technische universität
dortmund

Prozessorientierte Verarbeitung von biomedizinischen Daten

- Endbericht -

Projektgruppe ProBio

Patrick Brandt, Leonid Moor, Sebastian Opriel, Carsten Schulte, Fabian Schulte, Heiko Schwedhelm, Adrian Skirzynski, Andre Thevapalan, Marc Walocha, Jan Winkels und Matthias Wisniewski

19. September 2016

Betreuer:
Dipl.-Inf. Stefan Naujokat
Dr. Johannes Neubauer
Prof. Dr. Bernhard Steffen

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Programmiersysteme (LS5)
`http://ls5-www.cs.uni-dortmund.de`

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieser Enbericht fasst die Ergebnisse der Projektgruppe (PG) 586 „Prozessorientierte Verarbeitung von biomedizinischen Daten“ (ProBio) zusammen, die im Wintersemester 2014/2015 und im Sommersemester 2015 an der TU Dortmund stattgefunden hat. Dieser Bericht zeigt eine systematische Darstellung der bearbeiteten Problemstellungen auf, legt den dazu eingeschlagenen Lösungsweg dar und umfasst die Zusammenfassung der Diskussionsergebnisse.

1.1 Ablauf

Die Projektgruppen an der TU Dortmund erstrecken sich an der Fakultät für Informatik über zwei Semester. In dieser Zeit sollen die studentischen Teilnehmer der PG möglichst selbstständig das vorgegebene Ziel erreichen. Die Betreuer leiten die Gruppe an und stehen bei Fragen und Problemen der Gruppe zur Seite. Die Projektgruppen umfassen einen Arbeitsaufwand im Umfang von 8 Semesterwochenstunden.

1.2 Ziel der Projektgruppe

Das Ziel der PG ist die Entwicklung einer prozessorientierten Webanwendung in Zusammenarbeit mit Biomedizinern aus São Paulo in Brasilien. Die Anwendung soll es möglich machen, die bei den Biomedizinern anfallenden Daten zentral zu erfassen und (mit Hilfe von Prozessen) zu verarbeiten.

Ausgangspunkt der Webanwendung ist die Domänenmodellierung von biomedizinischen Daten und die Formalisierung von Prozessen, die zugehörige Arbeitsabläufe abbilden, in Form von Prozessmodellen. Die Domänen- und die Prozessmodellierung erfolgt auf Basis der Nutzung der Werkzeuge DyWA und jABC. Neben der Nutzung der Werkzeuge ist auch die Erweiterung dieser Werkzeuge nötig, um die spezifischen Anforderungen der Anwendungsdomäne der Biomedizin erfüllen zu können. Darunter fallen erweiterte Suchfunktionen, die Einbindung von Zugriffskontrollmechanismen, die Möglichkeit der Modellierung einer domänenspezifischen Weboberfläche und die Verwendung erweiterter Funktionen wie Benutzerinteraktionen in Prozessmodellen.

Am Ende soll es dem Domänenexperten möglich sein, mit Hilfe der von der PG ProBio erstellten Werkzeuge, in Verbindung mit den bereits bestehenden Werkzeugen (DyWA und jABC), selber eine Webapplikation bauen zu können, welche speziell auf die Anforderungen und Bedürfnisse der späteren Anwender zugeschnitten ist.

1.3 Teilnehmer

Die Betreuer der PG sind Dr. Johannes Neubauer, Dipl.-Inf. Stefan Naujokat und Dipl.-Inf. Oliver Bauer. Oliver Bauer betreute die PG bis zur zweiten Hälfte, ab der zweiten Hälfte übernahm die Betreuung Stefan Naujokat. Die teilnehmenden Studenten der PG sind Patrik Brandt, Leonid Moor, Sebastian Opriel, Carsten Schulte, Fabian Schulte, Heiko Schwedhelm, Adrian Skirzynski, Andre Thevapalan, Marc Walocha, Jan Winkels und Matthias Wisniewski.

1.4 Organisation

Die PG hat sich dazu entschieden, die 12 Teilnehmer auf vier kleinere Untergruppen aufzuteilen. Dadurch soll eine effektive Bündelung der Kräfte auf den jeweiligen Teilaufgabenbereich erreicht werden. In der Untergruppe „DyWA GUI“ kümmern sich Patrick, Jan, Sebastian und Matthias um eine Möglichkeit zur Modellierung der Oberfläche der domänenspezifischen Anwendung. Die zweite Untergruppe bilden Marc, Yannick und Heiko in der Gruppe „DyWA Logic Builder“. Die Gruppe „Logic“ implementiert zum Beispiel Timer für die Prozessausführung, sowie zusätzliche SIB-Bausteine um die Funktionen der jABC-Prozesse zu erweitern. Carsten, Leonid und Fabian bilden zusammen die Gruppe „DyWA Security“, welche sich um die Authentifizierung von Nutzern und die Autorisierung hinsichtlich der Prozesse kümmern. „DyWA Search“ mit Andre und Adrian bildet die letzte Teilgruppe. Diese Gruppe kümmert sich um eine Suchfunktion für domänenspezifische Anwendungen, welche unabhängig von der zugrunde liegenden Datenbank ist.

Eine Mitarbeit über die Grenzen dieser Aufteilung in kleinere Untergruppen war jederzeit erwünscht und es wurde durchaus über die Grenzen der Untergruppen hinaus einander ausgeholfen.

Wöchentlich wird eine gemeinsame Teamsitzung mit allen Teilnehmern und den Betreuern abgehalten, in welcher die Probleme und Fortschritte der einzelnen Teilgruppen besprochen werden. Die Protokolle der Sitzungen finden sich im Projekt-Wiki der PG.

Die Projektgruppe verwendet Redmine als Projektmanagementsoftware zur Bereitstellung des PG-Wikis, der Projektverwaltung, als Ticketsystem und zur Dokumentenablage. Dieses ist zu finden unter `https://projekte.itmc.tu-dortmund.de/projects/pg-probio/`.

1.5 Strukturierung

Der Abschlussbericht ist wie folgt strukturiert: In Kapitel 2 werden die Grundlagen und die Ausgangslage der Projektgruppe dargestellt. Mit Kapitel 3 werden die Zielsetzung und der grobe Lösungsansatz der PG näher erläutert. Anschließend folgen in den Kapiteln 4, 5, 6 und 7 die Ergebnisse der einzelnen Kleingruppen. Kapitel 8 gibt einen Überblick über die Lizenzen der verwendeten externen Software-Pakete. Mit Kapitel 9 werden die Ergebnisse des Abschlussberichtes kurz zusammengefasst und in Kapitel 10 ein Ausblick für die Weiterentwicklung der erstellten Werkzeuge gegeben.

Kapitel 2

Grundlagen

Das am Lehrstuhl existierende Framework zur Modellierung und Erstellung domänenspezifischer Anwendungen besteht aus der *Dynamic Web Application* (*DyWA*) und dem *Java Application Building Center* (*jABC*). Mit der DyWA kann das Datenmodell einer domänenspezifischen Anwendung über eine Weboberfläche modelliert werden. Mit der Software jABC können Geschäftsprozesse mithilfe einer grafischen Oberfläche modelliert werden. Durch einen Export der Datentypen aus der DyWA in das jABC können die domänenspezifischen Typen im jABC bei der Modellierung der Geschäftslogik verwendet werden. Die im jABC modellierten Prozesse können in einem weiteren Schritt wieder in die DyWA importiert werden und können dort ausgeführt werden. Dabei können domänenspezifische Objekte, die in der DyWA angelegt werden, zur Parametrisierung der Prozesse eingesetzt werden. Das Zusammenspiel von DyWA und jABC ist in Abbildung 2.1 in Form eines Zustandsautomaten dargestellt.
Die DyWA und das jABC basieren auf dem Prinzip der modellgetriebenen Softwareentwicklung und nutzen Codegenerierung, um aus den Modellen der Datentypen und Prozesse kompilierbare und ausführbare Programmbausteine zu erhalten. Durch die Codegenerierung ist es aus Sicht eines Anwenders nicht erforderlich, eigenen Programmcode zu schreiben. Eine domänenspezifische Anwendung kann über

