwischen kardialem Troponin und der links ventrikulären Ejektionsfraktion (LV EF) des Herzen bei Patienten mit Myokardinfarkt soll in diesem Szenario für Datensätze im RWH aus der CPU- und der Kardio-MRT Datenbank gezeigt werden.[32] Zur Nachbildung der Ergebnisse aus [ebd.] werden die Ergebnisse von hochsensiblen Troponin T-Bluttest hsTnT[33] 24, 48 und 96 Stunden nach Infarkt aus der CPU Datenbank, sowie Links ventrikuläre Ejektionsfraktion (LV EF) Werte aus Kardio-MRT Untersuchungen verwendet. Troponin T Bluttests werden nach der Einführung der hsTnT Untersuchung in der CPU nicht mehr durchgeführt.

[30] Daten nach §301 Sozialgesetzbuch V (SGB V).
[31] International Statistical Classification of Diseases and Related Health Problems (ICD-10).
[32] Zum Zusammenhang zwischen kardialem Troponin T und links ventrikulärer Ejektionsfraktion siehe auch [Hu07].
[33] Hochsensibles Troponin T (hsTnT), Weiterentwicklung des Blutparameters Troponin T [Kur11].

4 Implementierung

Im Kapitel Implementierung werden wichtige Implementierungsdetails der Report-Browsers dargestellt. Neben dem Modulsystem des Client und der Zugriffsschicht des Servers wird beschrieben, wie die Kommunikation zwischen Client und Server abläuft.

Einen großen Abschnitt nimmt die Beschreibung der grafischen Oberfläche ein. Hier wird aufgezeigt, wie die Funktionen des Report-Browsers genutzt werden können. Im letzten Abschnitt wird der Algorithmus beschrieben, mit dem MDX Anfragen generiert werden.

4.1 Client: Modularisierung

Mit Version 2.0 des Google Web Toolkits wurde eine Funktionalität eingeführt, die es ermöglicht, große Client-Applikationen mit Hilfe des Java-To-JavaScript Cross-Compilers in nachladbare Module zu zerteilen [Joh09]. Dieser als „Code Splitting“ bezeichnete Teilprozess des GWT Compilers basiert auf statischer Codeanalyse. In dieser Analyse wird, ausgehend von sogenannten „Split Points“ errechnet, welche Codesegmente benötigt werden, um die gewollten Funktionen im Browser zur Verfügung stellen zu können.

Es ist also möglich, sämtliche Grafiken, Texte, UI-Elemente und Programmlogik, die für eine gewisse Funktionalität benötigt werden, hinter einer Schaltfläche, die diese Funktionen einleitet (z.B. einem Menüeintrag), zu verstecken und erst in diesem Moment asynchron nachzuladen.

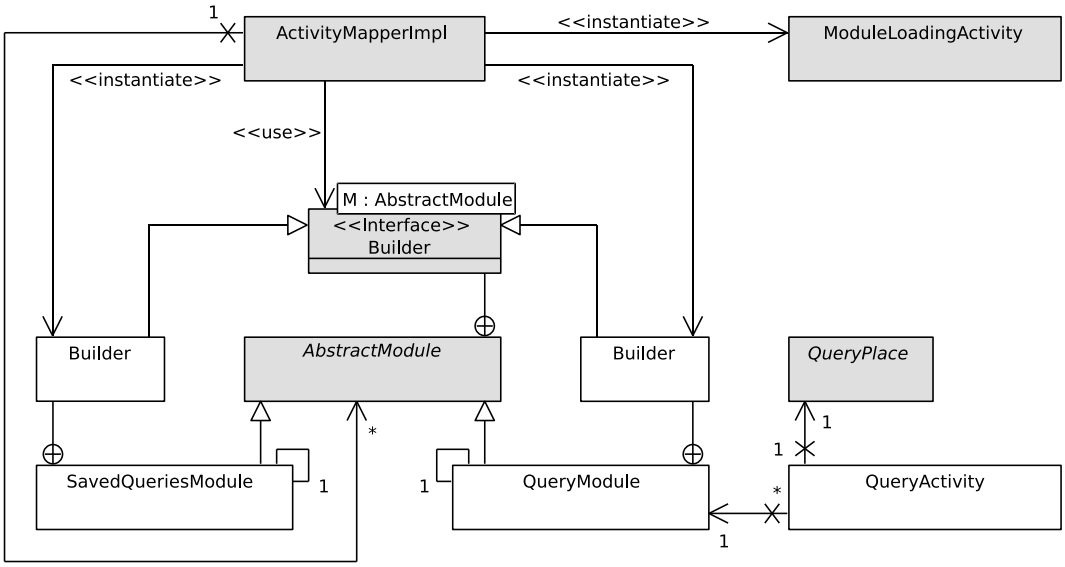


Abbildung 12: Klassendiagramm: Client-Modul-System

Abbildung 12 zeigt die für das Modulsystem benötigten Klassen. Ein konkretes Modul erbt von der Klasse „AbstractModule“ und beinhaltet als Factory die Zugriffspunkte auf sämtliche für das Modul benötigten Klassen, Grafiken, etc.. Diese konkreten Module definieren zudem eine „Builder“ Klasse, welche von der

„ActivityMapperImpl“ Klasse genutzt wird, um beim ersten Aktivieren einer Funktionalität „Activity“ den benötigten Code und alle weiteren Elemente, unter Zuhilfenahme eines asynchronen Server-Aufrufs, nachzuladen, zudem bei Eintreffen des Moduls (Callback) die gewünschte Funktionalität zu starten.

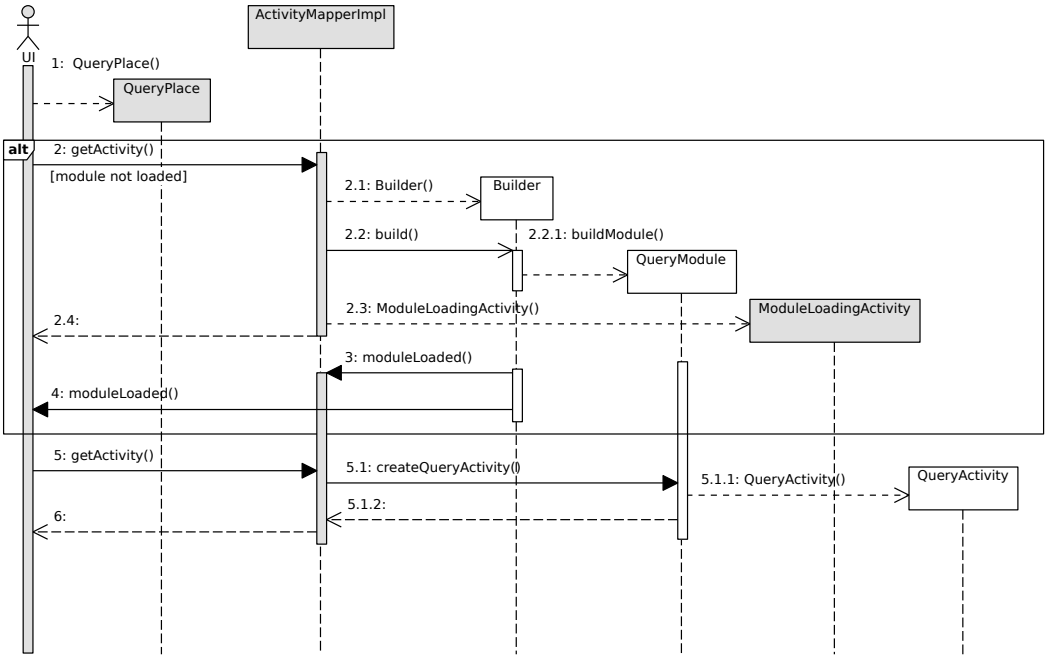


Abbildung 13: Sequenzdiagramm: Nachladen des Client-Moduls „Query“

Das in Abbildung 13 dargestellte Sequenzdiagramm zeigt beispielhaft den genauen Ablauf des Nachlade-Prozesses für das Query-Module.[34] Dieser beginnt mit der Navigation des Benutzers zum Applikationsort „QueryPlace“.[35] Während im Sequenzschritt 2.2.1 das Modul asynchron Nachgeladen wird, ist die Aktivität „ModuleLoadingActivity“ aktiv und gibt dem Benutzer eine optische Rückmeldung. Mit dem Callback „moduleLoaded“ des „Builders“ wird die eigentliche Funktionalität gestartet.

[34] Das Query-Module enthält alle Elemente die speziell für das Anzeigen und Bearbeiten von Anfragen und zum Darstellen von Ergebnissen benötigt werden.

[35] Funktionalitäten im Report-Browser werden als sogenannte „Activities“ modelliert. Startpunkte für diese Aktivitäten bilden „Places“. Diese Orte können entweder per URL oder per direktem JavaScript Aufruf gestartet werden [GwtMvp].

4.2 Client-Server Kommunikation

Um eine möglichst vollständige Entkoppelung von Client und Server Code zu erreichen, wird das Command Pattern genutzt. Im klassischen Command Pattern wird die Ausführung einer Funktion durch Aufrufen der „execute“ Methode eines speziellen Command-Objektes eingeleitet. Dieses Verhalten wird in der Report-Browser Architektur um zwei weitere Methoden im Command erweitert.

Commands enthalten eine „execute“ Methode, welche auf dem Server ausgeführt wird, um Funktionen des Servers zu starten (siehe auch Abschnitt 4.3 Server: Fassade). Ist diese Methode beendet oder bricht durch einen Ausnahmezustand ab, wird im Client entweder die „executeSucceeded“ oder die „executeFailed“ Methode des Commands aufgerufen.

Abbildung 14 zeigt die wichtigsten an der Client-Server Kommunikation beteiligten Klassen. Die Interfaces „RpcService“ und „RpcServiceAsync“ enthalten jeweils nur eine Methode. Diese „execute“ Methode, einmal in einer synchronen Variante (Server), einmal asynchron (Client), dient als Verbindungspunkt zwischen Client und Server.

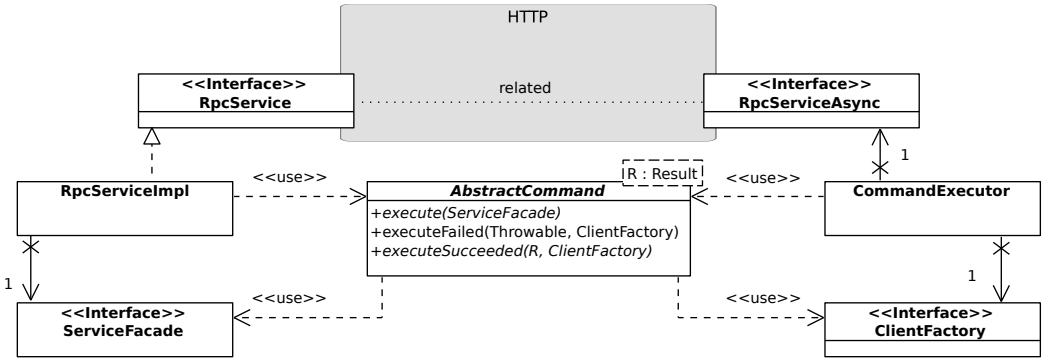


Abbildung 14: Klassendiagramm: Client-Server Kommunikation

Wird einer, durch GWT zur Verfügung gestellten Instanz, welche das Interface „RpcServiceAsync“ implementiert, ein konkretes Command durch den

CommandExecutor übergeben, wird dieses mit sämtlichen Datenfeldern serialisiert. Ist es per HTTP zum Server übertragen, wird es dort deserialisiert und dann die „execute“ Methode der RpcServiceImpl Klasse mit dem Command aufgerufen. Diese startet die „execute“ Methode des Commands und übergibt dieser eine Instanz der Server Fassade, als einzigen Zugriffspunkt auf die Funktionalitäten des Server.

Abbildung 15 verdeutlicht den Ablauf der Client-Server Kommunikation. Dem CommandExecutor wird hier das zum Ausführen von Anfragen an das Data Warehouse geschriebene Command „ExecuteQuery“ in Schritt 2 übergeben und dann in 2.1 asynchron an den Server übermittelt. Im Schritt 3.1.1.1 findet der eigentlich Funktionsaufruf im Server statt.

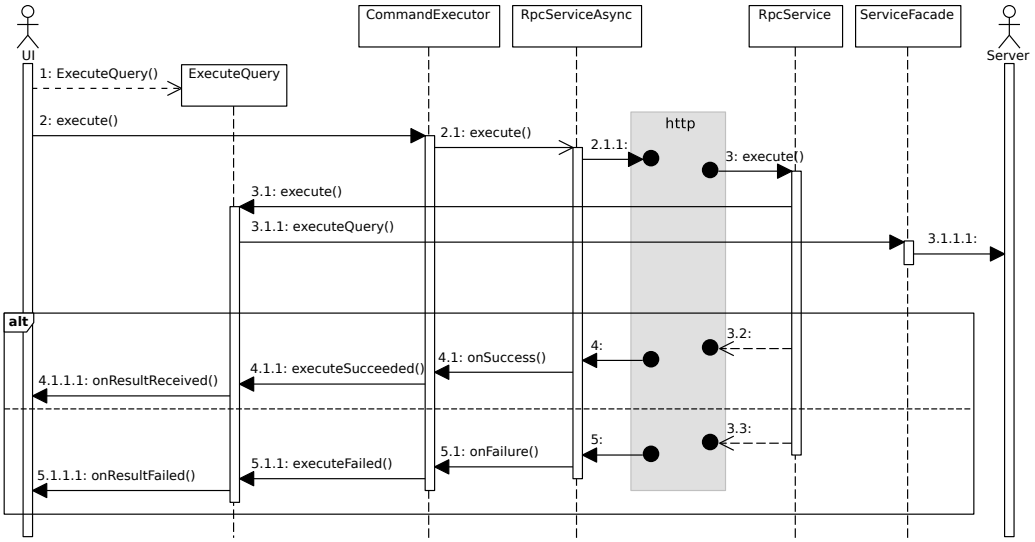


Abbildung 15: Sequenzdiagramm: Ausführen einer Anfrage

In 3.2 wird ein Ergebnis zurück zum Client übertragen, welches dort der „executeSucceeded“ Methode übergeben wird. In 3.3 ist der Ausnahmefall dargestellt: darin wird eine Exception zum Client übertragen und der Methode „ExecuteFailed“ übergeben.

4.3 Server: Fassade

Um die Geschäftslogik des Report-Browser vom Rest der Anwendung abzugrenzen und eine einheitliche Zugriffsschicht zur Verfügung zu stellen und zugleich Zugriffsrechte überprüfbar zu machen, wird das Facade Pattern in einer zweischichtigen Fassade umgesetzt. Die beiden Klassen „SecureServiceFacade“ und „BasicServiceFacade“ bilden eine Zuständigkeitskette (Chain of Resposibility). Unterschiedliche Service-, Datenzugriffs- und Mapper-Klassen werden von den beiden Fassaden-Klassen genutzt, um die Geschäftslogik nach außen abzubilden (siehe Abbildung 16).

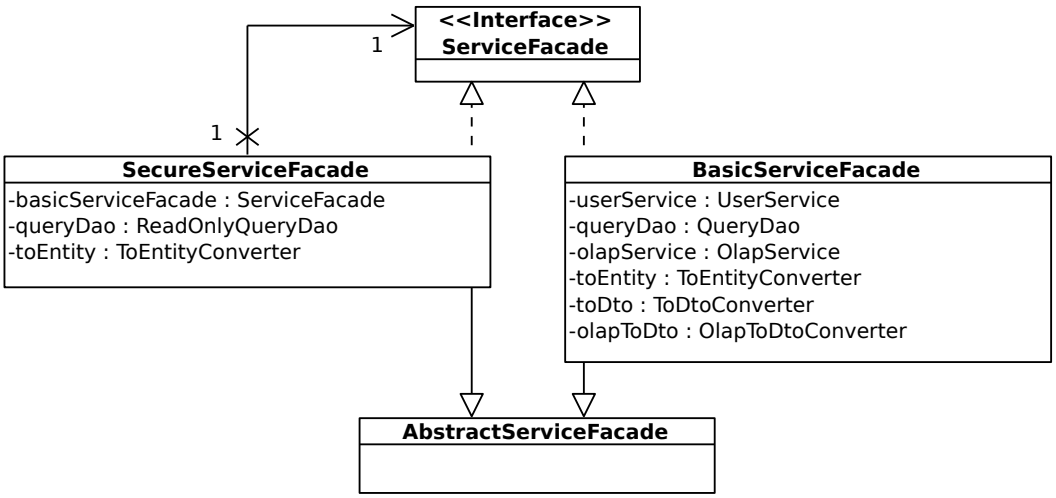


Abbildung 16: Klassendiagramm: Server Fassade als Zuständigkeitskette

Das Sequenzdiagramm in Abbildung 17 zeigt vereinfacht den serverseitigen Abblauf einer Data Warehouse Anfrage. In diesem Beispiel wird angenommen, dass die MDX-Anfrage durch den Anfragegenerator (Schritt 8.1) generiert wird. Dieser Schritt wird weggelassen, wenn eine MDX-Anfrage durch den Benutzer direkt eingegeben wurde.

Alle Aufrufe der Geschäftslogik laufen nach dem gleichen Prinzip ab: 1. Prüfung der Zugriffsrechte innerhalb des zur Laufzeit generierten Spring Security Proxies anhand der Annotation der Methoden in der „SecureServiceFacade“

Klasse [AT, S. 76], woraufhin ein Zugriff gegebenenfalls abgewiesen (Schritt 5) oder durchgelassen wird (Schritt 6).