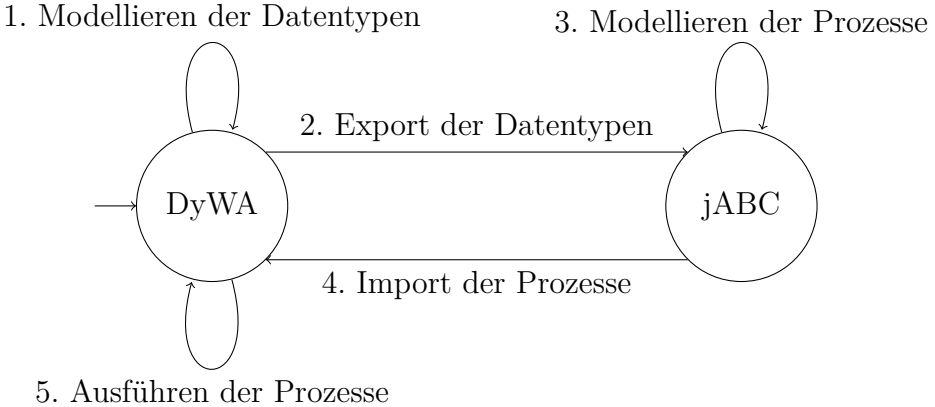

Abbildung 2.1: Zusammenspiel von jABC und DyWA

die verschiedenen Modelle schrittweise zusammengestellt und erweitert werden und wird schließlich in ausführbaren Code übersetzt und dem Anwender zur Verfügung gestellt. Mit Hilfe von Änderungen an den Modellen können auch Änderungen oder Ergänzungen der domänenspezifischen Anwendung erreicht werden, sodass keine Programmierkenntnisse für das Erstellen einer solchen Anwendung erforderlich sind.

Eine genauere Beschreibung der beiden vorhandenen Frameworks DyWA und jABC sowie der modellgetriebenen Softwareentwicklung und der Grundlagen der Codegenerierung folgt in den folgenden Abschnitten. Zunächst wird dazu die modellgetriebene Softwareentwicklung erläutert.

2.1 Modellgetriebene Softwareentwicklung

Die modellgetriebene Softwareentwicklung (engl. *Model Driven Software Development*, kurz *MDSD*) in Verbindung mit *Extreme Programming* basiert auf den drei folgenden Aspekten:

- Modellgetriebener Entwurf von Software,
- Serviceorientierung und
- die Fokussierung auf den Endanwender.

Hinter der modellgetriebenen Softwareentwicklung steckt der zentrale Begriff des Modells. Der Begriff umfasst die Aspekte der Abstraktion von der komplexen Realität, die Reduktion auf die (für den jeweiligen Zweck) wesentlichen Aspekte sowie die Zielorientierung des Modells [25]. Beispiele für Modelle sind die UML-Modelle, aber auch Prozessmodelle oder (formale) Grammatiken für (Programmier-)Sprachen. Die Kernideen der modellgetriebenen Softwareentwicklung bestehen in der Betrachtung des Modells als zentrales Entwicklungsartefakt, der automatischen Codegenerierung und der Entwicklung von *Domain Specific Languages* (*DSLs*). Mit DSLs erfolgt eine Spezialisierung auf eine konkrete Anwendungsdomäne und erlaubt so die bessere Einbindung von Experten unter Nutzung ihres Wissens. Bei der MDSD werden die Modelle bewusst entwickelt und sind die Grundlage der Codegenerierung.

Durch die MDSD in Verbindung mit der Fokussierung auf den Endanwender wird der Nutzer in alle Phasen des Lebenszyklus der Software einbezogen und wird in die Lage versetzt, die Ausgestaltung der Software maßgeblich steuern zu können. Mit der Bereitstellung von Bausteinen für Prozesse und Domäne kann der Nutzer die domänenspezifische Anwendung ohne Programmierkenntnisse selbst zusammenstellen und entwickeln.

Die Modellierung einer Anwendung findet auf verschiedenen Ebenen statt, die sich durch den Abstraktionsgrad bezüglich technischer Details unterscheiden. Die Modellierung auf unteren Ebenen ist in der Regel sehr technisch und betrifft die direkte Anbindung an eine Programmiersprache. Daher ist dort hinreichendes, technisches Wissen erforderlich. Auf den höheren Ebenen erfolgt die Modellierung auf einem hohen Abstraktionslevel, indem die Bausteine darunterliegender Ebenen verwendet werden.

Durch den Einsatz von Codegenerierung wird die Ausführbarkeit der erstellten Modelle gewährleistet. Zusammen bildet die Übersetzung aller Modelle, also die Modelle der Domäne und die Prozessmodelle, die Zielanwendung.

Im nächsten Abschnitt wird die Funktionsweise der DyWA näher erläutert.

2.2 DyWA

Basierend auf den Anforderungen und Zielen der Projektgruppe, wurde das DyWA-Framework (im Folgenden nur als DyWA bezeichnet) als Entwicklungsbasis gewählt. Die DyWA wurde am Lehrstuhl für Programmiersysteme (LS5) entwickelt (unter anderem als Grundlage für eine Bachelorarbeit [6]) und wird seitdem fortlaufend weiterentwickelt. Es handelt sich hierbei um ein Framework zur dynamischen Erstellung von Webanwendungen. Hieraus leitet sich auch der Name ab: **Dy**namic **W**eb **A**pplication. Die Idee für die DyWA entstand aus der Tatsache, dass es eine immer größere Herausforderung wird, Webanwendungen zu entwickeln. Nicht nur weil diese immer komplexer werden, sondern auch, weil viele Personengruppen an der Entwicklung beteiligt sind. Die beteiligten Personen lassen sich im Allgemeinen in Programmierer und Anwendungsexperten aufteilen. Dabei sind die Programmierer für die technischen Seite zuständig, wogegen die Anwendungsexperten sich mit der eigentlichen Domäne der Webanwendung beschäftigen. Ohne eine entsprechende Kommunikation kann die Entwicklung sehr mühsam werden. Gerade bei der Software- und Webentwicklung kann es passieren, dass sich die Anforderungen ständig ändern. Diese Änderungen können zum Beispiel mit einer Änderung der Domäne zusammenhängen. Dabei ist es oft der Fall, dass die Änderungen an die Programmierer weitergeleitet werden. Aber die Programmierer haben oft kein (oder nur ein sehr geringes) Wissen über die Domäne. Basierend auf den Lücken in den jeweiligen Gebieten kann es schnell zu Fehlern kommen.

Deshalb ist es notwendig, die Anwendungsexperten in die Entwicklung mit einzubeziehen. Dabei sollten sie nicht mit zu technischen oder abstrakten Komponenten konfrontiert werden. Im Gegensatz zu den Programmierern haben die Anwendungsexperten auf diesem Gebiet meist nicht viel Erfahrung. Es ist von Vorteil, ein Konzept zu entwickeln, dass beide Parteien mit ihren Wissenslücken unterstützt. Ein Ansatz der entwickelt wurde, um eine Verbindung zwischen beiden Parteien darzustellen, ist das Prototyping. Es werden fortlaufend Prototypen einer Anwendung erstellt und erweitert. Es wurden verschiedenste Ansätze entwickelt, die auch teilweise verschiedene Ziele verfolgen. Zum Beispiel soll bei einem Ansatz schon ein Programm mit gewissen Grundfunktionen erstellt werden, in einem anderen wird ein Prototyp erstellt, mit dessen Hilfe Erfahrungen gesammelt werden sollen. Problematisch bei vielen Ansätzen ist, dass ein gewisses technisches Wissen vorausgesetzt wird. Ideal ist ein Ansatz, der kein technisches Wissen von den Anwendungsexperten als gegeben setzt. Ein weiteres Problem stellen agile Änderungen dar. Agile Änderungen, die die Domäne beeinflussen, sollten auch möglich sein. Ein Weg, um agile Entwicklung zu unterstützen ist Service-Oriented-Computing. Die Webanwendung wird in Services zerlegt, die grundlegende Dienste zur Verfügung stellen. Geschäftsprozesse

können diese dann nutzen. Hier bleibt dennoch das Problem der Lücke zwischen den Kompetenzen der Programmierer und Anwendungsexperten bestehen.