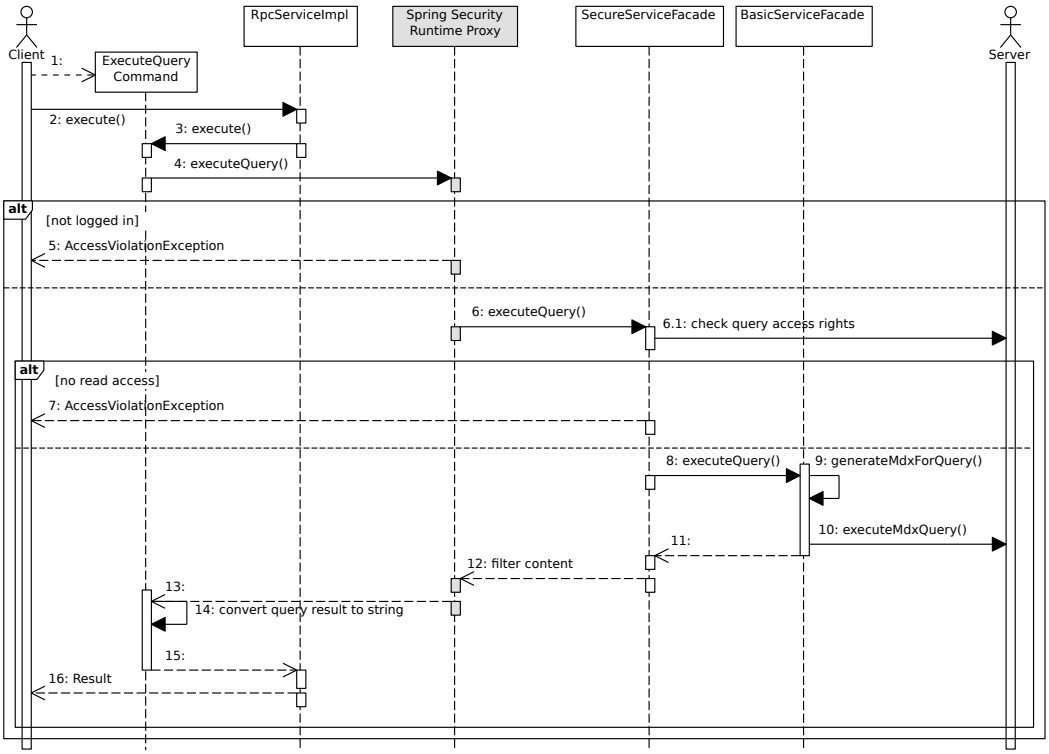


Abbildung 17: Sequenzdiagramm: Ausführen einer Anfrage auf dem Server

2. Aufruf der „BasicServiceFacade“ und Konvertierung der Transferobjekte in Domänen-Klassen. Danach folgt die eigentliche Geschäftslogik. Bevor von der „BasicServiceFacade“ ein Ergebnis zurückgegeben werden kann, erfolgt eine Konvertierung in die passenden Transferobjekte.

Im dritten Schritt findet in der „SecureServiceFacade“ eine manuelle - sowie im Spring Security Proxy eine automatische Filterung anhand von Annotation statt [ebd., S. 77].

Im Beispiel findet in der „SecureServiceFacade“ (Schritt 6.1) anhand der in der Datenbank gespeicherten Rechte für die gegebene Anfrage eine weitere Zugriffsprüfung statt, woraufhin auch hier gegebenenfalls die Anfrage abgewiesen wird.

4.4 Benutzerführung

Abbildung 18: Report-Browser: Startvorgang

Die folgenden Abbildungen (18 bis 29) liefern einen Überblick über die Funktionen der grafischen Oberfläche.
Während die Anwendung gestartet wird, ist Abbildung 18 zu sehen. Abbildung 19 zeigt die Login-Maske. Auskunft über einen nicht-erfolgreichen Login-Vorgang gibt die Statuszeile am unteren Bildschirmrand und ein Dialog.

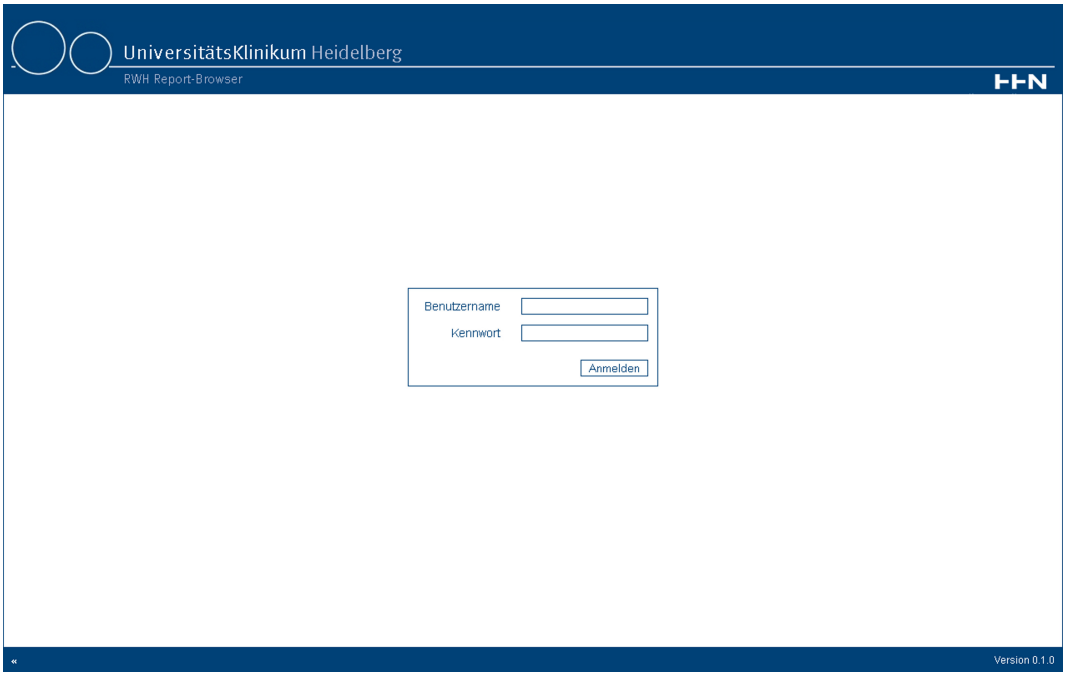


Abbildung 19: Report-Browser: Login

Hat sich ein Benutzer erfolgreich angemeldet, wird die Start Seite (Abb. 20) angezeigt und das Menü eingeblendet. Das Menü kann an die Rechte des angemeldeten Benutzers angepasst werden, so dass beispielsweise bestimmte Funktionen nicht angezeigt werden.
Wird im Menü „Neue Anfrage“ gewählt, wird das Client Module „Query“, zum starten der gewünschte Funktion, nachgeladen. Abbildung 21 zeigt die Anwendung, während ein Modul nachgeladen wird.

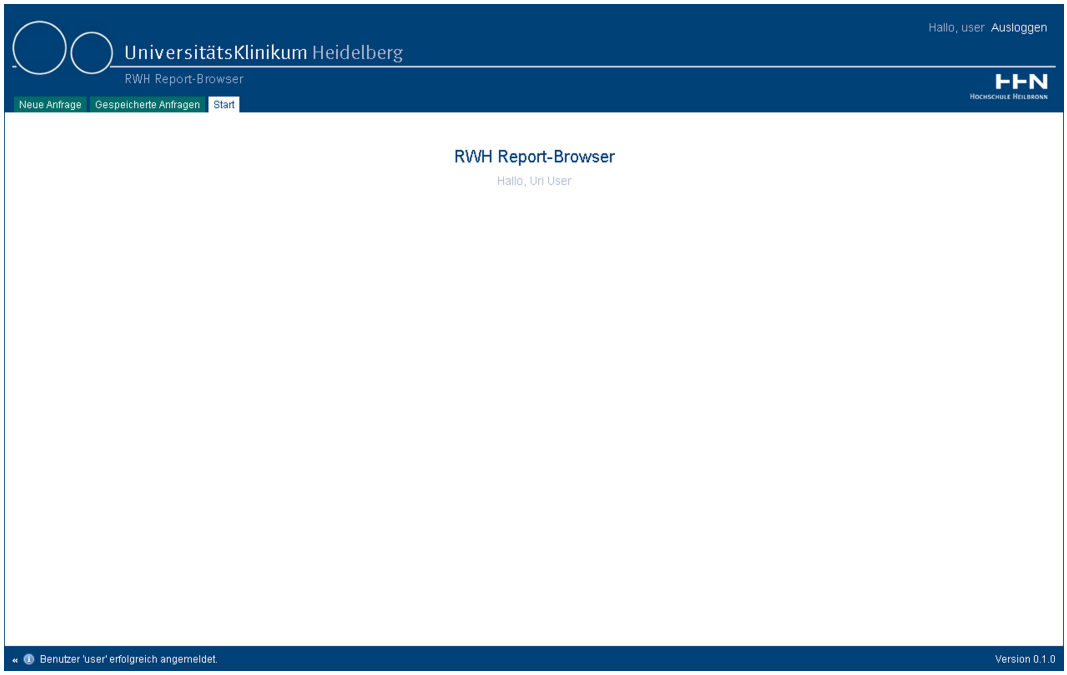


Abbildung 20: Report-Browser: Welcome

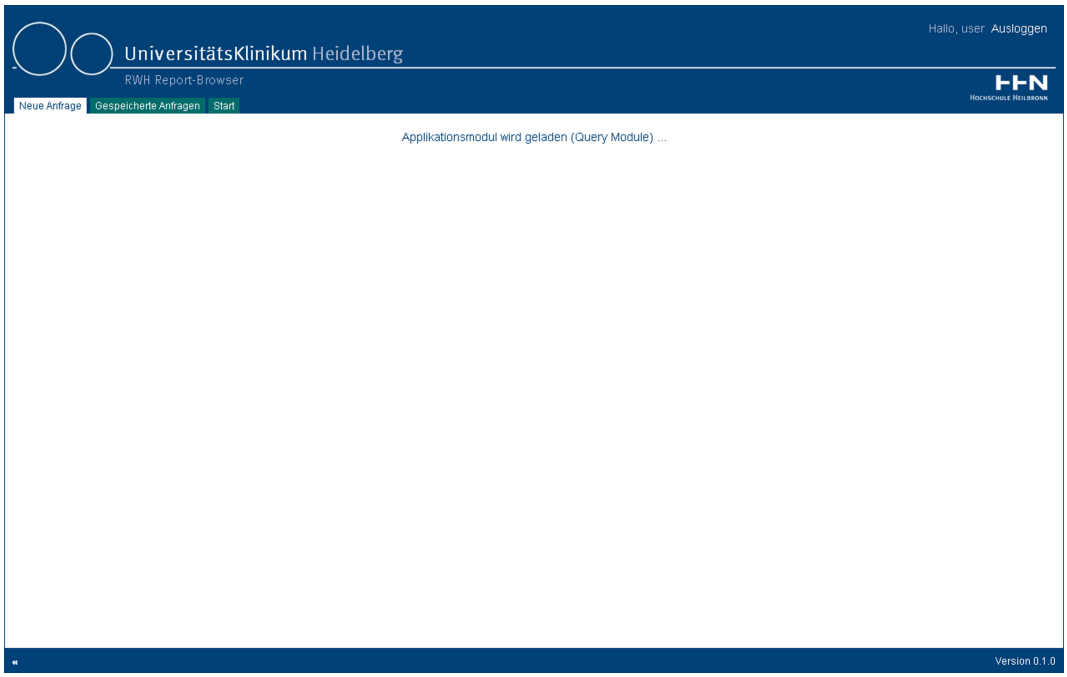


Abbildung 21: Report-Browser: Nachladen des Moduls „Query“

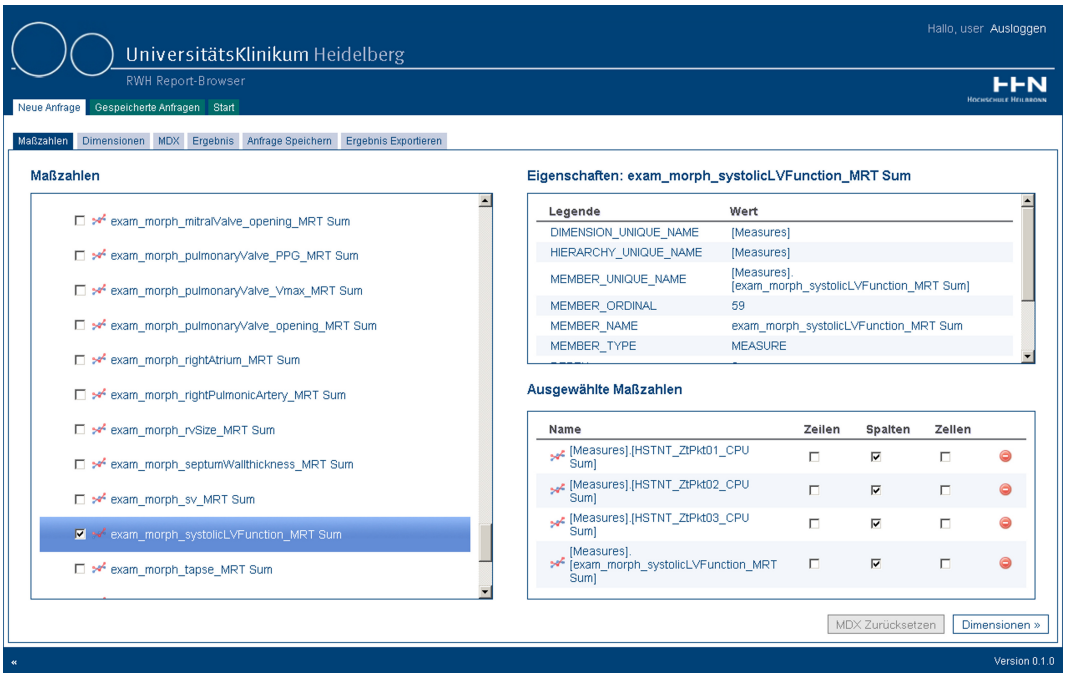


Abbildung 22: Report-Browser: Maßzahlen

Die Abbildungen 22 und 23 zeigen wie, mittels zweier Masken Maßzahlen (Abb. 22) und Dimensionen (Abb. 23) für die Generierung von Anfragen ausgewählt werden können.

Im linken Teil der Masken befindet sich eine Liste sämtlicher Maßzahlen beziehungsweise sämtlicher Dimensionen mit Hierarchien und Detailstufen zur Auswahl. Diese Elemente sind als Baum organisiert. Die Wurzel bilden die zur Verfügung stehenden Datenwürfel des OLAP Servers. Die Maßzahlen sind für jeden Würfel als Liste zu sehen. Dimensionen bilden die erste Ebene, darunter sind deren Hierarchien zu sehen und zuletzt können deren Detailstufen eingeblendet werden.

Der rechte Teil enthält unten eine Liste mit allen ausgewählten Elementen. Hier können für eindimensionale Abfragen Maßzahlen als Zeilen- oder Spaltenwerte gewählt werden. Für zweidimensionale Abfragen kann eine Maßzahl als Zellenwert bestimmt werden. Hierarchien und Level können als Zeilen- oder Spaltenbezeichnungen verwendet werden, hier besteht zudem die Möglichkeit,

den Messwert einer Hierarchie oder eines Detailstufen aggregiert darzustellen und nicht alle enthaltenden Member einzeln darzustellen.

Der obere rechte Teil der Masken enthält eine Liste sämtlicher zu einem in der linken Liste gewählten Element, verfügbaren Informationen. Hier kann zum Beispiel eine Beschreibung zu einer Maßzahl, wenn diese in der Datenbank hinterlegt ist, angezeigt werden.

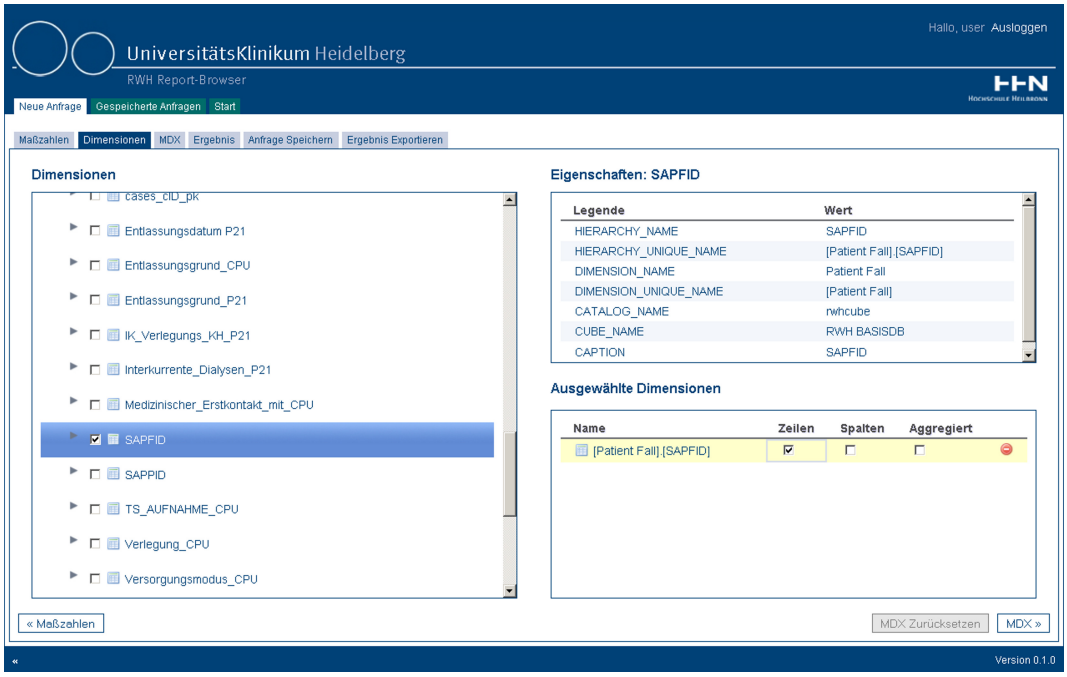


Abbildung 23: Report-Browser: Dimensionen

Wird zum dritten Reiter (Tab) navigiert, in dem dieser entweder direkt angeklickt wird oder die Schaltflächen am unteren Rand benutzt werden, wird die generierte MDX Anfrage sichtbar. Diese kann hier bearbeitet werden, was zu einer Deaktivierung der Schaltflächen in den Reitern Maßzahlen und Dimensionen führt. Siehe Abbildungen 39 und 40 im Anhang A.2.