Ziel ist es demnach, eine Architektur mit speziellen Komponenten zu entwickeln, in der die Programmierer domänenunabhängig und die Anwendungsexperten domänenspezifisch arbeiten können. Dabei sollte die Fehleranfälligkeit, die bei der notwendigen Kommunikation entsteht, minimiert werden. Im Rahmen der Bachelorarbeit wurde ein Prototyp einer Architektur entwickelt, die agile Entwicklung und Unterstützung beider Parteien ermöglichen soll. Die Anwendungsexperten werden sehr stark in den Fokus gesetzt, da sie auch ohne viel technisches Wissen mit der DyWA arbeiten können.

2.2.1 Grundlegende Konzepte und Architekturen

Für die DyWA wurde der Java EE Standard gewählt. Dieser bietet Techniken und Komponenten an, die für die Entwicklung sehr hilfreich sind. Darunter fallen die *Java Persistence API* (*JPA*) zur Persistierung von Daten, *Context and Dependency Injection* (*CDI*) für Abhängigkeiten zwischen Modulen und der *Servlet-Container* für die Kommunikation mit dem Anwender. Verwendet wird hierbei *HTML* um die Interaktion über einen Webbrowser zu ermöglichen. Als Datenbank wird *PostgreSQL* verwendet. Um Abhängigkeiten bei der Erstellung und Entwicklung zu verwalten, wird das Build-Tool *Maven* benutzt.

Für die Modellierung von Geschäftsprozessen wird *jABC* (Java Application Building Center) verwendet. Hiermit können Geschäftsprozesse in sogenannten Service Logic Graphs (SLGs) modelliert werden. SLGs sind gerichtete Kontrollflussgraphen, deren Knoten als Service Independent Building Blocks (SIBs) bezeichnet werden. Sie stellen Aktionen in Geschäftsprozessen dar und können Eingaben und Ausgaben verwalten. Dabei kann es immer nur ein Input-SIB in einem SLG geben, aber mehrere Output-SIBs. Diese speziellen SIBs werden mittels einem Icon dargestellt, welches einen senkrecht nach oben bzw. nach unten ausgerichteten Pfeil zeigt. Die Kanten werden als Branches bezeichnet und es kann je nach Bedarf pro SIB mehrere eingehende und ausgehende Branches geben. Werden während der Ausführung Daten benötigt, gibt es für jeden SLG einen Ausführungskontext, in dem diese gespeichert werden. Um neue Geschäftsprozesse zu modellieren, gibt es zwei Möglichkeiten. Die erste besteht darin, bestehende Methoden von Java-Klassen in SIBs (Services) zu überführen. Das Framework unterstützt hierbei die automatisierte Durchführung der Überführung von Java-Klassen in SIBs. Die Ausführung einer Aktion wird von dem SIB an die jeweilige Methode delegiert. Die zweite Methode besteht darin, aus schon vorhandenen SLGs SIBs zu erstellen.

Ein wichtiger Bestandteil des jABC ist der Codegenerator. Dieser überführt modellierte SLGs in Java-Code, der ausgeführt werden kann. Dabei wird aus jedem Prozess eine Klasse mit einer *execute*-Methode generiert. Für die Verwendung dieser generierten Prozess-Klassen wird durch die DyWA ein Controller generiert, der die Klassen als Methodenaufrufe innerhalb der Webanwendung zur Verfügung stellt.

2.2.2 Domäneneinbindung und Prozessintegration

Für die Anwendungsexperten ist es wichtig, im jABC domänenspezifische Services und Datentypen in Prozesse einbinden zu können. Deshalb ist die Bereitstellung einer Menge solcher Services und Daten durch die Programmierer sehr hilfreich. Dabei handelt es sich zum Beispiel um domänenspezifische Datentypen, die im jABC verwendet werden können. Diese werden vorher in der DyWA angelegt und dann exportiert. Die Anwendungsexperten sollten also aus ihrer Sicht in einer domänenspezifischen Umgebung arbeiten können, wogegen die Programmierer in einer unabhängigen Struktur arbeiten können. Umgesetzt wird das mit Hilfe eines Metaschemas, auf dem die DyWA arbeitet. Zu den Daten, die gespeichert werden sollen, werden noch zusätzlich Typinformationen gespeichert, um typisierte Daten zu realisieren. Die Typinformationen können im Betrieb angepasst werden, um konkrete Domänen zu modellieren. Diese Modellierung wird zudem an die Anwendungsexperten delegiert. Damit wir der Kompetenzbereich der Anwendungsexperten erweitert, ohne ihnen technisches Know-How vermitteln zu müssen. Weiterhin benötigen die Entwickler weniger fachliches Wissen.

Benutzt wird hierfür ein evolutionäres Typschema. Die gewünschten Objekte haben einen Namen und einen Typen, der auch aus mehreren Typen bestehen kann. Bei den Typen gibt es einmal die primitiven Typen, die auch nativ vom DyWA-Framework unterstützt werden. Bei diesen Typen handelt es sich um bekannte Datentypen wie String, Int und Boolean. Die nicht primitiven Datentypen hingegen entstehen während der Modellierung und bestehen meist aus mehreren Attributen, die wiederum entweder primitiv sind oder Assoziationen zu Objekten anderer modellierter Typen bilden. Als kleines Beispiel könnte man zunächst einen Typen Pupil betrachten. Der besitzt einen Vor- und Nachnamen, die man beide mit dem primitiven Typen String ausdrücken kann. In einem weiteren Schritt könnte man einen Typ Schoolclass erstellen, der aus mehreren Schülern besteht. Da der Typ Pupil schon vorher erzeugt wurde, kann er als Attribut genutzt werden. Mit diesem Verfahren ist es ohne Probleme möglich auch ein komplexes Domänenmodell zu modellieren.

Weiter besitzt die DyWA einen Codegenerator, mit dem aus den erstellten Typen Java-Klassen und Controller generiert werden. Die Controller ermöglichen die Arbeit mit den Typen (Erstellen, Ändern, Löschen von Objekten der Typen). Die Generierung dieser Klassen und Controller geschieht im Hintergrund und ist für die Anwendungsexperten nicht sichtbar. Das DyWA-Framework verfügt zusätzlich noch über einen sogenannten *Morpher*. Dieser wird für die Kommunikation zwischen Datenbank und Webanwendung in Bezug auf die Objekte und Typen benötigt. Diese werden in der Datenbank in einem Metaschema gespeichert (mehr dazu im Abschnitt 2.2.4). Um in der Webanwendung mit den Objekten und Typen arbeiten zu können, müssen die Informationen aus der Datenbank geholt und so umgewandelt werden, dass sie wieder in der Webanwendung für die Anwendungsexperten angezeigt werden können.

Änderungen während bzw. nach der Entwicklung lassen sich durch das Online - Offline-Komponentenprinzip einfach einpflegen. Die Webanwendung ist eine Online-Komponente. Bei der DyWA handelt es sich ebenfalls um eine Online-Komponente. Die DyWA bildet z. B. die Administrationsoberfläche für die Modellierung des Domä-

nenmodells. Änderungen können jederzeit vorgenommen werden. Durch die Codegeneratoren wird dann eine aktualisierte Version erstellt, die online geschaltet werden kann. Codegeneratoren stellen zusammen mit dem jABC die Offline-Komponenten da. Selbst Änderungen an der Domäne sind kein Problem, da Typen und Klassen auf einem Metaschema arbeiten. Da bei Typänderungen die Prozesse, die diese Typen und Klassen benutzen, offline angepasst werden, können die Prozesse in der Online-Komponente einfach mit ausgetauscht werden.