Mit Hilfe der Schaltfläche „MDX Zurücksetzen“ wird die MDX Anfrage auf den generierten Text zurückgesetzt und kann über die ersten beiden Reiter wieder verändert werden.

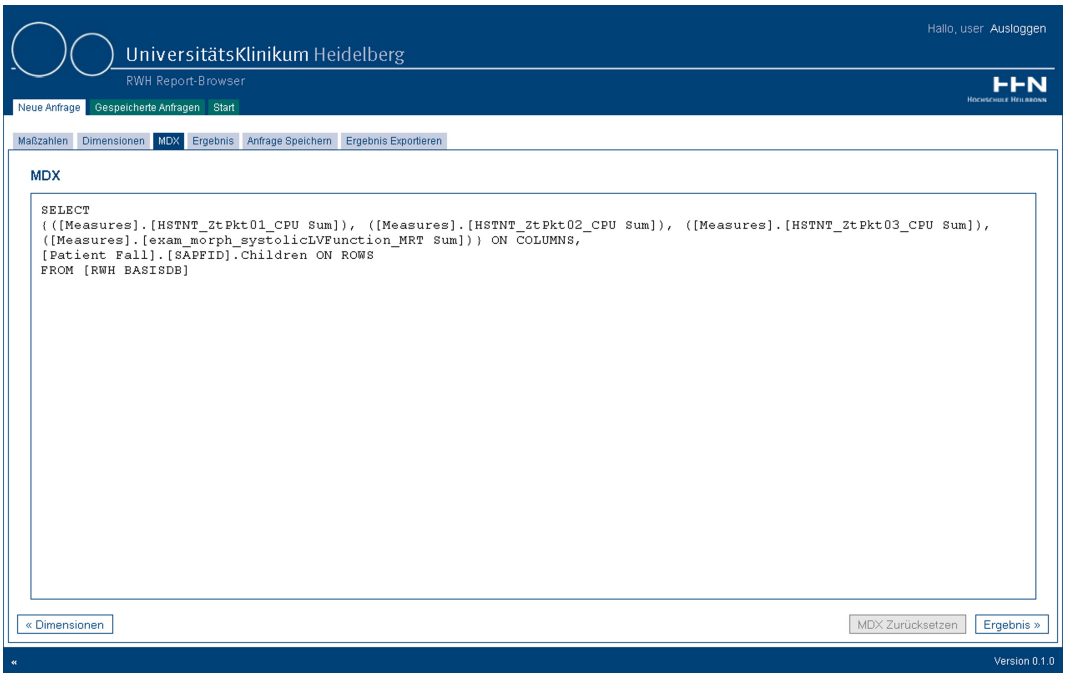


Abbildung 24: Report-Browser: MDX

Abbildung 25 zeigt wie, das Ergebnis einer Anfrage dargestellt wird. Während die Ergebnismenge berechnet wird, ist eine Meldung „Anfrage wird bearbeitet“ zu lesen (Abb. 37 im Anhang A.2). Dauert die Berechnung länger als 3 Sekunden wird, der Benutzer gefragt, ob die Auswertung abgebrochen werden soll. Dieser Dialog (Abb. 38 im Anhang A.2) erscheint alle 3 Sekunden erneut, wenn er mit „Nein“ beantwortet wird und verschwindet, sobald das Ergebnis eingetroffen ist.

Bevor ein Ergebnis zur Weiterverarbeitung mit Statistikwerkzeugen exportiert werden kann, muss die Anfrage abgespeichert werden. Abbildung 26 zeigt die Eingabemaske zum Abspeichern einer Anfrage. Hier kann neben einem Titel auch eine Beschreibung eingegeben werden. Wird eine Anfrage als „öffentlich“ markiert, ist diese unter den gespeicherten Anfragen für alle Benutzer sichtbar.

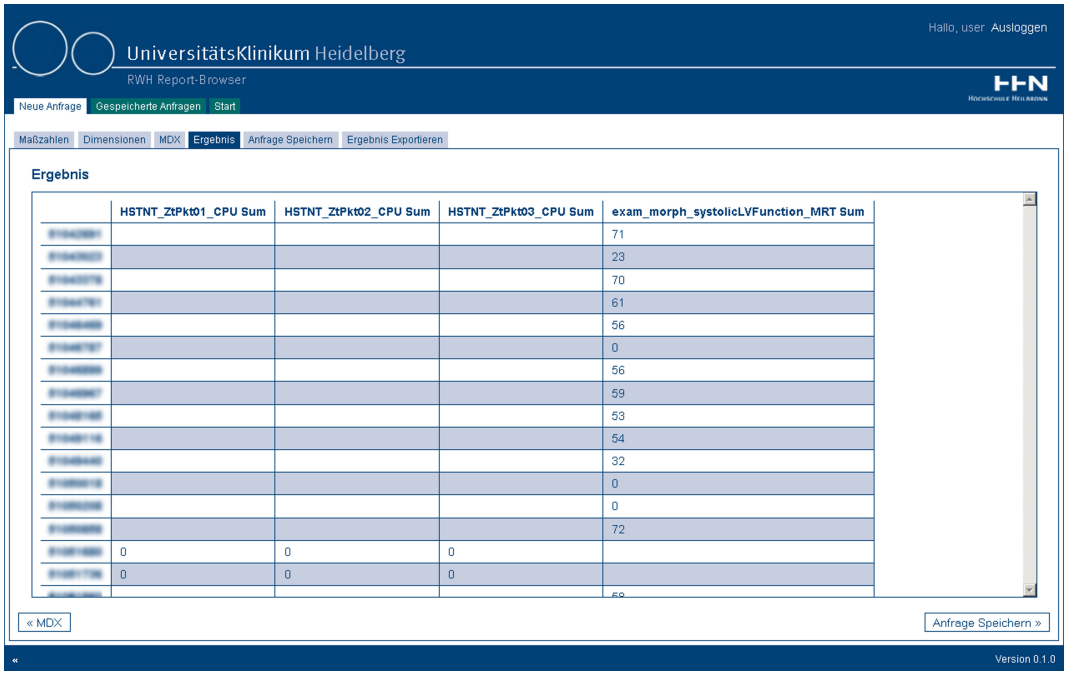


Abbildung 25: Report-Browser: Ergebnis

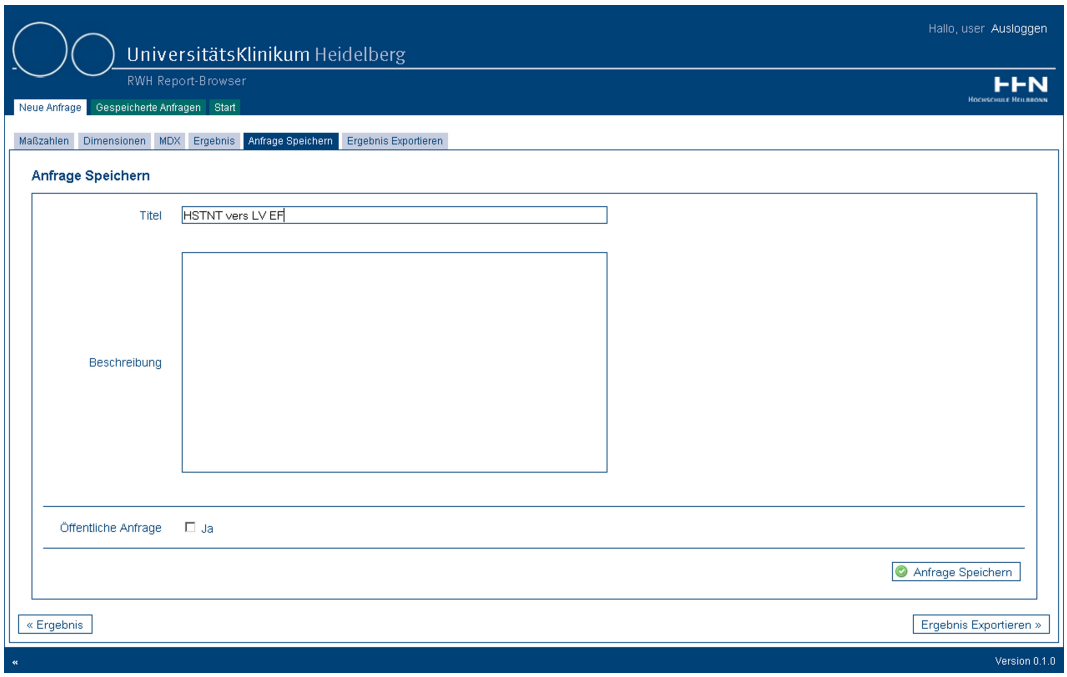


Abbildung 26: Report-Browser: Anfrage Abspeichern

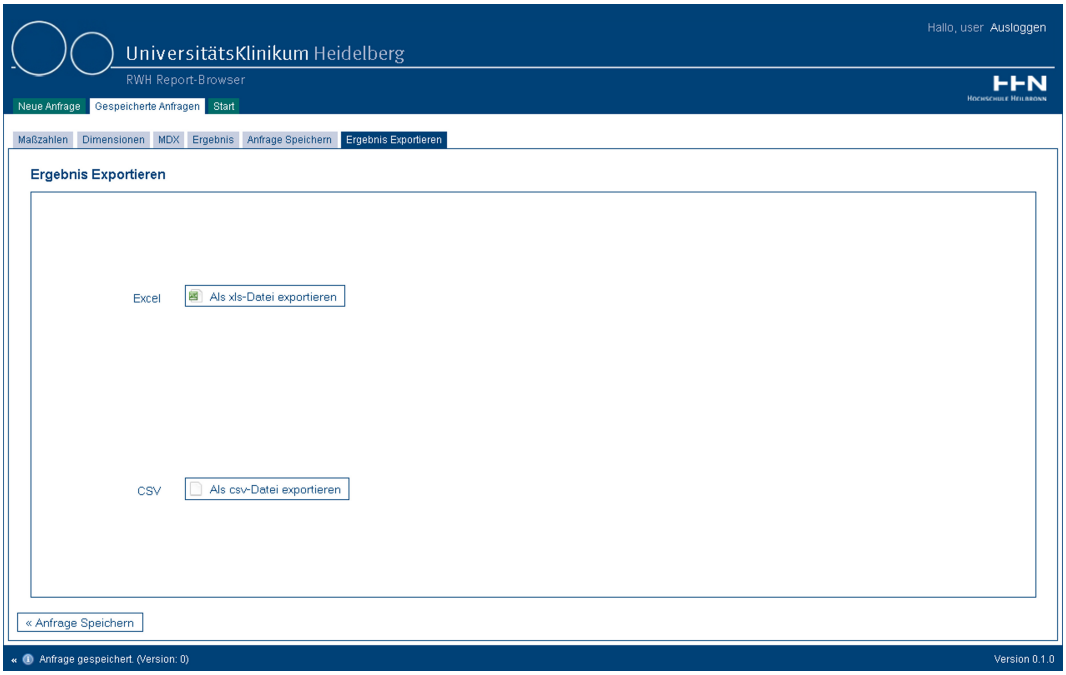


Abbildung 27: Report-Browser: Ergebnis Exportieren

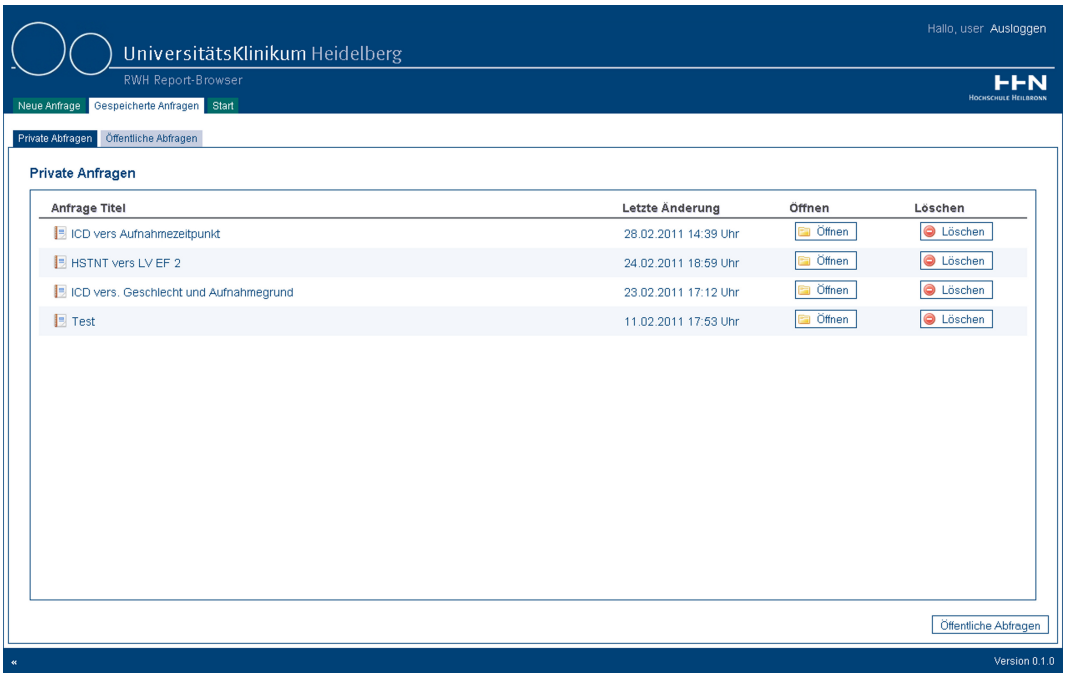


Abbildung 28: Report-Browser: Private abgespeicherte Anfragen

Ergebnisse können in zwei Formaten exportiert werden: zum einen im XLS-Format als Excel-Tabelle, zum anderen im CSV-Format als Komma getrennte Werte. Abbildung 27 zeigt die entsprechende Export-Maske. Wird ein Format gewählt, öffnet sich je nach Browser ein Dialog zum Abspeichern einer Datei. Das Verhalten des Firefox Browsers zeigt Abbildung 36 im Anhang A.2.

Hinter dem Menüeintrag „Gespeicherte Anfragen“ verbergen sich zwei Masken. Diese listen gespeicherte private Anfragen (Abb. 28) und gespeicherte öffentliche Anfragen (Abb. 29) auf. Wurde eine öffentliche Anfrage vom Benutzer selbst erstellt, ist diese in beiden Reitern zu sehen. Beide Listen sind nach dem letzten Änderungsdatum sortiert, so dass die neusten Anfragen oben aufgelistet sind.

Eine Anfrage kann mit Hilfe der Schaltfläche „Öffnen“ angezeigt werden. Es öffnet sich der Reiter „Ergebnis“ (Abb. 25). Des weiteren können private Anfragen nach einer Bestätigung mittels „Löschen“ entfernt werden.

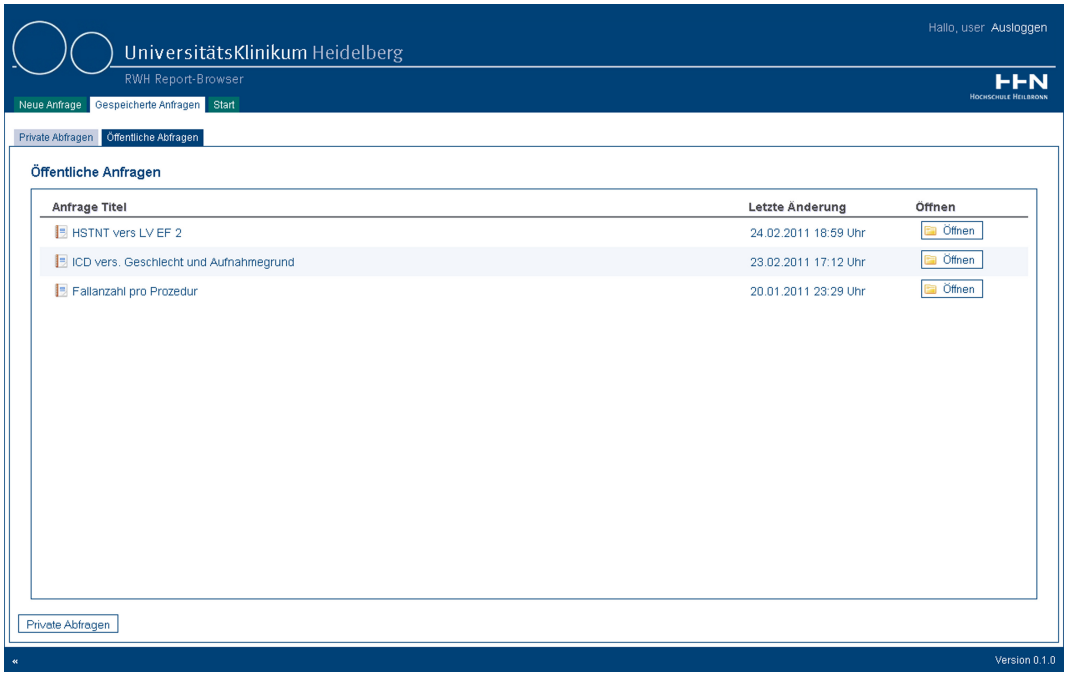


Abbildung 29: Report-Browser: Öffentliche abgespeicherte Anfragen

Alle dargestellten Eingabemasken haben, in Abhängigkeit des dargestellten Inhaltes, eindeutige Einträge in der Adresszeile des Webbrowsers. Werden diese URLs zum Beispiel per E-Mail an einen Kollegen gesendet, kann dieser direkt zum gleichen Inhalt, also zum Beispiel dem gleichen Ergebnis einer Anfrage, navigieren.[36]

4.5 Anfrage Generierung

Anfragen an ein OLAP basiertes Data Warehouse sind in der multidimensionalen Anfragesparche MDX zu formulieren.[37] Im Rahmen des Report-Browsers ist dies über die MDX Eingabemaske (Abb. 24) direkt möglich. Da jedoch davon ausgegangen werden kann, dass klinische Benutzer nicht in der Lage sind, ohne langwierige Einweisung MDX Anfrage selbständig zu formulieren, wurde für den Report-Browser ein Anfragegenerator realisiert.

Mit Hilfe der Eingabemasken für Maßzahlen und Dimensionen (Abb. 22 und 23) werden Elemente der Anfrage ausgewählt und durch die Anwendung in dem folgenden Modell abgespeichert. Abbildung 30 zeigt ein Diagramm der serverseitigen Klassen zur Repräsentation einer Anfrage.

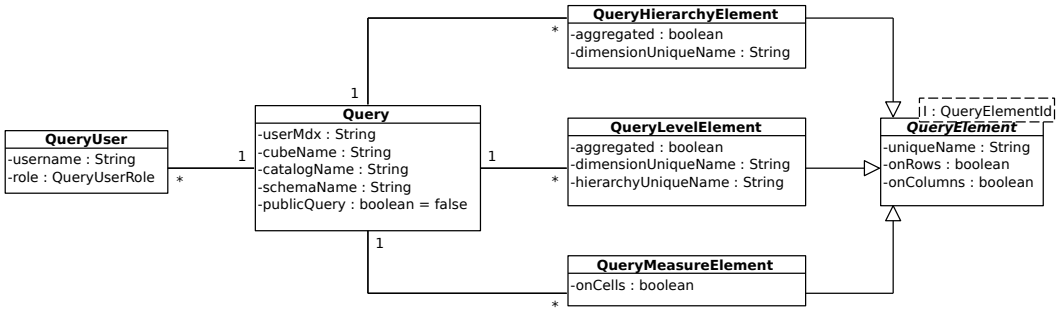


Abbildung 30: Klassen zur Repräsentation von Anfragen

[36] Wenn für diese Anfrage entsprechende Berechtigungen gesetzt sind. Ist der Benutzer nicht angemeldet, wird dieser zunächst zur Anmeldemaske umgeleitet.
[37] Mehr zu MDX siehe Abschnitt 2.4.3 (Seite 18)

Eine Anfrage besteht entweder aus einem einzelnen durch den Benutzer eingegebenen Text (String userMdx) oder aus mehreren Hierarchie-, Level- (Detailstufen) und Measure-Elementen (Maßzahlen). Die Zugriffsrechte auf eine gespeicherte Anfrage sind über das Datenfeld „role“ in der zugehörigen Klasse QueryUser geregelt. Hier kann zwischen Lese- und Schreibrechten unterschieden werden.

Wird dem Anfragegenerator ein Query-Objekt übergeben, berechnet diese die passende MDX-Anfrage anhand des in Abbildung 31 dargestellten Algorithmus. Aus Platzgründen ist der Algorithmus in der vorliegenden Grafik nebenläufig dargestellt, die vier vom Startpunkt (li.) ausgehenden Schritte werden jedoch hintereinander ausgeführt.

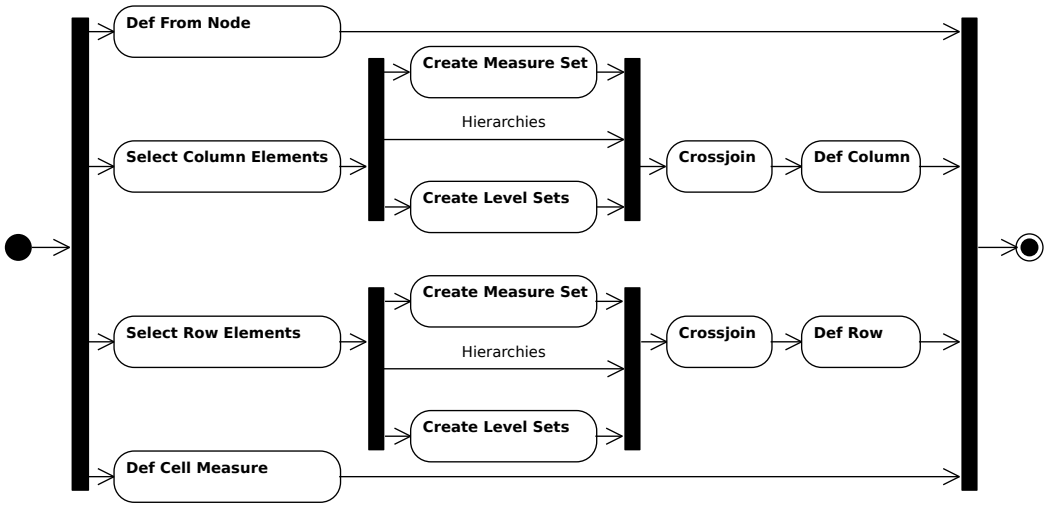


Abbildung 31: Algorithmus zur MDX-Anfrage Generierung

Der erste Schritt (Def From Node) definiert den „From“-Teil der Anfrage, der letzte Schritt (Def Cell Measure) den „Where“-Teil der Anfrage. Beide Schritte sind einfache Überführungsschritte.

Für die Zeilen- und Spaltenüberschriften werden zunächst alle passenden Elemente ausgewählt, dann werden mehrere Maßzahlen und Detailstufen anhand ihrer Hierarchie zu jeweils einer Mengen (Set) zusammengefasst. Anschließend

wird mit diesen Mengen sowie allen direkt ausgewählten Hierarchien ein Kreuzprodukt pro Achse definiert und dann als „Select ... On“ „Columns“ bzw. „Rows“ in der MDX Anfrage gesetzt.

Einschränkungen des Anfragegenerators und des Anfragemodells werden im Abschnitt 5.3 (Seite 67) deutlich.

5 Verifizierung

Im folgenden Kapitel werden die Ergebnisse der Verifikation des Entwurfs und der Zielsetzung diese Arbeit beschrieben.

Es werden sowohl Ergebnisse der Softwaretests und Aussagen über die grundsätzlich Testbarkeit der Softwarearchitektur als auch implementierte Abwehrstrategien zu den beschriebenen Angriffsszenarien aufgeführt. Außerdem werden die Ergebnisse der inhaltlichen Verifikation anhand zweier Szenarien wie auch die Ausführungsgeschwindigkeit der damit verbundenen Anfragen beleuchtet.

5.1 Softwaretests

Um eine hohe Code-Qualität für den Report-Browser sicherzustellen, werden automatische Softwaretests mit Hilfe der Test Bibliothek JUnit[38] erstellt. In diesen Modultests werden die Klassen des Report-Browser Quellcodes einzeln getestet. Abhängigkeiten zu anderen Modulen (Klassen) werden durch sogenannte Mocks, also Dummy-Klassen, ersetzt. Um diese Dummy-Klassen nicht selber schreiben zu müssen, wird das Framework EasyMock eingesetzt.[39] Mit EasyMock können Mock-Objekte zur Laufzeit generiert werden, für die das Aufrufverhalten des getesteten Moduls simuliert werden kann. Am Ende des Tests wird überprüft, ob das gewünschte Verhalten beobachtet wurde.

5.1.1 Testabdeckung

Im Rahmen dieser Arbeit wurden vor allem Unittest für die Klassen des Servers sowie für die zur Kommunikation zwischen Client und Server nötigen Klassen realisiert. Tabelle 1 zeigt die Testabdeckung des Report-Browsers.[40] In der Zeile „Server“ sind alle Instruktionen der Klassen zusammengefasst, die ausschließlich für den serverseitigen Quellcode benötigt werden. Die Zeile „Client“ listet die Ergebnisse für den clientseitigen Code auf, während in der Zeile „Shared“ Ergebnisse von Klassen verzeichnet sind, die sowohl auf der Server- als auch auf der Clientseite Verwendung finden.

In der Menge der Instruktionen des Client-Quellcodes sind ebenso 6858 Anweisungen aufgeführt, die auf Grund ihrer Abhängigkeiten zur Webbrowser-Laufzeitumgebung, also unter anderem zum DOM, derzeit nicht sinnvoll automatisch durch JUnit-Softwaretests getestet werden können. Es handelt sich dabei um Klassen aus dem Java Paket „de.rwh.browser.client.views.*“, welche

[38] JUnit: `http://junit.org` (besucht am 01.05.2011).
[39] EasyMock: `http://easymock.org` (besucht am 01.05.2011)
[40] Die hier vorliegenden Zahlen wurden mit Hilfe des Eclipse-Plugins EclEmma erstellt. EclEmma: `http://eclemma.org` (besucht am 01.05.2011).

	Instruktionen	getestete		nicht getestete	
Server	10949	4668	42,6 %	6281	57,4 %
Shared	7638	3215	42,1 %	4423	57,9 %
Client	16788	1034	6,2 %	15754	93,8 %
Σ	35375	8923	25,2 %	26452	74,8 %

Tabelle 1: Testabdeckung durch automatische Softwaretests

keine aufwändige Logik enthalten und grundsätzlich nur eine Verbindungsschicht zwischen HTML-Elementen und dem Java Quellcode darstellen.

5.1.2 Testbarkeit der Client-Architektur

Damit eine weitgehende Testbarkeit der Client-Anwendung gewährleisten werden kann wird im Report-Browser das MVP Entwurfsmuster umgesetzt (siehe Abschnitt 3.3.1). Alle Views werden in der Anwendung dabei durch Interfaces (Schnittstellen) repräsentiert. Dies ermöglichtes, die auf HTML-Elementen basierenden Implementierungen für den Webbrowser im Testfall durch Dummy-Objekte zu ersetzten.

Listing 4 zeigt einen Ausschnitt aus dem JUnit-Test zum Presenter LoginActivity (LoginActivityTest). Aufgabe des LoginActivity Presenters ist es, Eingaben des Benutzers in der Anmeldemaske auf Vollständigkeit hin zu überprüfen und den eigentlichen Anmeldevorgang auf dem Server durchzuführen. Anhang A.3 enthält einen ausführbaren Ausschnitt der Klasse LoginActivityTest.

Der hier gezeigte Testfall prüft, wie die Klasse LoginActivity auf das Betätigen der Schaltfläche „Anmelden“ reagiert, wenn weder Benutzername, noch Kennwort eingegeben wurden. In den Zeilen 4 bis 27 wird das erwartete Aufrufverhalten des Presenters aufgezeichnet. Hier werden alle erwarteten Methoden-Aufrufe und deren Übergabe- sowie Rückgabewerte simuliert.

Der Aufruf „replayAll()“ versetzt alle Dummy-Objekte in den Abspielmodus. In Zeile 33 wird der eigentlich zu testende Aufruf durchgeführt. Die Zeilen 35

bis 46 testen die Zustände der beteiligten Klassen und in Zeile 48 wird geprüft, ob alle zu erwartenden Aufrufe der Dummy-Objekte korrekt durchgeführt wurden.

```
80  @Test
81  public void testOnSubmitButtonClickedUnameAndPasswordInvalid() {
82    //record behavior of mocked classes
83    expect(loginViewMock.getUsername()).andReturn("");
84    expect(loginViewMock.getPassword()).andReturn("");
85
86    loginViewMock.setUsernameInvalid(true);
87    expectLastCall().once();
88    loginViewMock.setPasswordInvalid(true);
89    expectLastCall().once();
90
91    loginViewMock.setPasswordFocus(false);
92    expectLastCall().once();
93    loginViewMock.setUsernameFocus(true);
94    expectLastCall().once();
95
96    expect(footerMessagesMock.usernameAndPasswordInvalid()).
97      andReturn(UNAME_AND_PASSWORD_INVALID);
98
99    Capture<ShowFooterMessage> capturedMessage =
100     new Capture<ShowFooterMessage>();
101   Capture<LoginActivity> capturedActivity =
102     new Capture<LoginActivity>();
103
104   eventBusMock.fireEventFromSource(capture(capturedMessage),
105     capture(capturedActivity));
106   expectLastCall().once();
107
108   replayAll(); //replay behavior of mocked classes
109
110   LoginActivity loginActivity =
111     new LoginActivity(place, clientFactoryMock);
112   loginActivity.onSubmitButtonClicked(); //call tested method
```

```
113
114   FooterMessage message = (FooterMessage) ReflectionTestUtils.
115     getField(capturedMessage.getValue(), "message");
116
117   assertEquals(UNAME_AND_PASSWORD_INVALID, message.getText());
118   assertEquals(Type.Warning, message.getType());
119
120   assertEquals(loginActivity, capturedActivity.getValue());
121
122   assertTrue((Boolean) ReflectionTestUtils.
123     getField(loginActivity, "usernameInvalid"));
124   assertTrue((Boolean) ReflectionTestUtils.
125     getField(loginActivity, "passwordInvalid"));
126
127   verifyAll(); //verify behavior of mocked classes
128 }
```

Listing 4: Ausschnitt aus dem JUnit Test für den Presenter: LoginActivity

Nach dem hier beschriebenen Prinzip lässt sich das gesamte Verhalten des Presenters LoginActivity testen. Testfälle für weitere Presenter der Anwendung können in ähnlicher Weise implementiert werden.

5.2 Abwehrstrategien

Eine Überprüfung der Abwehrstrategie gegen die im Abschnitt 3.6 (auf Seite 38) beschriebenen Angriffsszenarien wird in dieser Arbeit, da es sich um eine Machbarkeitsstudie nicht durchgeführt. Im Folgenden werden nur die implementierten Gegenmaßnahmen dargestellt.

5.2.1 Injection

Zur Abwehr von Injection Angriffen gegen Datenbanken sollen Algorithmen zum Filtern von potenziell gefährlichen Inhalten, also durch die Benutzer ge-

neriert oder veränderbare Inhalte, sowie parametrisierte Datenbankanfragen (prepared statements) eingesetzt werden [OWASP10].

Alle Zugriffe auf die relationale Datenbank des Report-Browser zum Speichern, Lesen und Verändern von Anfragen werden mit Hilfe des Objektrelationalen Abbildungswerkzeugs Hibernate[41] durchgeführt. Darin integriert sind Mechanismen zum Filtern von Inhalten und zur Nutzung von parametrisierten Anfragen.

Ähnliches gilt für den Zugriff auf das RWH selbst. Hier wird das Framework olap4j eingesetzt. Zugriffe werden jedoch bisher nicht besonders gefiltert. Die Veränderung von Daten des RWH durch MDX Anfragen ist, aufgrund der Sicherheitseinstellungen in der RWH Datenbank, jedoch nicht ohne weiteres möglich.

5.2.2 Cross-Site Scripting (XSS)

Als bevorzugte Methode zur Abwehr von XSS Angriffen gilt die Filterung von Inhalten. Dafür ist es nötig, potenziell gefährliche HTML-Elemente, die JavaScript enthalten oder referenzieren können, aus Benutzereingaben zu entfernen, bevor diese wieder in eine Webseite integriert werden [ebd.].

Alle nicht statischen Inhalte des Report-Browser werden, mit Hilfe der in das GWT integrierten Funktionen[42] zum Filtern von HTML, vor der Darstellung von gefährlichen Inhalten befreit.

5.2.3 Broken Authentication and Session Management

Um ein korrektes Verhalten beim Authentifizieren von Benutzern und beim Anlegen, Benutzen und Zerstören von Sessions zu gewährleisten, wird das Spring

[41] Hibernate: `http://hibernate.org` (besucht am 01.05.2011).
[42] Siehe Abschnitt 2.6 (Seite 23).

Security Framework[43] in die Report-Browser Anwendung integriert. Zudem werden Session-Ids nicht in URLs preisgegeben und bei jedem Anmelden neu zufällig vergeben.

Das Abmelden und damit einhergehende Löschen der Session erfolgt nach 30 Minuten[44] automatisch und kann durch die Benutzer jederzeit über die grafische Benutzeroberfläche manuell durchgeführt werden.

5.3 Inhaltliche Verifikation

Die im Abschnitt 3.7 beschriebenen Szenarien werden in dieser Arbeit dazu genutzt, um festzustellen, ob der Report-Browser grundsätzlich dazu geeignet ist wissenschaftliche Fragestellungen aus den Bereichen DRG-Management sowie Grundlagenforschung und Versorgungsforschung in der Medizin zu beantworten.