2.2.3 Architektur

Bei dem Architekturmodell handelt es sich grob um eine klassische Mehr-Schichten-Architektur. Vorhanden sind eine Persistenz-, eine Geschäftslogik- und eine Präsentationsschicht. Abbildung 2.2 zeigt das Architekturmodell.

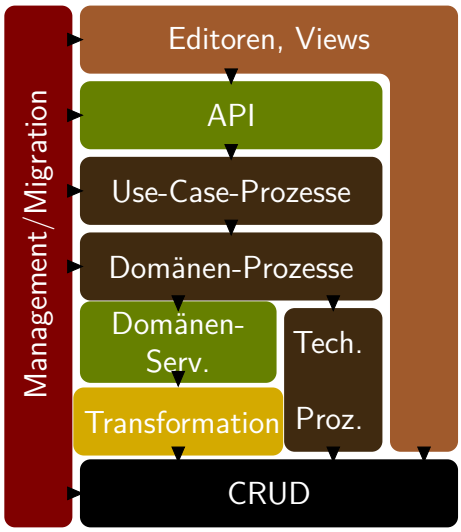

Abbildung 2.2: Architektur der DyWA

Als Grundlage sieht man hier die Datenbank mit den Operationen **C**reate, **R**ead, **U**pdate und **D**elete (CRUD). Die Daten liegen in Form des Metaschemas vor und werden von der DyWA verwaltet. Die generierten Services werden in den Komponenten Transformation und Domänen-Service verwaltet. Diese Komponenten agieren direkt mit dem Metaschema der Datenbank. Das Metaschema wird zudem durch die Domänen-Service-Komponente für die höheren Schichten verdeckt. Es ist auch möglich, über technische Prozesse die Datenbankanbindung zu realisieren. Diese arbeiten aber domänenunspezifisch und müssen extra modelliert werden. Da die Anwendungsexperten über den Web-Editor Typinformationen editieren können, greift dieser Teil des DyWA-Frameworks auch auf die Datenbank zu. Hierbei werden die Daten dynamisch in eine domänenspezifische Darstellung überführt. Bei den Domänen-Prozessen werden einzelne Services kombiniert und in den Use-Cases modelliert man die Prozesse, die die Anwendungsschnittstelle des Kunden definieren. Über die API-Komponente kann man mit externen Komponenten kommunizieren. Da die Migration und das Management von Daten gerade bei agilen Änderungen

sehr wichtig ist und an verschiedenen Stellen stattfinden kann, ist die Komponente Migration/Management in das gesamte Architekturmodell integriert. Im Folgenden werden die einzelnen Schichten etwas näher erläutert.

2.2.4 Persistenz-Schicht

Wie schon erwähnt, werden neben den eigentlichen Daten auch Informationen über die Domäne gespeichert. Das angewandte Typsystem basiert darauf, native und komplexe Datentypen zu unterscheiden. In der DyWA werden folgende native Datentypen unterstützt:

- String
- Long
- Double
- Boolean
- Timestamp
- List<String>

Um die komplexen Datentypen zu beschreiben, gibt es die Typen

- DBObject
- List<DBObject>

Bei Listen ist auf die Restriktion zu achten, dass immer nur ein modellierter Typ abgespeichert werden kann. Das Typschema lässt sich je nach Bedarf auch ohne Probleme erweitern. In Abb. 2.3 ist ein vereinfachtes Diagramm zur Umsetzung zu sehen.

An der Spitze der Hierarchie sieht man die Klassen DBIdentified und DBNamed. Durch das Attribut *id*, welches unique ist, wird es der DyWA ermöglicht, Objektinstanzen eindeutig zu identifizieren. Das Attribut *name* ist lediglich für den Benutzer zur Behandlung auf der Weboberfläche nötig. Damit kann er Instanzen im Benutzer-Interface selber unterscheiden. Die Klasse DBType ist die tatsächliche Umsetzung der Typen. Hierbei werden die Attribute *deleted* und *fields* eingeführt. Durch *deleted* wird angezeigt, dass der Typ gelöscht wurde. Dabei wird er nicht tatsächlich aus der Datenbank gelöscht. Die Gründe dafür werden in Abschnitt 2.2.6 erläutert. In *fields* werden die Attribute des erstellten Typs wie vorher erwähnt mit einem Namen und Typen geschrieben. Diese Liste besteht aus dem Typen DBField. Hierbei wird über *fieldType* der Typ des Attributes angegeben. Dabei handelt es sich um die schon vorgestellten nativen und komplexen Typen. Im Falle eines komplexen Typen kann man den referenzierten Typ über *typeConstraint* festlegen.

Erstellte Objekte werden über *type* einem Typen zugeordnet. Die einzelnen Attribute von Objekten werden über das Attribut *properties* realisiert. In der Datenbank

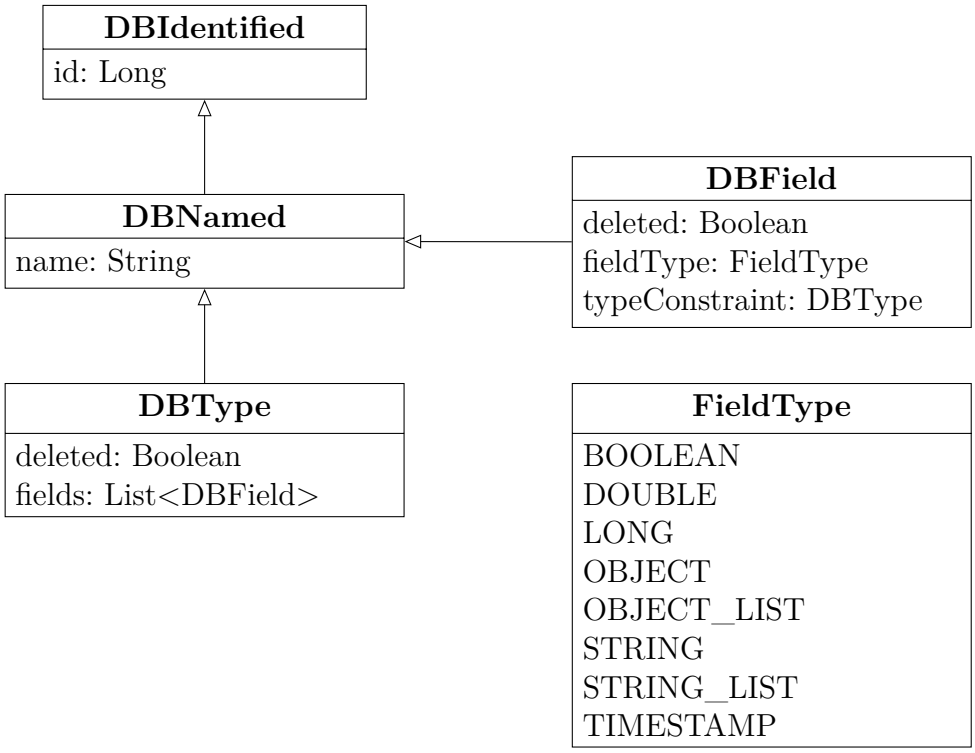

Abbildung 2.3: Vereinfachtes UML-Diagramm zur Typsystemumsetzung (Entnommen aus [6])

existiert eine Tabelle DBProperties. Dort werden die Werte der Attribute eines Objektes gespeichert. In der Klasse DBProperties werden die unterstützten Datentypen jeweils als Set gespeichert (Abb. 2.4). In Abb. 2.5 ist die Umsetzung der einzelnen Properties und Speicherung dieser zu sehen. Es gibt eine Tabelle für jeden möglichen nativen Typen. Über diese Tabellen werden dann die eigentlichen Werte gespeichert.