5.3.1 Verweildauer

Im ersten Verifikationsszenario soll die Verweildauer stationärer Patienten, getrennt für Frauen und Männer, anhand der ICD-10 Diagnose dargestellt werden. Im Report-Browser wird dazu der Anfragegenerator benutzt und als Maßzahl die durchschnittliche Verweildauer aus den §21-Daten gewählt. Als Dimensionen werden für die Spaltenüberschrift das Geschlecht der Patienten und für die Zeilenüberschriften die Ausprägungen der ICD-10 Diagnose gewählt.

Listing 5 zeigt die durch den Report-Browser generierte Abfrage. Mit Hilfe des Schlüsselwortes „Children“ werden alle Ausprägungen einer Dimension im Ergebnis aufgelistet.

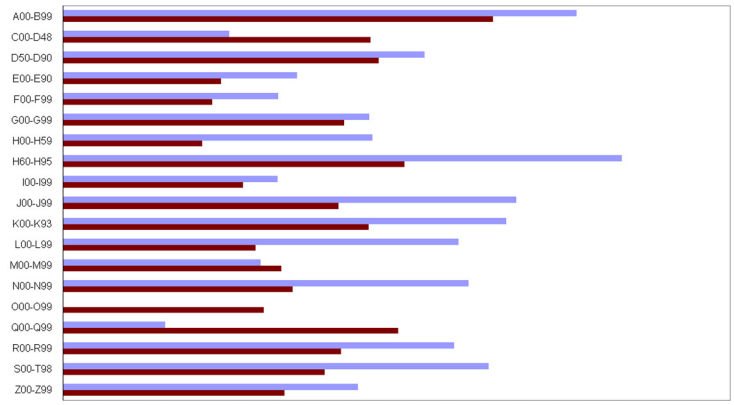
[43] Spring Security: `http://static.springsource.org/spring-security/site` (besucht am 01.05.2011).
[44] Dieser Wert kann konfiguriert werden.

```
1 SELECT
2 [Patient Patient].[Geschlecht_P21].Children ON COLUMNS,
3 [Krankheitsbild Diagnosen_P21].[ICD_P21].Children ON ROWS
4 FROM [RWH BASISDB]
5 WHERE [Measures].[AvgVwd21]
```

Listing 5: Abfrage: Verweildauer (generierter MDX Code)

Das Ergebnis dieser Anfrage umfasst 1140 Zeilen und 2 Spalten und zeigt für sämtliche gespeicherten ICD-10 Diagnosen die durchschnittliche Verweildauer, getrennt für Männer und Frauen. Derzeit steht keine ICD-10 Dimension, bzw. Hierarchie zur Verfügung, die ICD-10 Diagnosen in den 12 Kapiteln des ICD-10 zusammenfasst.

Abbildung 32 zeigt eine im Nachhinein durchgeführte Zusammenfassung der Ergebnisse in den ICD-10 Kapiteln.



	Z00-Z99	S00-T98	R00-R99	Q00-Q99	O00-O99	N00-N99	M00-M99	L00-L99	K00-K93	J00-J99	I00-I99	H60-H95	H00-H59	G00-G99	F00-F99	E00-E90	D50-D90	C00-D48	A00-B99
Männer	13,24	19,13	17,57	4,583		18,23	8,859	17,78	19,91	20,37	9,638	25,13	13,9	13,75	9,649	10,52	16,23	7,462	23,08
Frauen	9,937	11,75	12,48	15,07	9	10,31	9,813	8,643	13,72	12,36	8,067	15,33	6,25	12,63	6,689	7,086	14,18	13,82	19,33

Abbildung 32: Verweildauer [Tage] stationärer Patienten

Eine Überprüfung der dargestellten Ergebnisse wurde nicht durchgeführt. Für das ICD-10 Kapitel „Schwangerschaft, Geburt und Wochenbett“ (O00-O99) liegt nur für Frauen ein Ergebnis vor, dies lässt vermuten, dass die Daten valide sind.

5.3.2 Kardiales Troponin

Mit Hilfe des zweiten Verifikationsszenarios soll überprüft werden, ob sich die Ergebnisse (Korrelation von hsTnT Blutwerten und LV EF Messwerten) des Papers „Relative Role of NT-pro BNP and Cardiac Troponin T at 96 hours for Estimation of Infarct Size and Left Ventricular Function After Acute Myocardial Infarction“ von Steen et al. [Ste07] mit Hilfe des Report-Browsers und den im RWH gespeicherten Daten reproduzieren lassen.

Listing 6 zeigt die mit dem Report-Browser generierte MDX Anfrage an das RWH. Als Maßzahlen (Measures) wurden drei Erhebungen des hsTnT Blutwertes aus der CPU Datenbank sowie der LV EF Messwert aus der Kardio-MRT Datenbank gewählt (siehe Abb. 22 auf Seite 52). Diese werden für alle Patienten, anhand ihrer Fallnummer, aufgelistet (Auswahl der Dimension siehe Abb. 23 auf Seite 53).

```
1 SELECT
2 {([Measures].[HSTNT_ZtPkt01_CPU Sum]),
3       ([Measures].[HSTNT_ZtPkt02_CPU Sum]),
4       ([Measures].[HSTNT_ZtPkt03_CPU Sum]),
5       ([Measures].[exam_morph_systolicLVFunction_MRT Sum])}
6       ON COLUMNS,
7 [Patient Fall].[SAPFID].Children ON ROWS
8 FROM [RWH BASISDB]
```

Listing 6: Abfrage: LV EF - hsTnT (generierter MDX Code)

Das Ergebnis dieser Anfrage (ausschnittsweise zusehen in Abb. 25 auf Seite 55) umfasst 4 Spalten und 25.306 Zeilen. Innerhalb dieser Daten finden sich nur wenige Patienten, für die sowohl hsTnT Blutwerte als auch LV EF Messwerte vorliegen. Um die Ergebnismenge zu verkleinern, wurde der generierte MDX Code angepasst.

Listing 7 zeigt die angepasste MDX Anfrage. Die Zeilen 7 bis 13 sind gegenüber der generierten Anfrage verändert; hier werden Patienten ausgeblendet, für die

keine LV EF Messwerte vorliegen und bei denen der hsTnT Blutwert zum zweiten Erhebungszeitpunkt leer ist. Schlussendlich werden die Daten, aufsteigend nach den hsTnT Werten der zweiten Erhebung, sortiert.

```
SELECT
{([Measures].[HSTNT_ZtPkt01_CPU Sum]),
      ([Measures].[HSTNT_ZtPkt02_CPU Sum]),
      ([Measures].[HSTNT_ZtPkt03_CPU Sum]),
      ([Measures].[exam_morph_systolicLVFunction_MRT Sum])}
      ON COLUMNS,
Filter(
  Order(
    NonEmpty([Patient Fall].[SAPFID].Children,
      [Measures].[exam_morph_systolicLVFunction_MRT Sum]),
      [Measures].[Measures].[HSTNT_ZtPkt02_CPU Sum], DESC),
      [Measures].[exam_morph_systolicLVFunction_MRT Sum] <> 0
      AND [Measures].[HSTNT_ZtPkt02_CPU Sum] <> null) ON ROWS
FROM [RWH BASISDB]
```

Listing 7: Abfrage: LV EF - hsTnT (angepasster MDX Code)

Eine ausschnittsweise Darstellung des Ergebnisses im Report-Browser zeigt Abbildung 33. Das gesamte Ergebnis umfasst 4 Spalten und 126 Zeilen.

Eine Korrelation der hsTnT Blutwerte mit dem LV EF Messwert konnte in der Ergebnismenge nicht festgestellt werden. Dies ist auf die nicht durchgeführte Selektion der Patienten, anhand der im Aufsatz von Steen et al. beschriebenen Einschlusskriterien (u.a. Diagnose), zurückzuführen [Ste07].

Die in der RWH Datenbank erfolgte Verknüpfung der Datensätze aus den CPU und Kardio-MRT Datenbanken konnte in den Quellsystemen überprüft werden und ist korrekt.

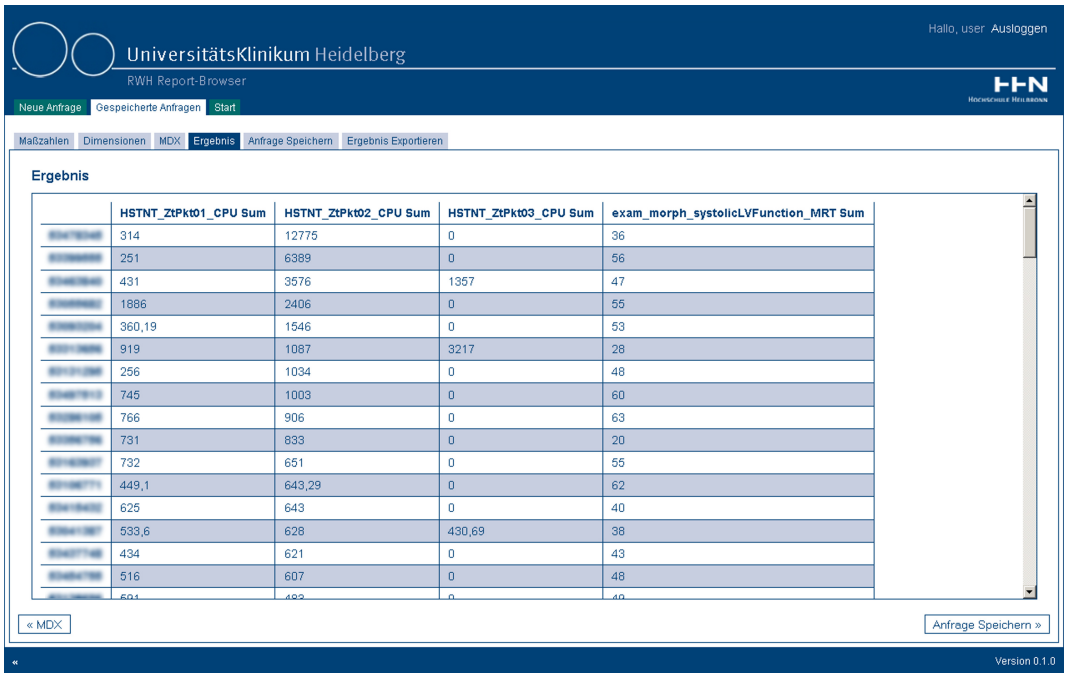


Abbildung 33: Report-Browser: LV EF - hsTnT Ergebnis (angepasster MDX Code)

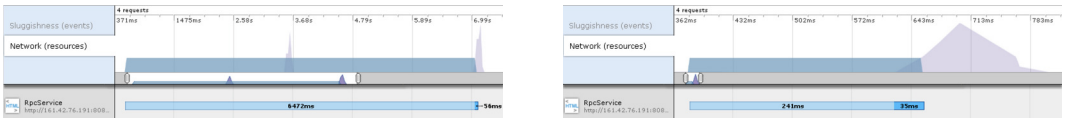
5.4 Ausführungsgeschwindigkeit

Das Qualitätsempfinden der Benutzer einer Anwendung ist unter anderem abhängig von der Performance, also der Ausführungsgeschwindigkeit der Anwendung [SJB97].

Eine erste objektive Bewertung der Ausführungsgeschwindigkeit des Report-Browsers ist mit den folgenden Ergebnissen möglich. Tabelle 2 zeigt die Ausführungsgeschwindigkeit für drei Anfragen an das RWH. Für die erste Messung „Verweildauer“ wurde das erste Verifikationsszenario benutzt. Die Messwerte der Szenarien „LV EF - hsTnT 1 und 2“ basieren auf der generierten- und auf der angepassten Anfrage aus dem zweiten Verifikationsszenario. Diese Messwerte können mit Hilfe des Werkzeugs Speedtracer erhoben werden.[45]

[45] Speedtracer: `http://code.google.com/webtoolkit/speedtracer` (besucht am 01.05.2011).

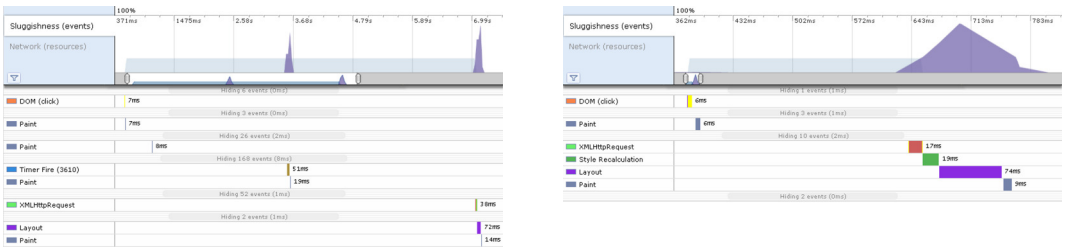
Abbildung 34 zeigt den Serverzugriff beim Ausführen der Anfrage „Verweildauer“ links (a), ohne dass bereits ein Ergebnis zwischengespeichert ist und rechts (b) mit zwischengespeichertem Ergebnis. Alle Geschwindigkeitstests wurden mit Hilfe von gespeicherten Anfragen durchgeführt.



(a) ohne Cache (b) Cachetreffer

Abbildung 34: Netzwerk: Anfrageausführung

Das erste Ereignis in den Zeitachsen der Abbildung 35 „DOM (click)“ ist das Betätigen der Schaltfläche „Öffnen“ in der Liste der gespeicherten Anfragen (siehe Abb. 28/29 Seite 56/57). Nach ca. 10 ms (erster „Paint“ Eintrag) wird dem Benutzer eine Rückmeldung darüber gegeben, dass die Anfrage im Moment bearbeitet wird (siehe Abb. 37 im Anhang A.2). Ist das Ergebnis nicht innerhalb von 3 Sekunden eingetroffen, wird nachgefragt, ob die Berechnung abgebrochen werden soll (siehe Abb. 38 im Anhang A.2). In der Abbildung 35 ist dieses Verhalten an der 3.6 Sekunden Marke nach Start der Aufzeichnung zu sehen (Timer Fire (3610)).



(a) ohne Cache (b) Cachetreffer

Abbildung 35: UI: Anfrageausführung

Im Cachetreffer Fall (b) trifft das Ergebnis nach ungefähr 240 ms ein und wird angezeigt, im Falle des ersten Zugriffes (a) erst nach circa 6.5 Sekunden. Im letzteren Falle wird dann automatisch die Abbruchmeldung ausgeblendet, falls diese nicht durch den Benutzer bereits entfernt wurde.

Die nachfolgende Tabelle 2 zeigt deutlich, dass der zweite Zugriff auf das RWH immer deutlich schneller ist als der erste. Dies ist auf den im olap4j Framework eingebauten Zwischenspeicher zurückzuführen.

Szenario	Spalten	Zeilen	Daten	Zugriff 1	Zugriff 2	Zugriff 3
Verweildauer	3	1141	7262 B	6472 ms	241 ms	209 ms
LV EF - hsTnT 1	5	127	21057 B	1149 ms	362 ms	342 ms
LV EF - hsTnT 2	5	25307	185 kB	152,9 s	4482 ms	4134 ms

Tabelle 2: Geschwindigkeitsvergleich Anfrageausführung

Die Ausführungsgeschwindigkeit der ersten Anfrage ist abhängig von der Komplexität der Anfrage und von der Größe der Ergebnismenge (Zeilen mal Spalten), alle weiteren Zugriffe (bei Cachetreffern) hängen nur von der Menge der zu übertragenden Daten ab.

6 Diskussion

Das letzte Kapitel dieser Arbeit beleuchtet die Ergebnisse. Neben einer inhaltlichen Beurteilung aus Sicht der Anwender wird eine technologische Bewertung vorgenommen.

Die Diskussion listet Maßnahmen auf, die vor der Übergabe des Report-Browsers in den Routinebetrieb durchgeführt werden müssen und zeigt das Potenzial der Anwendung in einem Ausblick auf.

6.1 Beurteilung

Ziel dieser Arbeit ist die Durchführung einer Machbarkeitsstudie. So soll geprüft werden, ob sich ein Abfragewerkzeug entwickeln lässt, das klinischen Benutzern ermöglicht, Medizin-Controlling- und medizinische Fragestellungen mit Hilfe der OLAP Datenbank des RWH Projekts zu beantworten. Ein wichtiger Fokus liegt dabei auf der Wahl einer geeigneten Softwarearchitektur.

Im Rahmen dieser Arbeit wurde der Report-Browser als Webanwendung entwickelt. Dieser wird im Abschnitt 6.1.1 inhaltlich, das heißt aus Sicht der klinischen Benutzer, beurteilt. Eine technische Beurteilung findet sich im Abschnitt 6.1.2, hier wird der Fokus auf technologische Probleme und Lösungen gesetzt. Aus der inhaltlichen und technischen Beurteilung wird deutlich, dass Anforderungen an ein Anfragewerkzeug aufgedeckt wurden, welche vor der Übergabe des Report-Browsers in einen Routinebetrieb noch umgesetzt werden müssen. Diese Anforderungen werden im Abschnitt 6.2 beschrieben. Die Diskussion schließt mit einer abschließenden Bewertung und einem Ausblick auf mögliche Erweiterungen des Report-Browsers im Abschnitt 6.3 ab.

6.1.1 Inhaltliche Beurteilung

Die folgende inhaltliche Beurteilung soll deutlich machen, ob der Report-Browser den Anforderungen der klinischen Benutzer gerecht wird. Im Rahmen der Anforderungsanalyse dieser Arbeit wurden acht funktionale und sechs-nicht funktionale Anforderungen definiert (Abschnitt 3.2, Seite 28). Die wichtigsten funktionalen Anforderungen sind: Erstellen von Abfragen an das RWH und Darstellen von Ergebnissen.