Zum besseren Verständnis wird ein kleines Beispiel aufgeführt: Erstellt wird ein Typ *Person* mit Vorname und Nachname. Dabei ist dann in DBType *name* = *Person* und *id* eine zufällig vergebene id (in diesem Beispiel die 1). Durch *fields* werden der Vorname und der Nachname realisiert. Es gibt in diesem Fall zwei *fields* mit den Namen Vorname und Nachname. Als *fieldType* ist jeweils String eingetragen. Ein *typeConstraint* wird nicht benötigt. Es kann nun ein DBObject erstellt werden namens Max Mustermann. Die *type_id* ist 1. In der Tabelle DBStringProperty werden zwei Einträge hinzugefügt für Max und Mustermann. Auch hier werden über DBIdentified ids vergeben. Darüber findet die Zuordnung statt.

Über eingebaute Controller im DyWA-Framework und entsprechende Interfaces in der Web-Schicht können die Anwendungsexperten die Domäne modellieren. Für komplexe Klassen erstellen sie Instanzen der Klasse DBType und speichern diese ab. In der ersten Evolutionsstufe befüllen sie die DBFields mit nativen Datentypen. Wird aus diesen erstellten Typen ein neuer komplexer Typ erstellt, kann man die zuvor erstellten Typen als Attribute nutzen. Dazu werden sie in *fields* gespeichert und die *typeConstraints* werden vergeben. Werden in Objekten schon vorhandene Objekte referenziert, wird dies über einen Zeiger realisiert. Hierbei wird die eindeu-

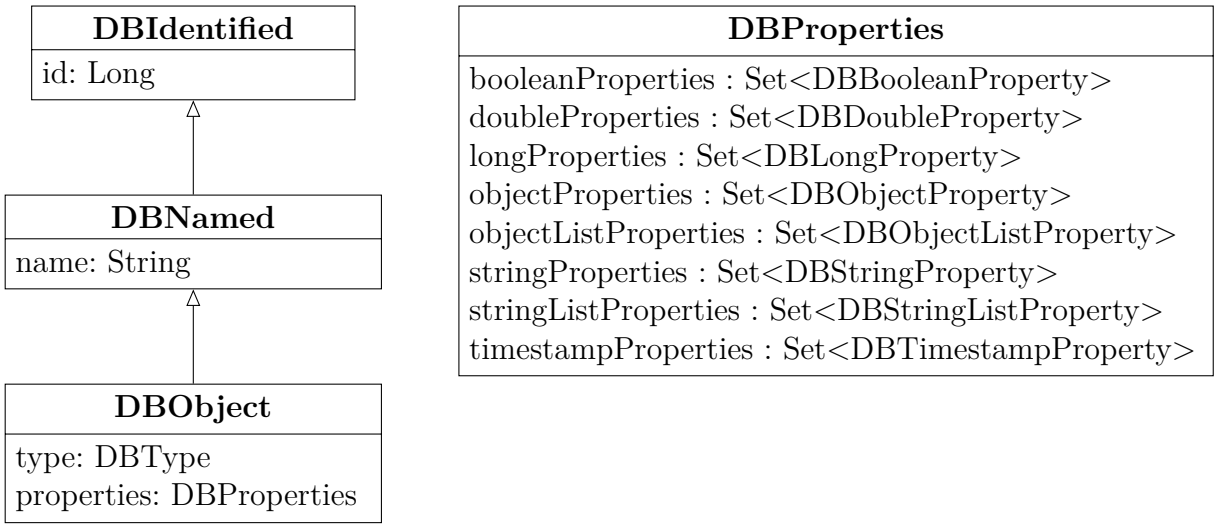

Abbildung 2.4: Vereinfachtes UML-Diagramm zur Assoziation von Typen und Daten (Entnommen aus [6])

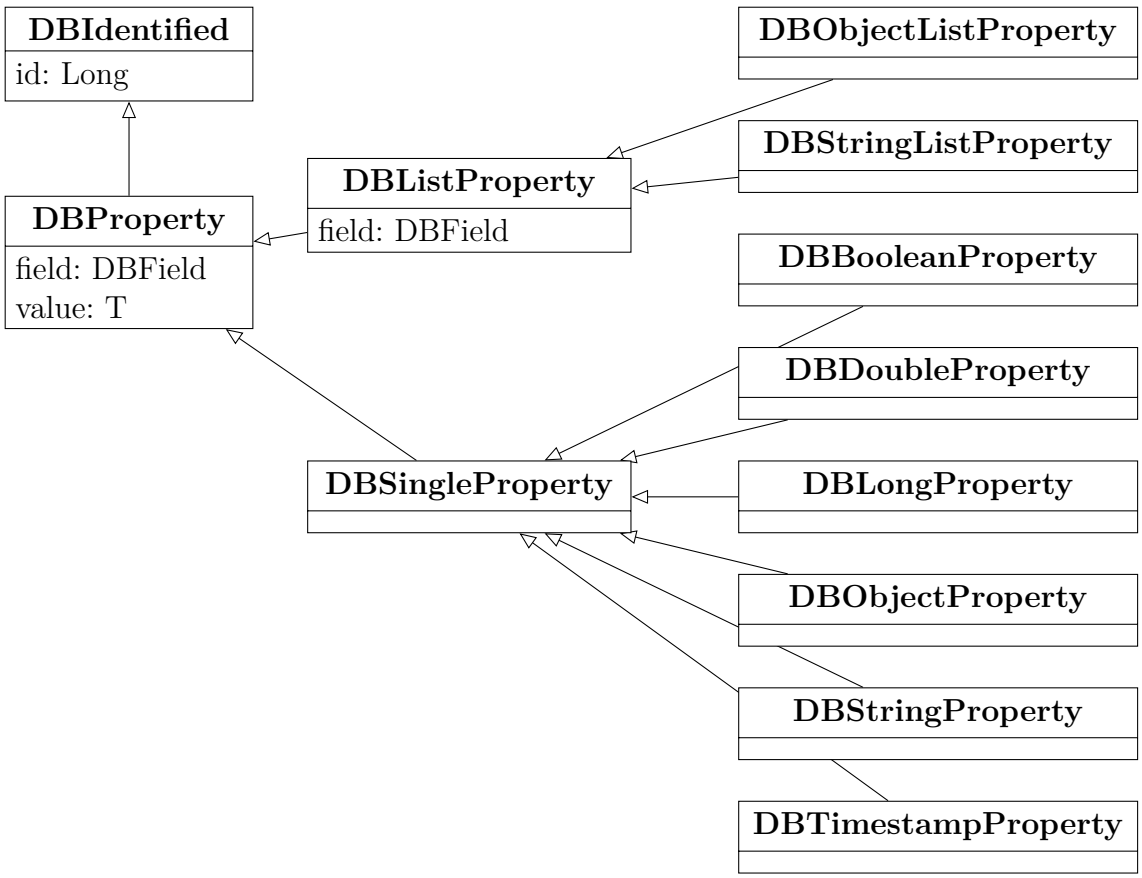

Abbildung 2.5: Vereinfachtes UML-Diagramm zur Speicherung (Entnommen aus [6])

tige ID verwendet. Die Referenzierung von mehreren Objekten, zum Beispiel über eine Liste, wird über eine Mapping-Table realisiert. Dieses Verfahren kann man dann nach Belieben fortführen. Intern werden vom Framework bei der Speicherung und Synchronisation von Daten noch weitere Schritte durchgeführt. Es werden weitere Klassen benutzt, die für jeden Typen nochmal eigene Klassen erstellen und verwalten. Darunter zählen auch die Vorgänge während der Umbenennung von Instanzen und Attributen.