Im Abschnitt 5.3 (Seite 67) des Kapitels Verifizierung wird gezeigt, wie mit dem Report-Browser Anfragen an das RWH gestellt werden können. Der in 5.3 beschriebene und in 4.4 (Seite 50) mit den Abbildungen 22 bis 24 dargestellte Anfragegenerator wird im Report-Browser benutzt, um Anfragen zu stellen, ohne MDX Quelltext schreiben zu müssen. Die Verifikationsszenarien in 5.3

machen jedoch deutlich, dass mit dieser Art Anfragen zu formulieren, diverse Einschränkungen verbunden sind.

Das Szenario „Verweildauer“ zeigt die große Abhängigkeit zum Dimensionsmodell der OLAP Datenbank. Das Ergebnis der Anfrage muss hier noch bearbeitet werden, um die eigentlich gewollte Aussage - durchschnittliche Verweildauern für die ICD-10 Kapitel - darstellen zu können. Hier wird deutlich: Nur wenn geeignete Dimensionen, Hierarchien und Detailstufen im Datenmodell vorhanden sind, können die gewünschten Ergebnisse einfach produziert werden.

Eine weitere Einschränkung des Report-Browsers macht das Verifikationsszenario „LV EF - hsTnT“ deutlich: Allein durch die manuelle Anpassung der MDX Anfrage konnte hier die Ergebnismenge ausreichend verkleinert werden. Mit den Anpassungen des MDX Quelltextes wurden nicht-gewollte, unvollständige Datensätze ausgeschlossen. Dass der Report-Browser eine Anpassung des MDX Quelltextes möglich macht, ist gut: so können Einschränkung des Dimensionsmodells oder des Report-Browsers selbst umgangen werden. Dieses sollte an dieser Stelle jedoch eigentlich nicht nötig sein, denn das händische Anpassen kann von ungeschulten klinischen Benutzer nicht geleistet werden. Im Rahmen der Verifizierung wurde somit deutlich, dass Funktionen zum Ausblenden von unvollständigen Datensätzen oder zum Sortieren der Ergebnismenge in der Anforderungsanalyse nicht aufgedeckt wurden.

Ist eine Ergebnismenge gefunden, kann diese mit dem Report-Browser exportiert werden. Die Dateiformate CSV und XLS wurden realisiert und konnten mit Microsoft Excel korrekt geöffnet werden. Eine Verifizierung der Formate mit Statistiksoftware wie SAS oder SPSS wurde nicht durchgeführt. Die Dokumentation dieser Anwendungen lässt jedoch den Schluss zu, dass hier keine Probleme zu erwarten sind.

Kollaboratives Erstellen und Bearbeiten von Anfragen wurde als weitere Funktion des Report-Browsers definiert. Derzeit unterscheidet die Anwendung zwischen öffentlichen und privaten Anfragen. Öffentliche Anfragen können vom Autoren verändert und von allen Benutzer abgefragt werden. Private Anfragen

stehen nur dem Autor zur Verfügung. Eine Vergabe von komplexeren Schreib- und Leserechten auf gespeicherten Anfragen ist im Report-Browser konzeptionell vorgesehen und auf Datenbankebene bereits umgesetzt.

Grundsätzlich ist der Report-Browser dazu geeignet, von klinischen Benutzer nach einer kurzen Einführung, als Abfragewerkzeug für die RWH Datenbank zu fungieren. Dies konnte in einer ersten User-Evaluation bestätigt werden. Ob Anfragen in einer einfachen Art und Weise gemacht werden können, ist jedoch im großen Maße von den in der OLAP Datenbank definierten Maßzahlen, Dimensionen, Hierarchien und Detailstufen abhängig. Die Möglichkeiten, Anfragen zu speichern und zu öffnen sowie das Exportieren von Ergebnissen, decken die meisten Anforderungen an ein Abfragewerkzeug ab.

6.1.2 Technische Beurteilung

Neben der Untersuchung der Umsetzbarkeit der funktionalen und nichtfunktionale Anforderungen an ein Report-Werkzeug wurde auch die Wahl einer geeigneten Softwarearchitektur als Ziel dieser Arbeit definiert. Die häufig eher philosophisch diskutierte Frage, wann eine Softwarearchitektur eine gute Softwarearchitektur ist, soll an dieser Stelle folgendermaßen beantwortet werden: Eine gute, das heißt geeignete Softwarearchitektur ermöglicht es, alle Anforderungen der Nutzer an die Software zu erfüllen. Sie ist zudem möglichst offen für zukünftige Erweiterungen der Software und bietet den Entwicklern der Software Richtlinien und Strukturen an die Software produktiv entwickeln zu können. Hierbei spielt vor allem eine starke Kohäsion der an der Architektur beteiligten Module und Klassen eine wichtige Rolle. Zudem hat die Architektur großen Einfluss auf die Testbarkeit mittels automatischer Softwaretest.

Der Abschnitt 3.3 (Seite 32) zeigt den grundsätzlichen Aufbau der Architektur des Report-Browsers. Umgesetzt wird die serverseitige Architektur mit Hilfe

von Standardtechnologien. Unter anderem kommen das Spring Framework[46] und Hibernate[47] zum Einsatz.

Den zentralen Anlaufpunkt für den Zugriff auf die Geschäftslogik des Report-Browsers bildet dabei die zweigeteilte Servicefassade: hier können weitere Funktionen leicht hinzugefügt werden. Die Fassade bietet zudem eine einheitliche Schnittstelle, um automatische Integrationstest zu realisieren. Während Zugriffs- und Filterlogik in einem Teil der Fassade realisiert sind, wird davon unabhängig die Geschäftslogik im zweiten Teil der Fassade realisiert. Beide Aspekte können unabhängig von einander und gemeinsam getestet werden.

Hauptaspekt der clientseitigen Architektur ist das MVP Entwurfsmuster. Dieses Muster bietet, im Gegensatz zum weit verbreiteten MVC Ansatz, eine stärkere Kohäsion und ist passender auf die Bedürfnisse eine Webanwendung ausgerichtet. Das MVC Pattern ist von Vorteil, wenn das Observer-Muster [Gam95, S. 293] eingesetzt werden kann: dies ist in HTTP basierten Client-Server-Anwendungen nicht der Fall. Eine klare Aufteilung von Zuständigkeiten und vor allem eine logikfreie Oberfläche (View) bieten eine gute Voraussetzung für eine testbare Architektur. Im Abschnitt 5.1.2 (Seite 63) konnte gezeigt werden, dass sich das Verhalten der Presenter leicht testen lässt. Dies ist der starken Kohäsion der Architektur und dem konsequenten Einsatz von Dependency Injection, also dem expliziten Definieren von Abhängigkeiten einer Klasse mit Hilfe ihres Konstruktors, geschuldet.

Um Client und Server logisch zu trennen, aber im gleichen Moment leicht miteinander zu verbinden, wurde ein modifiziertes Command Muster umgesetzt. Abschnitt 4.2 (Seite 46) zeigt, wie das Entwurfsmuster umgesetzt wurde: entscheidend ist der zweigeteilte Ausführungsschritt. So enthält jedes Command zwei Methoden, die erste wird auf dem Server ausgeführt, um auf die Geschäftslogik zuzugreifen, die zweite auf dem Client, sobald die serverseitige Ausführung abgeschlossen ist. Beispielsweise wird im Command zum Ausführen einer

[46] Spring Framework: `http://www.springsource.org/documentation` (besucht am 01.05.2011)
[47] Hibernate: `http://hibernate.org` (besucht am 01.05.2011).

MDX Anfrage das Ergebnis auf dem Server in einer HTML Tabelle abgebildet. Diese wird auf der Clientseite nurnoch in das DOM integriert. An dieser Stelle kann möglicherweise durch die Wahl eines kleineren Übertragungsformates die Ausführungsgeschwindigkeit von zwischengespeicherten Anfragen erhöht werden. Eine solche Vermutung lassen die Untersuchungen im Abschnitt 5.4 (Seite 71) zu. Hier bietet die Architektur die Möglichkeit, zwischen einer schnelleren Umwandlung des Ergebnisses in Elemente, die im Webbrowser angezeigt werden können, und einer schnelleren Übertragung des Ergebnisses zum Client abzuwägen.

Die Untersuchungen im Abschnitt 5.4 geben erste Hinweise auf die Leistungsfähigkeit der Report-Browser Anwendung. Inwieweit diese Ergebnisse auch für parallele Zugriffe mehrerer Benutzer zutreffen, muss mit Hilfe eines Last- bzw. Stresstests noch untersucht werden. Es wurde in jedem Falle deutlich, dass die Ausführungsgeschwindigkeit durch Zwischenspeicherung drastisch verkürzt werden kann. Dagegen sind die Zeiten zum nicht-zwischengespeicherten Ausführen von Anfragen, abhängig von der Komplexität der Abfrage, aber vor allem von der Größe der Ergebnismenge. Hier können zusätzliche Hierarchien und Detailstufen helfen, die Ergebnismenge klein zu halten.

Die im Report-Browser umgesetzten Abwehrmechanismen gegen bekannte Angriffsszenarien werden im Abschnitt 5.2 (Seite 65) beschrieben. Eine genaue sicherheitstechnische Betrachtung des Report-Browser war jedoch nicht Teil der Zielsetzung dieser Arbeit und wurde daher auch nur oberflächlich durchgeführt. Eine detailliertere Betrachtung und Überprüfung sollte vorgenommen werden. Es sollte darauf geachtet werden, dass das HTTPS Protokoll zur Übertragung von Daten zwischen Client und Server des Report-Browsers und zwischen Server und OLAP Datenbank eingesetzt wird. Eine solche Umstellung hat jedoch möglicherweise Auswirkungen auf die Ausführungsgeschwindigkeiten.

Abschnitt 5.1 beschreibt die Ergebnisse der Verifikation des Report-Browsers mittels automatischer Softwaretests. Die dort gezeigte Testabdeckung der Anwendung (25,2%) lässt keine objektiven Aussagen über die Qualität des Quelltextes zu. Eine hohe Testabdeckung war auch nicht Ziel dieser Machbarkeits-

studie. Es konnte jedoch gezeigt werden, dass sich die Anwendung gut mit Hilfe von automatischen Softwaretests überprüfen lässt.

An dieser Stelle sollen nun noch die Softwarebibliotheken Google Web Toolkit (GWT) und Open Java API for OLAP (olap4j) genauer beleuchtet werden.

Für die grafische Benutzeroberfläche wurde das GWT ausgewählt, da es als Webanwendungsbibliothek große Flexibilität beim Entwickeln einer speziellen Benutzerschnittstelle bietet und mit diesem Framework grundsätzlich sehr interaktive Anwendungen erstellt werden können. Betrachtet man das Look and Feel, also das Aussehen und die Handhabung des Report-Browsers, werden die Vorteile der GWT Bibliothek deutlich. Ein hohes Maß an Interaktivität ist mit klassischen Seiten basierten Ansätze (Java Server Pages) nicht, und mit neueren Bibliotheken (Java Server Faces in der zweiten Version) nur mit größerem Aufwand möglich. Auf Grund der guten Unterstützung durch klassische Java IDEs und dem GWT Entwicklungsmodus (Hosted Mode), lassen sich Anwendungen mit dem GWT zügig entwickeln.

Im Rahmen der Report-Browser Entwicklung wurde GWT in der Version 2.1.1 verwendet, hier konnten einige Fehler entdeckt werden, die bisher verhindert haben, den Report-Browser mit dem Internet Explorer in Version 7 zu benutzen. Keine Fehler konnten in FireFox ab Version 3.5, Google Chrome ab Version 10 und im Internet Explorer 8 festgestellt werden. Derzeit liegt GWT in Version 2.2 vor. Ob sich mit diesem Update der Support des Internet Explorer 7 verbessert hat, wurde bisher nicht untersucht.

Zur Anbindung der OLAP Datenbank wurde im Report-Browser das Open Source Framework olap4j in Version 0.9.8 verwendet. Das Framework lag zum Entwicklungszeitpunkt noch nicht in einer stabilen Version vor, wurde aber vor kurzem in Version 1.0.0 veröffentlicht. Zur Bibliothek olap4j bestehen derzeit keine Alternativen. Der 2000 begonnene Java Specification Request mit der Nummer 69, Java OLAP Interface (JOLAP) wurde nicht beendet und ist derzeit inaktiv. Eine spezielle Java-Schnittstelle existiert zum OLAP Server der Firma Oracle, jedoch nicht zum Server von Microsoft. Eine Eigenentwicklung,

aufsetzend auf die XMLA SOAP Schnittstelle des OLAP Servers, wurde nicht verfolgt.

olap4j erfüllt derzeit alle Anforderungen des Report-Browsers. Die Bibliothek ist als Erweiterung der JDBC Schnittstellen entwickelt worden, dies hat zur Folge, dass viele der Schnittstellen nicht implementierte Funktionen enthalten, die nur bei relationale Datenbanken Verwendung finden. Durch die Erweiterung des JDBC APIs lässt sich olap4j jedoch in Connection Pooling Frameworks wie, c3p0[48] und DBCP[49] integrieren. Im Report-Browser findet c3p0 Verwendung.

Nach einem Neustart des Report-Browser Servers kann ein Fehler im olap4j Framework (0.9.8) beobachtet werden; hier wird beim Absetzen der ersten Anfrage eine „RecursivPopulationException“ geworfen. Alle weiteren Abfragen funktionieren ohne Probleme. Ob dieses Problem mit olap4j in Version 1.0.0 noch besteht, wurde bisher nicht untersucht.

Es bleibt festzuhalten, dass sich die Architektur des Report-Browsers durch eine gute Erweiterbarkeit und einfache Testbarkeit auszeichnet, sie ist daher grundsätzlich geeignet, um mit ihr ein Abfragewerkzeug für OLAP Datenbanken zu erstellen.

6.2 Übergang in den Routinebetrieb

Um den Report-Browser in den Routinebetrieb übergeben zu können, müssen eine Reihe von Maßnahmen und Untersuchungen noch durchgeführt werden. Einige andere können auf einen späteren Zeitpunk verschoben und dann als Update des Report-Browsers veröffentlicht werden.

Vor der Freigabe des Report-Browser an die Benutzer der Medizinischen Universitätsklinik sollte das Benutzerverzeichnis (AD) des Uniklinikums Heidel-

[48] c3p0: `http://sourceforge.net/projects/c3p0` (besucht am 01.05.2011).
[49] DBCP: `http://jakarta.apache.org/commons/dbcp` (besucht am 01.05.2011).

berg in den Autorisierungs- und Authentifizierungsprozess des Rerport-Browsers eingebunden werden. Durch die bereits erfolgte Einbindung des Spring Security Frameworks sollte dies keine größeren technischen Probleme breiten.[50] Eine Abstimmung mit den Administratoren des AD ist jedoch Voraussetzung.

Der zu Testzwecken in Betrieb genommene Report-Browser ist derzeit nur über das nicht verschlüsselte Protokoll HTTP verfügbar. Bevor die Anwendung in den Routinebetrieb übergeben wird, sollte hier auf das verschlüsselte Protokoll HTTPS umgestellt werden. Dazu muss ein geeignetes Zertifikat auf dem Server installiert werden. Hierfür sind ebenso Abstimmungen mit den Administratoren des Klinikums nötig.

Um die Angriffsfläche für mögliche Injection-Angriffe (vergl. 3.6.1, Seite 38) der OLAP Datenbank zu verkleinern, sollte evaluiert werden, ob die Funktionen zum manuellen Anpassen der MDX Anfragen allen Benutzern und nicht zum Beispiel nur Administratoren zur Verfügung stehen sollten.

Derzeit wird im Uniklinikum Heidelberg der Internet Explorer in Version 7 standardmäßig eingesetzt. In diesem Webbrowser funktioniert die Report-Browser Anwendung nicht. Es muss also untersucht werden, ob dies mit GWT in Version 2.2 noch der Fall ist, Änderungen am Report-Browser selbst die Probleme lösen können oder die Aktualisierung des Internet Explorers im Uniklinikum auf Version 8 abgewartet werden kann.

Im Abschnitt 6.1.1 (Seite 76) sind zwei fehlende Funktionen beschrieben, welche im Rahmen der Verifikation der Report-Browsers aufgedeckt wurden. Es ist zu überlegen, diese Funktionen, das Ausblenden von unvollständigen Datensätzen und das Sortieren von Datensätzen, vor der Übergabe des Report-

[50] Der Verzeichnisdienst Active Directory von Microsoft stellt eine LDAP Schnittstelle zur Verfügung. Informationen zum Einbinden eines LDAP Benutzerverzeichnis in das Spring Security Framework unter `http://static.springsource.org/spring-security/site/docs/2.0.x/reference/ldap.html` (besucht am: 01.05.2001).

Browsers in den Routinebetrieb noch zu realisieren oder später als Update zur Verfügung zu stellen.

Durch die Verifikation des Report-Browsers konnte zudem festgestellt werden, dass die Benutzerfreundlichkeit beim Erstellen von Abfragen stark von den definierten Maßzahlen, Dimensionen, Hierarchien und Detailstufen abhängt. Es ist zu empfehlen, hier möglichst gut geeignete Datenstrukturen zur Verfügung zustellen. Wird der Datenwürfel des OLAP Servers aktualisiert, ist zu beachten, dass derzeit die Benutzerrolle für den Zugriff auf den Datenwürfel nach einer Aktualisierung manuell eingerichtet werden muss. Da die Aktualisierung des Datenwürfels in Zukunft nachts automatisch stattfinden soll, muss ein Weg gefunden werden, den Zugriff durch den Report-Browser nicht zu blockieren.

6.3 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein Werkzeug zum Durchsuchen von großen Datenmengen mittels mutidimensionaler Anfragen an eine Online Analytical Processing Datenbank erstellt. Dieses Werkzeug macht klinische Routinedaten und medizinische Forschungsdaten zugänglich. In dieser Arbeit wurden in einer Analyse funktionale und nicht-funktionale Anforderungen an ein solches Werkzeug beschrieben. Zum Bewältigen dieser Anforderungen wurde eine Softwarearchitektur erstellt und anhand dieser der Report-Browser, unter der Nutzung von modernen Softwaretechnologien, umgesetzt. Die weitestgehende Erfüllung der Anforderungen an den Report-Browser konnte mit Hilfe zweiter Verifikationsszenarien sichergestellt werden.

Die Qualität der Ergebnisse, die dieses Werkzeug liefern kann, bleibt, wie bei jedem anderen Datenverarbeitungswerkzeug auch, abhängig von der Qualität der zu verarbeitenden Daten. Hier müssen Wege gefunden werden, eine hohe Datenqualität von der Erfassung bis zur Auswertung zu garantieren. Nur wenn der Richtigkeit der Daten vertraut werden kann, sind diese zum Beantworten von wissenschaftlichen Fragestellungen geeignet. Werkzeuge zur automatischen

Überprüfung der Datenqualität sowie Möglichkeiten zur besseren Betrachtung von Ergebnismengen, zum Beispiel mit Hilfe von Pivot-Tabellen, können in Zukunft im Report-Browser umgesetzt werden.

Der Report-Browser und die zugehörige Architektur bieten das Potenzial, zu einer einheitlichen Zugriffsplattform für das Research Warehouse Projekt zu werden. Eine Integration des Pseudonymisierungsdienstes in die grafische Benutzeroberfläche des Report-Browsers wird angestrebt und dürfte, aus technologischer Sicht, keine Probleme bereiten.

So entsteht ein Werkzeug, das sowohl die medizinische Primärforschung als auch die Versorgungsforschung beim Umgang mit klinischen Routinedaten unterstützt.

Literatur

[Abd09] Ahsan Abdullah. „Analysis of mealybug incidence on the cotton crop using ADSS-OLAP (Online Analytical Processing) tool“. In: *Computers and Electronics in Agriculture* 69.1 (2009), S. 59–72. ISSN: 0168-1699.

[Arc09] Secure Arc. *Reverse Proxy Pattern*. 2009. URL: http://www.securearc.com/wiki/index.php?title=Reverse_Proxy_Pattern&oldid=1051 (besucht am 01. 05. 2011).

[AT] Ben Alex und Luke Taylor. *Spring Security - Reference Documentation*. URL: http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.pdf (besucht am 01. 05. 2011).

[BMBF10] Bundesministerium für Bildung und Forschung (BMBF). *Verbundprojekt KIS-basierte Unterstützung der Patientenrekrutierung in klinischen Studien*. 2010. URL: http://www.gesundheitsforschung-bmbf.de/de/2127.php#KIS (besucht am 01. 05. 2011).

[BSS00] Wolfgang Bartel, Stefan Schwarz und Gerhard Strasser. „Der ETL-Prozess des Data Warehousing“. In: *Data Warehousing Strategie*. Hrsg. von Reinhard Jung und Robert Winter. 2000, S. 43–60. ISBN: 3540673083. URL: http://books.google.de/books?id=g03a2leQSOsC&pg=PA43 (besucht am 01. 05. 2011).

[CSL04] Sanjay Chaudhary, Vikram Sorathia und Zakir Laliwala. „Architecture of sensor based agricultural information system for effective planning of farm activities“. In: *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on*. IEEE. 2004, S. 93–100. ISBN: 0769522254.

[Dan10] Daniel Danilatos. „GWT testing best practices“. In: *Google IO Developer Conference*. Google Inc. 2010. URL: http://www.google.com/events/io/2010/sessions/gwt-continuous-build-testing.html (besucht am 01. 05. 2011).

[Dug08] Martin Dugas et al. „Workflow to improve patient recruitment for clinical trials within hospital information systems - a case-study“. In: *Trials* 9.1 (2008), S. 2. ISSN: 1745-6215. DOI: 10.1186/1745-6215-9-2. URL: http://www.trialsjournal.com/content/9/1/2 (besucht am 01. 05. 2011).

[Fow02] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley, 2002. ISBN: 0321127420.

[Gam95] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. ISBN: 0201633612.

[Gan10] Thomas Ganslandt et al. „Single-Source-Projekte am Universitätsklinikum Erlangen - Beispielprojekte zur Nutzung von Daten aus der elektronischen Krankenakte für die klinische und translationale Tumorforschung“. In: *Forum der Medizin Dokumentation und Medizin Informatik (MDI)* 2010.2 (2010), S. 62–66.

[Gar05] Jesse James Garrett et al. *Ajax A new approach to web applications*. 2005. URL: http://www.adaptivepath.com/ideas/essays/archives/000385.php (besucht am 01. 05. 2011).

[Gro05] Thomas G. Grobe. „Stationäre Versorgung - Krankenhausbehandlungen“. In: *Routinedaten im Gesundheitswesen, Handbuch Sekundäranalyse Grundlagen, Methoden und Perspektiven*. Hrsg. von Enno Swart und Peter Ihle. Huber, 2005, S. 79–98. ISBN: 3456842376.

[GU10] Daniel Guermeur und Amy Unruh. *Google App Engine Java and GWT Application Development*. Packt Publishing, 2010. ISBN: 9781849690447.

[GwtMvp] Google Web Toolkit. *GWT MVP Development with Activities and Places*. URL: `http://code.google.com/webtoolkit/doc/latest/DevGuideMvpActivitiesAndPlaces.html` (besucht am 01. 05. 2011).

[GwtRpc] Google Web Toolkit. *Making Remote Procedure Calls*. URL: `http://code.google.com/webtoolkit/doc/latest/tutorial/RPC.html` (besucht am 01. 05. 2011).

[HK11] Julian Hyde und Barry Klawans. *olap4j Specification*. 2011. URL: `http://www.olap4j.org/olap4j_fs.html` (besucht am 01. 05. 2011).

[HRM00] Dimitar Hristovski, Mitja Rogac und Mladen Markot. „Using data warehousing and OLAP in public health care.“ In: *Proceedings of the AMIA Symposium*. American Medical Informatics Association. 2000, S. 369–373.

[Hu07] Cheng Lin Hu et al. „Troponin T measurement can predict persistent left ventricular dysfunction in peripartum cardiomyopathy“. In: *Heart* 93.4 (2007), S. 488–490. DOI: `10.1136/hrt.2006.087387`. URL: `http://heart.bmj.com/content/93/4/488.abstract` (besucht am 01. 05. 2011).

[Ihl05] Peter Ihle. „Datenschutzrechtliche Aspekte bei der Erhebung von GKV-Routinedaten“. In: *Routinedaten im Gesundheitswesen, Handbuch Sekundäranalyse Grundlagen, Methoden und Perspektiven*. Hrsg. von Enno Swart und Peter Ihle. Huber, 2005, S. 195–201. ISBN: 3456842376.

[JF10] Philo Janus und Guy Fouché. *Pro SQL Server 2008 Analysis Services*. Apress, 2010. ISBN: 1430219955.

[Joh09] Bruce Johnson. „GWT Can Do What?!?! A Preview of Google Web Toolkit 2.0“. In: *Google IO Developer Conference*. Google Inc. 2009. URL: http://www.google.com/events/io/2009/sessions/GwtPreviewGoogleWebToolkit2.html (besucht am 01. 05. 2011).

[Kie10] Meinhard Kieser. *Vorlesung Medizinische Biometrie und Epidemiologie 1*. Studiengang Medizinische Informatik Heidelberg / Heilbronn, 2010.

[Köb98] Johannes Köbberling. „Der Begriff der Wissenschaft in der Medizin“. In: Arbeitsgemeinschaft der Wissenschaftlichen Medizinischen Fachgesellschaften. 1998. URL: http://www.awmf.org/fileadmin/user_upload/Die_AWMF/Service/Gesamtarchiv/AWMF - Konferenz/Der_Begriff_der_Wissenschaft_in_der_Medizin.pdf (besucht am 01. 05. 2011).

[Kur11] Kerstin Kurz et al. „Comparison of the new high sensitive cardiac troponin T with myoglobin, h-FABP and cTnT for early identification of myocardial necrosis in the acute coronary syndrome“. In: *Clinical Research in Cardiology* (2011), S. 209–215. DOI: 10.1007/s00392-010-0230-y.

[Lei08] Tom Leighton. „Improving Performance on the Internet“. In: *Queue* 6 (6 2008), S. 20–29. ISSN: 1542-7730. DOI: 10.1145/1466443.1466449. URL: http://doi.acm.org/10.1145/1466443.1466449 (besucht am 01. 05. 2011).

[MSDN11] Microsoft Developer Network (MSDN). *Specifying the Contents of a Query Axis (MDX)*. 2011. URL: http://msdn.microsoft.com/en-us/library/ms146052.aspx (besucht am 01. 05. 2011).

[Mur07] Shawn N. Murphy et al. „Architecture of the open-source clinical research chart from Informatics for Integrating Biology and the Bedside“. In: *AMIA Annual Symposium Proceedings*. American Medical Informatics Association. 2007, S. 548–552.

[Mur09] Shawn N. Murphy. *Enabling the Healthcare Enterprise for Discovery Research with i2b2*. 2009. URL: http://btris.nih.gov/presentations/Shawn_Murphy_presentation.pdf (besucht am 01. 05. 2011).

[Mur10] Shawn N. Murphy et al. „Serving the enterprise and beyond with informatics for integrating biology and the bedside (i2b2)“. In: *Journal of the American Medical Informatics Association* 17.2 (2010), S. 124. ISSN: 1527-974X.

[MW02] Rainer Metschke und Rita Wellbrock. *Datenschutz in Wissenschaft und Forschung*. Berliner Beauftragter für Datenschutz und Informationsfreiheit, 2002. URL: http://www.datenschutz-berlin.de/attachments/47/Materialien28.pdf (besucht am 01. 05. 2011).

[Nus10] Brigitte Nussbaum. *Jede dritte Studie scheitert mangels Probanden*. Presse- und Informationsstelle Westfaelische Wilhelms-Universität Münster. 2010. URL: http://idw-online.de/de/news380049 (besucht am 01. 05. 2011).

[OK07] Christian Ohmann und Wolfgang Kuchinke. „Meeting the challenges of patient recruitment: A role for electronic health records“. In: *International Journal of Pharmaceutical Medicine* 21.4 (2007), S. 263–270. ISSN: 1364-9027.

[OWASP10] Dave Wichers et al. *OWASP Top 10 Application Security Risks - 2010*. Open Web Application Security Project (OWASP). 2010. URL: http://www.owasp.org/index.php?title=Top_10_2010-Main&oldid=82438 (besucht am 01. 05. 2011).

[PDS10] Hans-Ulrich Prokosch, Martin Dugas, Sebastian Claudius Semler et al. *Projektgruppe Nutzung von Elektronischen Krankenakten für die klinische Forschung*. 2010. URL: http://www.pg-ss.imi.uni-erlangen.de (besucht am 01. 05. 2011).

[Pfa03] Holger Pfaff. „Versorgungsforschung–Begriffsbestimmung, Gegenstand und Aufgaben“. In: *Gesundheitsversorgung und Disease Management–Grundlagen und Anwendungen der Versorgungsforschung, Bern, S* (2003), S. 13–23.

[PG09] Hans-Ulrich Prokosch und Thomas Ganslandt. „Perspectives for Medical Informatics: Reusing the Electronic Medical Record for Clinical Research“. In: *Methods Inf Med* 48.1 (2009), S. 38–44. DOI: 10.3414/ME9132.

[Pom] Klaus Pommerening et al. *Das TMF-Datenschutzkonzept für medizinische Datensammlungen und Biobanken*. URL: http://www.staff.uni - mainz.de/pommeren/Artikel/05_pommerening_sax_mueller_speer_ganslandt_drepper.pdf (besucht am 01. 05. 2011).

[Pot96] Mike Potel. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java*. Taligent Inc. 1996. URL: http://www.wildcrest.com/Potel/Portfolio/mvp.pdf (besucht am 01. 05. 2011).

[Pro10] Hans-Ulrich Prokosch. „Single-Source-Aktivitäten in Deutschland“. In: *Forum der Medizin Dokumentation und Medizin Informatik (MDI)* 2010.2 (2010), S. 56–57.

[Reb09] Herbert Rebscher. „Warum brauchen wir Versorgungsforschung?“ In: *Gesundheitssysteme im Wandel*. Hrsg. von Herbert Rebscher und Stefan Kaufmann. Economica Verlag, 2009, S. 119–136. ISBN: 3870817704. URL: http://books.google.de/books?id=ZCFCCeKG5sYC&pg=PA119 (besucht am 01. 05. 2011).

[Röh09] Bernd Röhrig et al. „Studientypen in der medizinischen Forschung“. In: *Deutsches Ärzteblatt* 106 (2009), S. 262–268.

[Rya09] Ray Ryan. „Google Web Toolkit Architecture: Best Practices For Architecting Your GWT App“. In: *Google IO Developer Conference*. Google Inc. 2009. URL: http://www.google.com/events/io/2009/sessions/GoogleWebToolkitBestPractices.html (besucht am 01. 05. 2011).

[Sac96] David L. Sackett et al. „Evidence based medicine what it is and what it isn't“. In: *Bmj* 312.7023 (1996), S. 71. ISSN: 0959-8138.

[Sch07] Norbert Schmacke. „Versorgungsforschung Hoffnungsträger oder Modernismus?“ In: *G+G Wissenschaft (GGW)* 1/2007 (2007), S. 7–13.

[SDN01] Sun Developer Network (SDN). *Core J2EE Patterns - Data Access Object*. 2001. URL: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html (besucht am 01. 05. 2011).

[SI05] Enno Swart und Peter Ihle, Hrsg. *Routinedaten im Gesundheitswesen*. Huber, 2005. ISBN: 3456842376.

[SJB97] Andrew Sears, Julie A. Jacko und Michael S. Borella. „Internet delay effects: how users perceive quality, organization, and ease of use of information“. In: *CHI'97 extended abstracts on Human factors in computing systems: looking to the future*. ACM. 1997, S. 353–354. ISBN: 0897919262.

[Smi09] Josh Smith. „WPF Apps With The Model-View-ViewModel Design Pattern“. In: *MSDN Magazine* (2009).

[Ste07] Henning Steen et al. „Relative role of NT-pro BNP and cardiac troponin T at 96 hours for estimation of infarct size and left ventricular function after acute myocardial infarction“. In: *Journal of Cardiovascular Magnetic Resonance* 9.5 (2007), S. 749–758. ISSN: 1097-6647.

[Wiki11a] Wikipedia. *Cross-Site Scripting — Wikipedia, Die freie Enzyklopädie*. 2011. URL: `http://de.wikipedia.org/w/index.php?title=Cross-Site_Scripting&oldid=86887657` (besucht am 01. 05. 2011).

[Wiki11b] Wikipedia. *Online analytical processing — Wikipedia, The Free Encyclopedia*. 2011. URL: `http://en.wikipedia.org/w/index.php?title=Online_analytical_processing&oldid=409799010` (besucht am 01. 05. 2011).

[Win98] Alfred Winter et al. „Das Management von Krankenhausinformationssystemen: Eine Begriffsdefinition“. In: *Informatik, Biometrie und Epidemiologie in Medizin und Biologie* 29.2 (1998), S. 93–105.

[Wu06] Cai Wu. „Development of A Medical Informatics Data Warehouse“. In: *AMIA Annual Symposium Proceedings*. American Medical Informatics Association. 2006, S. 1148.

[XMLA] *XML for Analysis council web site*. URL: `http://www.xmla.org` (besucht am 01. 05. 2011).