2.2.5 Transformations-Schicht

Um Domänen im jABC bereitzustellen, enthält die DyWA einen Generator. Dieser liest die Typen und Felder aus der Datenbank aus und erzeugt daraus domänenspezifische Java-Klassen. Für jeden neu erstellten Typen wird also eine neue Klasse erstellt. Die Attribute dieser Klasse werden aus den angegebenen *fields* des Typen generiert. Für jede Klasse werden auch Controller generiert, die die CRUD-Operationen der Datenbank unterstützen. Die Operationen werden dann in den Kontext der Domäne eingebettet. Dies ist nötig, da in der Datenbank ein allgemeines Schema genutzt wird. Die Anwendungsexperten arbeiten aus ihrer Sicht mit den domänenspezifischen Daten und das allgemeine Schema wird durch die Controller vor ihnen versteckt. Die erstellten Klassen werden dann an die Programmierer weitergeleitet und diese betten sie in das jABC ein. Zwischen der Datenbank und der domänenspezifischen Typen wird mittels IDs referenziert. Somit ist z. B. eine Umbenennung kein Problem.

2.2.6 Prozess-Schicht

Die Modellierung der Geschäftsprozesse findet im jABC4 statt. Das eingesetzte Build-Tool Maven erzeugt für Typklassen und Controller jeweils Maven-Artefakte. Diese können im jABC genutzt werden. Auch andere jABC-Projekte können integriert werden. Somit können Anwendungsexperten Prozesse separat entwickeln, sowie in verschiedenen Projekten wiederverwenden. Dabei ist es nicht notwendig, dass sich z. B. die Anwendungsexperten für die Domänen-Prozesse (siehe 2.2.3) und für die Use-Case-Prozesse absprechen. Domänenunabhängige Prozesse (z. B. der Versand von Mails), die von den Anwendungsexperten benutzt werden müssen, können schon im Vorfeld integriert werden. Sollte zu Beginn nicht klar sein, wie diese auszusehen haben, gibt es auch die Möglichkeit, diese als Schnittstellen zu integrieren und später zu modellieren.

Wie bereits erwähnt, werden durch das jABC die Prozesse in ausführbaren Code übersetzt. Ein Generator der DyWA iteriert beim Import der Prozessmodelle über die Prozesse und erstellt einen Prozesscontroller. Über diesen Controller können die einzelnen Prozesse über Methoden ausgeführt werden.

Derzeit ist es über das Framework noch nicht möglich, Bedienoberflächen zu erstellen. Der Grund hierfür ist, dass diese nicht vereinheitlicht betrachtet werden können, da es, basierend auf den Anforderungen, einfach zu viele mögliche Variationen gibt. Durch die verwendete Struktur und Architektur ist es aber möglich, diese

parallel zu erstellen. Die Komponenten der Oberflächen müssen dann nur mit den Geschäftsprozessen verbunden werden. Das Framework besitzt dennoch generische Komponenten zur Unterstützung bestimmter Aktivitäten. Darunter fallen die schon vorgestellten Typ- und Objekt-Editoren. Das Kernstück des Frameworks bildet der Prozess-Selektor (Abbildung 2.6). In diesem können Prozesse unter Parameterangaben ausgeführt werden.

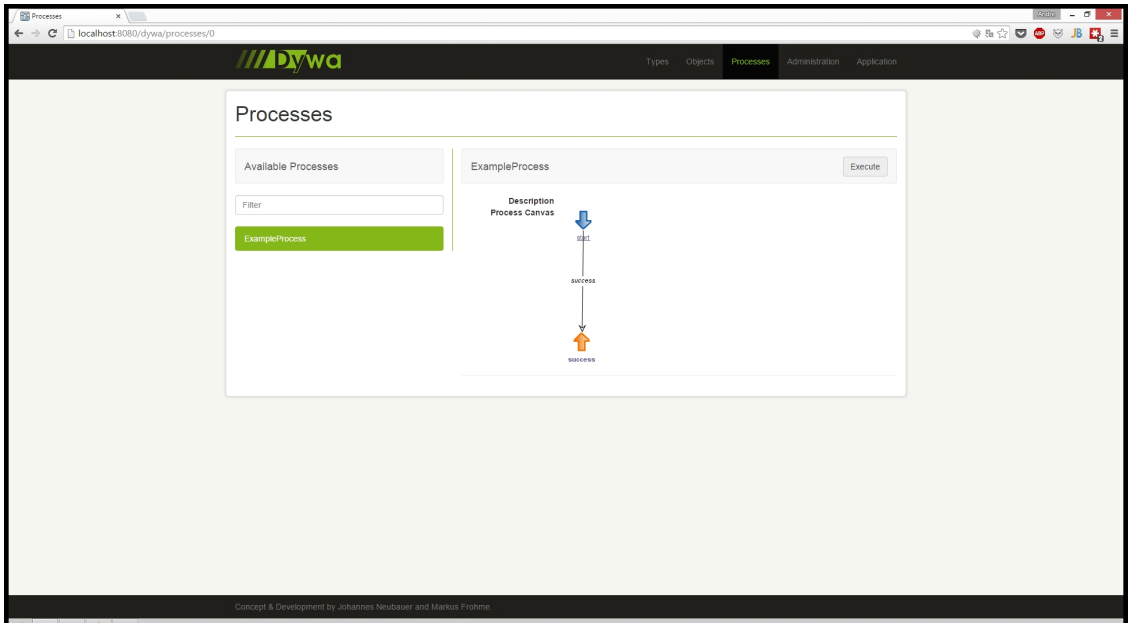

Abbildung 2.6: Prozess-Selektor der DyWA

Hinzu kommt der Schema-Synchronisierer. Dieser ist zuständig für die Migration der Daten. Nicht mehr benötigte Attribute oder Typen können gelöscht werden. Objekte, denen bestimmte Properties durch eine Migration fehlen, können aktualisiert werden. An dieser Stelle sieht man u. a. den Grund, warum Typen nur als gelöscht markiert werden. Im Entwicklungsverlauf können so Veränderungen nachvollzogen und behandelt werden.

2.2.7 Migration und agile Änderungen

In diesem Abschnitt wird kurz erläutert, wie die Möglichkeiten der agilen Domänenmodellierung und die Migration von Daten realisiert wurden. Das hier behandelte Problem lässt sich so formulieren: Wie kann man Änderungsanforderungen an die Geschäftsprozesse und den Domänenkontext umsetzen?

DBTypes und DBObjects lassen sich, wie schon erwähnt, erstellen, ändern und löschen. Die DyWA kümmert sich intern um die nötigen Änderungen in der Datenbank. Es müssen noch die Auswirkungen auf die Geschäftsprozesse betrachtet werden, wenn die Domäne geändert wird. Glücklicherweise ergibt sich für die Geschäftsprozesse dadurch kein Problem. Da die Prozesse auf einem Snapshot der Domäne arbeiten, können Änderungen vorgenommen werden, die erst zur Geltung

kommen, wenn eine neue Version online gestellt wird. Durch den vorhandenen Morpher wird das Metaschema sofort mit angepasst. Falls ein Prozess ein neues Objekt erzeugt, dessen Properties geändert wurden, wird dieses Objekt einfach ohne die Properties erzeugt, bis der Geschäftsprozess angepasst wird. Als Wert wird dann null gesetzt. Die Umbenennung von Objekten hat ebenfalls keine Auswirkung, da die Objekte über eine eindeutige ID identifiziert werden. Trotz solcher Änderungen kann die Webanwendung weiter genutzt werden.

Um die Geschäftsprozesse letztlich der neuen Domäne anzupassen, können sie wie bisher angepasst werden. Durch das DyWA-Framework werden die neuen Typen übergeben und können die Alten ersetzen. Dann wird eine neue Anwendungsschnittstelle generiert und in eine neue Webanwendung eingebettet. Ein Problem ergibt sich lediglich dadurch, das alte Datensätze nicht über die neuen Properties verfügen. Daher müssen alte Objekte mit der neuen Domäne synchronisiert werden. Diese Synchronisation kann auf zwei Wegen ausgeführt werden. Einmal können die Programmierer eine Datenbank-basierte Migration ausführen. Wie der Name schon sagt, werden die Änderungen direkt in der Datenbank vollzogen. Eine weitere Möglichkeit besteht über den Synchronisierer. Hier sehen die Anwendungsexperten veraltete Objekte und können diese editieren. Als gelöscht markierte Objekte können endgültig gelöscht werden. Es wäre auch denkbar, eigens dafür geschriebene Services oder Prozesse einzubetten.