Anhänge

A.1 GWT Hello World Listings

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='de_vogella_gwt_helloworld'>
  <inherits name='com.google.gwt.user.User'/>

  <entry-point class='de.rwh.test.HelloWorld'/>
</module>
```

Listing 8: Modul Konfigurationsdatei

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
    charset=UTF-8">
    <title>Hello World</title>
    <script type="text/javascript" language="javascript"
    src="de.rwh.test/de.rwh.test.HelloWorld.nocache.js"></script>
  </head>
  <body>
  </body>
</html>
```

Listing 9: Modul Startseite

A.2 Bildschirmdrucke Report-Browser

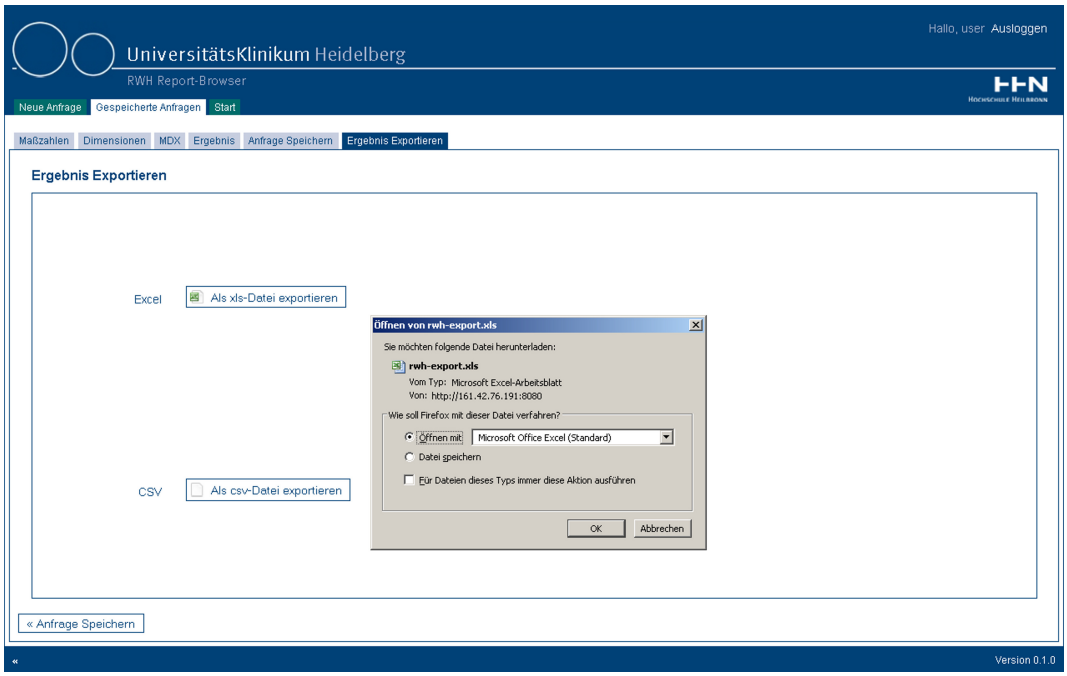


Abbildung 36: Report-Browser: Dialog zum Abspeichern eines exportierten Ergebnisses

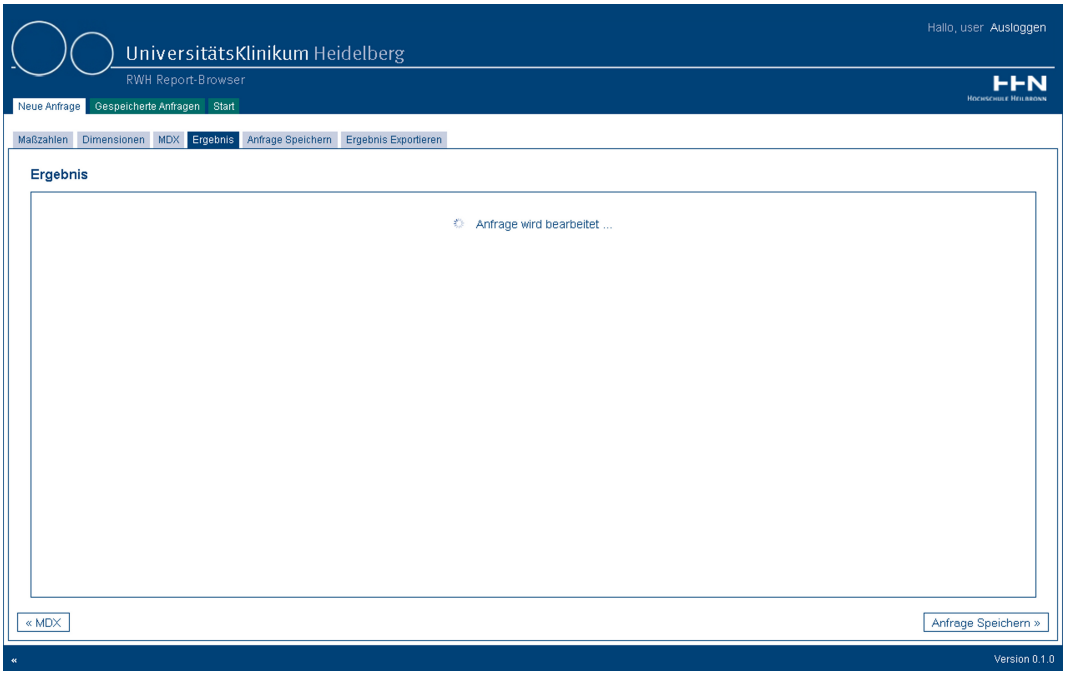


Abbildung 37: Report-Browser: Ergebnisberechnung

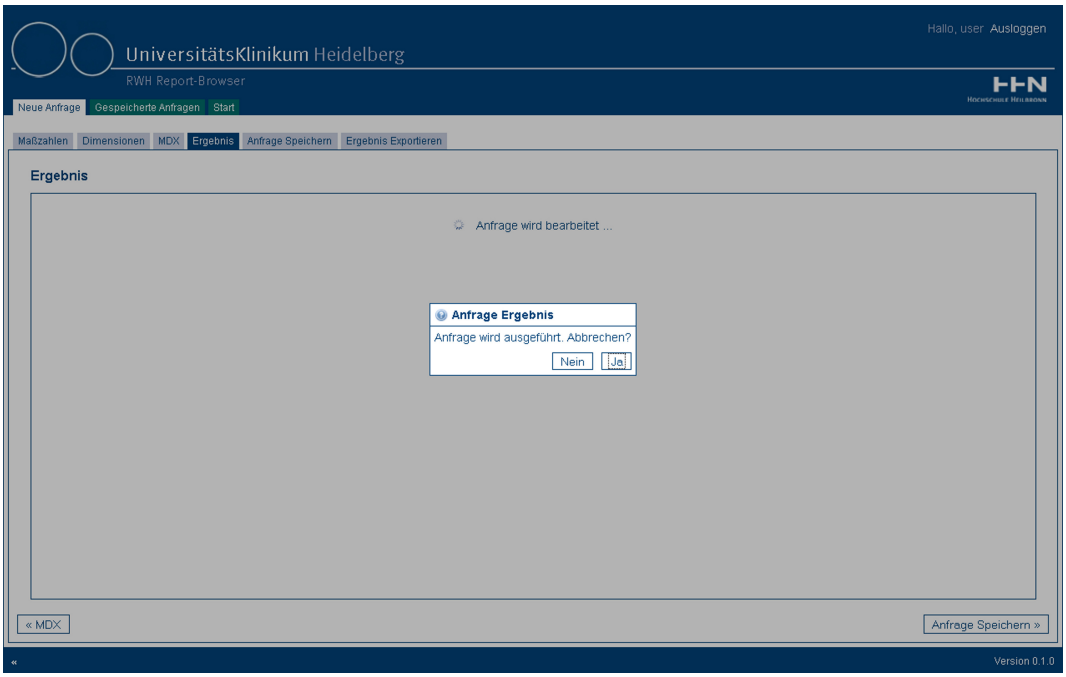


Abbildung 38: Report-Browser: Ergebnisberechnung nach 3 Sekunden

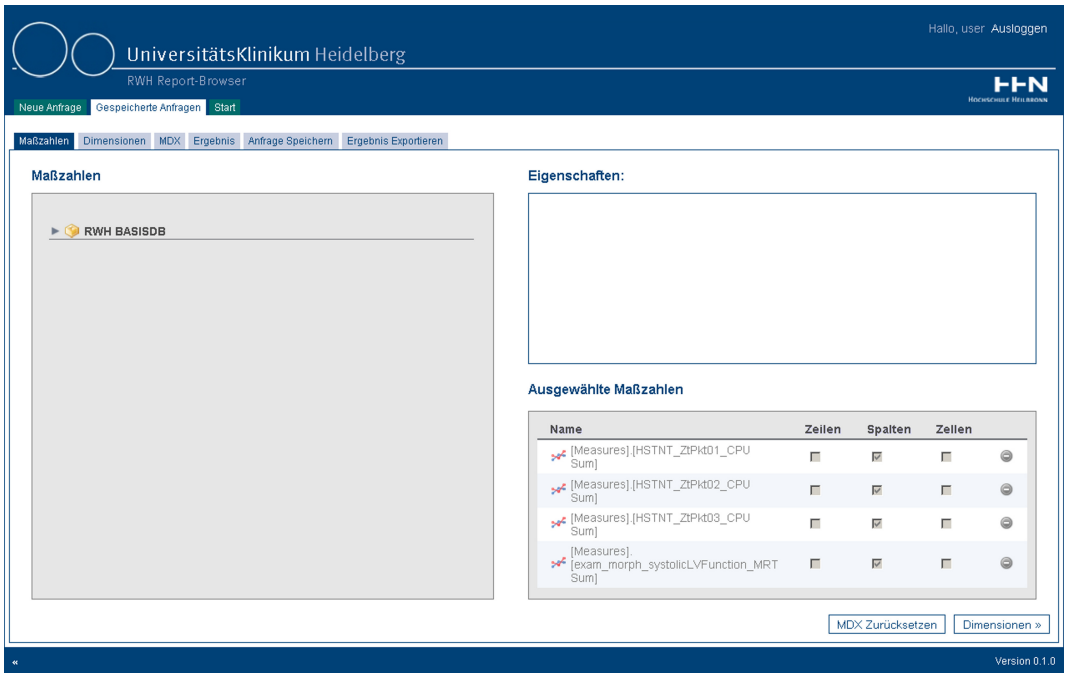


Abbildung 39: Report-Browser: Maßzahlen, bei manuell verändertem MDX

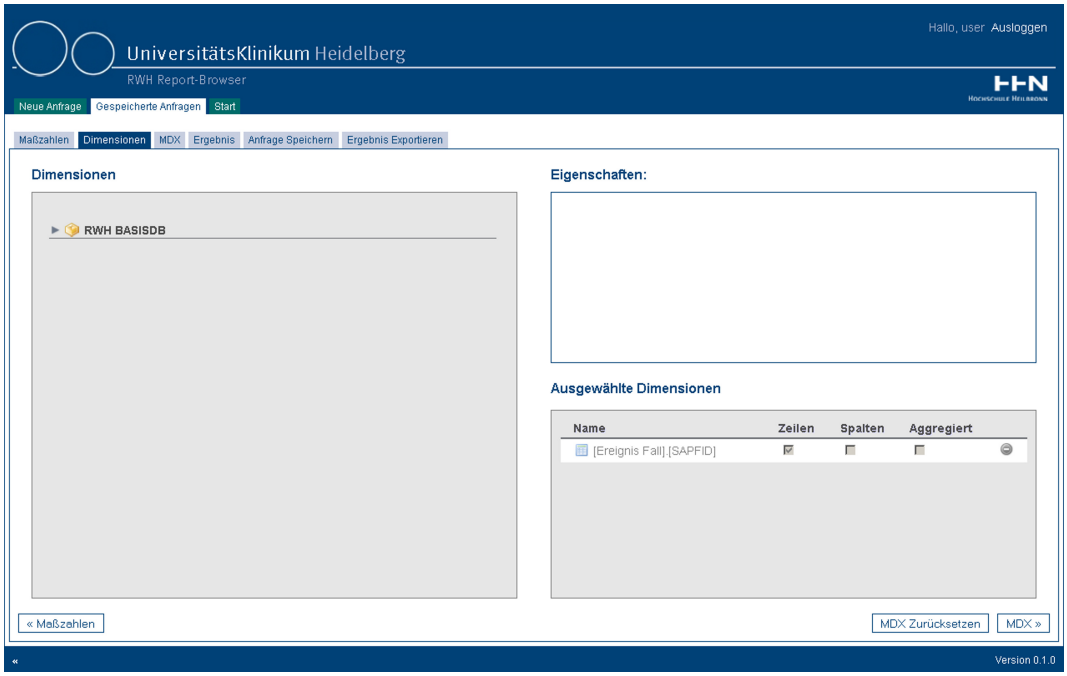


Abbildung 40: Report-Browser: Dimensionen, bei manuell verändertem MDX

A.3 Ausschnitt JUnit Test: LoginActivity

```
1  package de.rwh.browser.client.activities;
2
3  import static org.easymock.EasyMock.*;
4  import static org.junit.Assert.*;
5
6  import org.easymock.Capture;
7  import org.junit.Before;
8  import org.junit.Test;
9  import org.springframework.test.util.ReflectionTestUtils;
10
11 import com.google.gwt.event.shared.EventBus;
12
13 import de.rwh.browser.client.common.FooterMessage;
14 import de.rwh.browser.client.common.FooterMessage.Type;
15 import de.rwh.browser.client.common.WindowFunctionWrapper;
16 import de.rwh.browser.client.events.ShowFooterMessage;
17 import de.rwh.browser.client.places.Login;
18 import de.rwh.browser.client.presenter.DialogPresenter;
19 import de.rwh.browser.client.presenter.MainPresenter;
20 import de.rwh.browser.client.resources.txt.DialogMessages;
21 import de.rwh.browser.client.resources.txt.FooterMessages;
22 import de.rwh.browser.client.views.login.LoginView;
23 import de.rwh.browser.shared.common.ClientFactory;
24
25 public class LoginActivityTest
26 {
27   private static final String UNAME_AND_PASSWORD_INVALID =
28     "UsernameAndPasswordInvalid";
29
30   private Login place;
31   private ClientFactory clientFactoryMock;
32   private EventBus eventBusMock;
33   private FooterMessages footerMessagesMock;
34   private DialogPresenter dialogPresenterMock;
35   private DialogMessages dialogMessagesMock;
```

```
36  private MainPresenter mainPresenterMock;
37  private LoginView loginViewMock;
38  private WindowFunctionWrapper windowMock;
39
40  @Before
41  public void before() {
42    place = new Login();
43
44    clientFactoryMock = createMock(ClientFactory.class);
45    eventBusMock = createMockBuilder(EventBus.class).createMock();
46    footerMessagesMock = createMock(FooterMessages.class);
47    dialogPresenterMock =
48      createMockBuilder(DialogPresenter.class).createMock();
49    dialogMessagesMock = createMock(DialogMessages.class);
50    mainPresenterMock = createMockBuilder(MainPresenter.class).
51      createMock();
52    loginViewMock = createMock(LoginView.class);
53    windowMock = createMock(WindowFunctionWrapper.class);
54    expect(clientFactory.getWindowFunctionWrapper()).
55      andReturn(windowMock);
56    expect(clientFactory.getEventBus()).andReturn(eventBusMock);
57    expect(clientFactory.getFooterMessages()).
58      andReturn(footerMessagesMock);
59    expect(clientFactory.getDialogPresenter()).
60      andReturn(dialogPresenterMock);
61    expect(clientFactory.getDialogMessages()).
62      andReturn(dialogMessagesMock);
63    expect(clientFactory.getMainPresenter()).
64      andReturn(mainPresenterMock);
65    expect(clientFactory.getLoginView()).andReturn(loginViewMock);
66  }
67
68  private void replayAll() {
69    replay(clientFactoryMock, eventBusMock, footerMessagesMock,
70      dialogPresenterMock, dialogMessagesMock, mainPresenterMock,
71      loginViewMock);
72  }
73
```

```
74    private void verifyAll() {
75      verify(clientFactoryMock, eventBusMock, footerMessagesMock,
76        dialogPresenterMock, dialogMessagesMock, mainPresenterMock,
77        loginViewMock);
78    }
79
80    @Test
81    public void testOnSubmitButtonClickedUnameAndPasswordInvalid() {
82      // record behavior of mocked classes
83      expect(loginViewMock.getUsername()).andReturn("");
84      expect(loginViewMock.getPassword()).andReturn("");
85
86      loginViewMock.setUsernameInvalid(true);
87      expectLastCall().once();
88      loginViewMock.setPasswordInvalid(true);
89      expectLastCall().once();
90
91      loginViewMock.setPasswordFocus(false);
92      expectLastCall().once();
93      loginViewMock.setUsernameFocus(true);
94      expectLastCall().once();
95
96      expect(footerMessagesMock.usernameAndPasswordInvalid()).
97        andReturn(UNAME_AND_PASSWORD_INVALID);
98
99      Capture<ShowFooterMessage> capturedMessage =
100       new Capture<ShowFooterMessage>();
101     Capture<LoginActivity> capturedActivity =
102       new Capture<LoginActivity>();
103
104     eventBusMock.fireEventFromSource(capture(capturedMessage),
105       capture(capturedActivity));
106     expectLastCall().once();
107
108     replayAll(); //replay behavior of mocked classes
109
110     LoginActivity loginActivity =
111       new LoginActivity(place, clientFactoryMock);
```

```
112     loginActivity.onSubmitButtonClicked(); //call tested method
113
114     FooterMessage message = (FooterMessage) ReflectionTestUtils.
115       getField(capturedMessage.getValue(), "message");
116
117     assertEquals(UNAME_AND_PASSWORD_INVALID, message.getText());
118     assertEquals(Type.Warning, message.getType());
119
120     assertEquals(loginActivity, capturedActivity.getValue());
121
122     assertTrue((Boolean) ReflectionTestUtils.
123       getField(loginActivity, "usernameInvalid"));
124     assertTrue((Boolean) ReflectionTestUtils.
125       getField(loginActivity, "passwordInvalid"));
126
127     verifyAll(); //verify behavior of mocked classes
128   }
129 }
```

Listing 10: Ausschnitt aus dem JUnit Test für den Presenter: LoginActivity

Eidesstattliche Erklärung

Titel:	Implementierung und Verifikation eines Report-Browsers für Multidimensionale Datenbanken im Klinikumfeld
Autor:	Hauke Hund
Matrikelnummer:	164778

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen (einschließlich elektronischer Quellen) direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich Unterstützungsleistung von folgenden Personen erhalten:

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und ist auch noch nicht veröffentlicht.

Ort, Datum Unterschrift