2.2.8 Agile DyWA-gestützte Entwicklung

In diesem Abschnitt wird kurz erläutert, wie eine DyWA-unterstützte Entwicklung ablaufen kann:

Zu Beginn ist keine Domäne vorhanden. Jedoch können die Anwendungsexperten über die Weboberfläche eine Domäne mit DBTypes gestalten (Abbildung 2.7). Im Objekt-Editor (Abbildung 2.8) können domänenspezifische Daten eingepflegt werden. Die Anwendungsexperten können den Programmierern aber auch einen Export von alten Daten geben, die von den Programmierern einpflegt werden. Ein erfolgreicher Import kann auch sofort im Editor verifiziert werden.

Unabhängig davon können die Anwendungsexperten schon die Prozesse modellieren. Domänenspezifische Klassen können aus der DyWA expotiert und im jABC zur Verfügung gestellt werden. Domänenunabhängige Services können von den Programmierern auch schon eingepflegt werden (z. B. für den Versand von E-Mails). Sind die Anwendungsexperten fertig, übermitteln sie ihre Prozesse, Typen und Objekte an die Programmierer, die dann mit der DyWA einen Prototypen erstellen. Die Weboberfläche kann ganz unabhängig von all diesen Schritten modelliert werden.

Änderungen der Domäne und der Prozesse können einfach von den Anwendungsexperten in den generischen Komponenten von DyWA und im jABC durchgeführt werden. Wenn dafür auch die alten Daten aktualisiert werden müssen, weil zum Beispiel ein neues Attribut hinzukommt, kann das über den Synchronisierer geschehen.

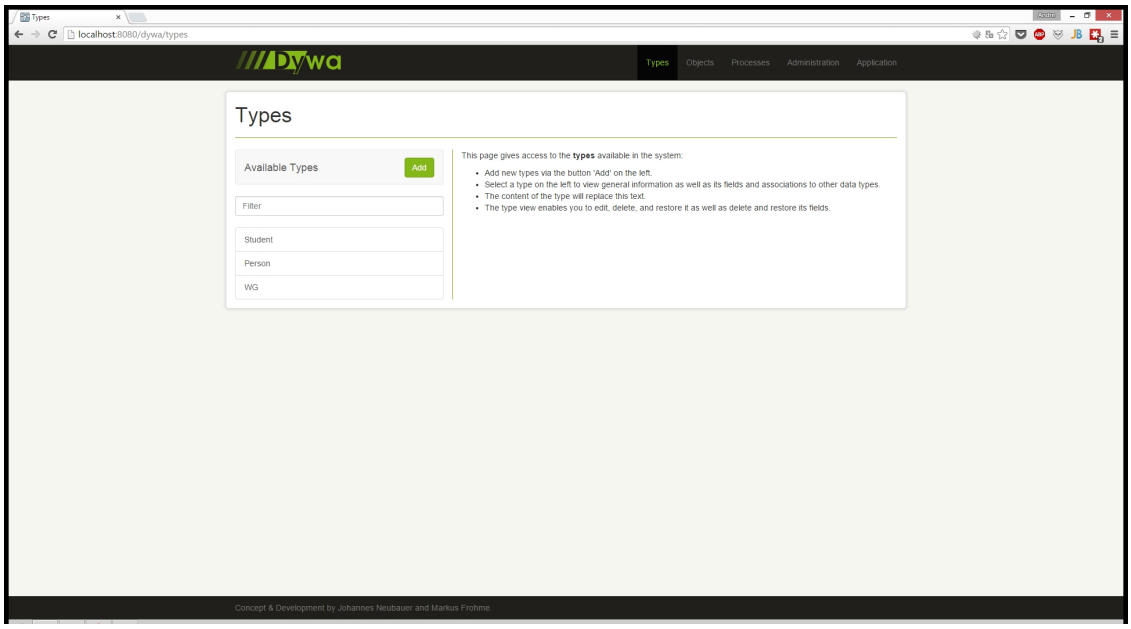

Abbildung 2.7: Fenster zur Erstellung von Typen

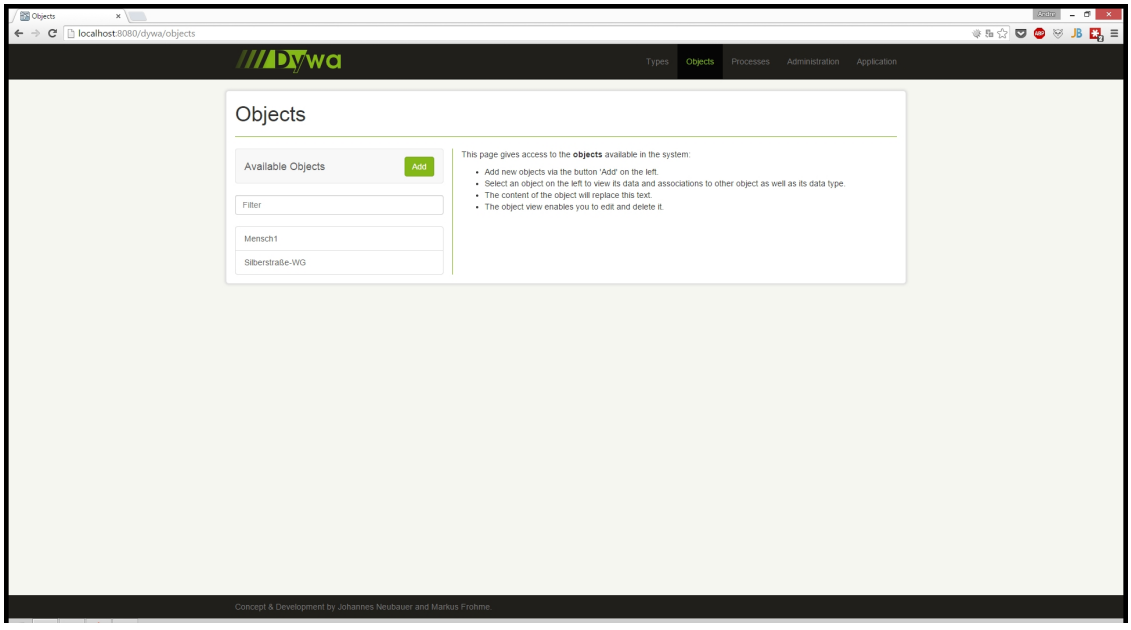

Abbildung 2.8: Fenster zur Erstellung von Objekten

2.2.9 DyWA 0.5 Release

Zahlreiche Änderungen an der DyWA traten zum Release-Zeitpunkt der DyWA-Version 0.5 ein. Diese sollen in diesem Abschnitt kurz eingeführt werden.

Die wichtigste Änderung ermöglicht es nun, dass sich eigene Web-Plug-ins bei der DyWA registrieren können. Diese Web-Plug-ins werden dann im Header der DyWA in einem Drop-Down-Menü verlinkt. Durch die Verwendung von Plug-ins sind Erweiterungen und Ergänzungen an der DyWA leicht per Plug-in modular bereitstellbar. Es werden keine Eingriffe in die DyWA selbst nötig. Die Erweiterungen der PG wurden nach dem Release der DyWA-Version 0.5 umgestellt, so dass diese alle als Web-Plug-ins in der DyWA registriert werden.
Weitere Änderungen sind:

- Es besteht die Möglichkeit, Eventlistener für Entity-Lifecycles (PrePersist, PostUpdate, etc.) zu registrieren. Darüber können z.B. Security-Constraints illegale Datenänderungen eines Users verhindern.
- Es besteht nun die Möglichkeit, in die Codegenerierung der domänenspezifischen Klassen einzugreifen um beispielsweise Annotationen hinzuzufügen. Zusätzliche Annotationen könnten für Security-Anforderungen notwendig sein.
- In der DyWA-Version 0.5 werden Prozesscontroller und Prozesshandler nur noch für konfigurierbare Business-Prozesse generiert.

2.2.10 Prozessaufrufe und Chaining

Prozesse können in einer Prozesskette (Chaining) ausgeführt werden, dies zentralen Mechanismus dar, der bei der Ausführung von Prozessen genutzt werden kann. Vor der Ausführung bzw. Fortsetzung eines Prozesses wird eine Kette von Aufrufen sogenannter *Handler* abgearbeitet, wobei am Ende der Kette der Aufruf einer Methode auf dem Prozess erfolgt. Die einzelnen Handler behandeln jeweils einen Aspekt für Prozessaufrufe wie z. B. die Überprüfung der Zugriffskontrolle und reichen die Aufrufe an das nächste Kettenglied weiter. Diese „Implementierung“ eines Call-Stacks wird als *Chaining* bezeichnet und ist seit der Version 0.4 der DyWA verfügbar.

Das Chaining basiert auf dem Prinzip, dass für jede orthogonale Funktionalität ein eigener *Handler* implementiert wird. Damit folgt das Chaining dem Programmierparadigma der aspektorientierten Programmierung. Die Handler rufen sich entlang einer aufgebauten Kette (*Chain*) gegenseitig auf und erhalten die Rückgabe des jeweils nachfolgenden Handlers, die vor der eigenen Rückgabe manipuliert werden kann. Dabei kann sowohl vor als auch nach dem Aufruf einer Methode des nächsten Handlers zusätzliche Logik ausgeführt werden. Somit können durch zusätzliche Handler in der Aufrufkette beliebige Features um die Prozesse herum gelegt werden, wie beispielsweise die Überprüfung von Zugriffskontrollmechanismen oder die gesonderte Behandlung von Rückgaben von Prozessen im Fall von Prozessunterbrechungen.

Am Ende einer Kette von Handlern steht ein *DefaultHandler*. Dieser enthält die ursprüngliche Funktionalität eines ProcessControllers und ruft direkt eine Methode

des Prozesses auf. Wir haben u. a. für die Behandlung von Unterbrechungen in Prozessen einen zusätzlicher Handler implementiert, welcher die Logik für die Persistierung der Prozessinstanz enthält, ggf. Timer anlegt und erhaltene Daten anschließend nach außen weitergibt. Dieser Handler wird als *ExternalHandler* bezeichnet.

Prioritäten Jeder Handler besitzt eine Priorität, die darüber entscheidet, an welcher Position der Handler in einer Kette eingefügt wird. Durch die Reihenfolge der Handler in der Chain wird die Reihenfolge der Abarbeitung bestimmt. Der DefaultHandler besitzt standardmäßig die kleinste Priorität, da dieser stets das letzte Element der Kette bilden soll. Der ExternalHandler besitzt eine größere Priorität, sodass er vor dem DefaultHandler in der Kette steht. Eine Chain wird innerhalb einer Methode eines ProcessControllers aufgebaut und dann gestartet. Für die Ausführung der Prozesse über einen ProcessController wird der Kette vor dem ersten Aufruf eines Kettenglieds ein *SetupHandler* vorangestellt. Dieser Handler initialisiert die Kette mit der Prozessinstanz, auf der alle Aufrufe von Prozessmethoden erfolgen. Mit Hilfe der Methoden *setup* und *forward* wird die Prozessinstanz allen Handlern der Kette zur Verfügung gestellt.

Allgemeiner Ablauf der Abarbeitung einer Chain Die Funktionsweise des Chainings ist in Abbildung 2.9 in allgemeiner Form dargestellt. Dabei bezeichnet „before“ den Teil der Logik, der vor dem *forward*-Aufruf in einem Handler der Kette ausgeführt wird und die Eingaben vor der Weitergabe an die *forward*-Methode des nächsten Handlers manipulieren kann. Analog steht „after“ die für Programmlogik, die nach dem Aufruf der *forward*-Methode des nächsten Handlers in der Kette ausgeführt wird. Dort ist die Manipulation des erhaltenen Rückgabewertes vor der Rückgabe an den eigenen Aufrufer möglich. Bei dem in der Abbildung mit „Start“ bezeichneten Handler handelt es sich um den SetupHandler, auf dem bereits beim Erzeugen die *setup*-Methode aufgerufen wird. Anschließend folgt mit den *handle*-Aufruf der Start der Verarbeitung durch die Chain. Der mit „Ende“ bezeichnete Handler repräsentiert den DefaultHandler, welcher innerhalb der *handle*-Methode einen Methodenaufruf auf einer Prozessinstanz durchführt.

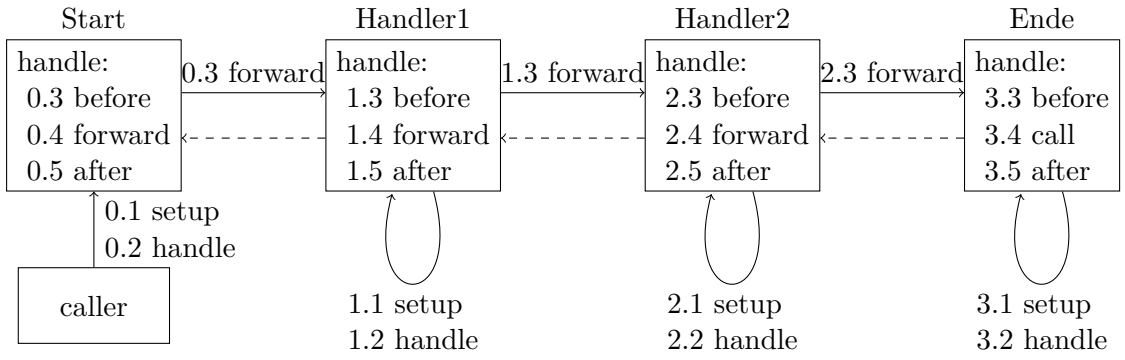

Abbildung 2.9: Grundidee des Chaining

Beispiel Wir betrachten nun ein Minimalbeispiel, in dem lediglich die drei zuvor genannten Handler (SetupHandler, ExternalHandler und DefaultHandler) vorkommen und eine Unterbrechung im Prozess existiert. Der ProcessController ruft die *handle*-Methode auf dem SetupHandler auf und startet damit die Chain. Dieser besitzt bereits eine Referenz auf die aktuelle Prozessinstanz und ruft auf dem nächsten Handler, dem ExternalHandler, die *forward*-Methode auf. Die Logik innerhalb des ExternalHandlers wird erst ausgeführt, nachdem die *forward*-Methode auf dem DefaultHandler aufgerufen wurde. Daher wird der ExternalHandler erst nach einer Rückgabe dieser Methode aktiv. Der DefaultHandler führt dann auf der ihm bekannten Prozessinstanz die *execute*-Methode (bzw. im Falle einer vorherigen Unterbrechung eine *resume*-Methode) aus. Die Rückgabe wird an den ExternalHandler zurückgereicht, wobei die Rückgabe bei vollständigem Ausführen eines Prozesses den String „success“ als Status enthält und bei einer Unterbrechung zum Beispiel den String „waitForTimerGraph“ (für die Unterbrechung durch einen Timer) enthält. Im ersten Fall reicht der ExternalHandler die Rückgabe direkt an den eigenen Aufrufer aus der Chain weiter. Im Falle einer Unterbrechung persistiert der ExternalHandler die laufende Prozessinstanz, startet einen Timer und reicht die Rückgabe danach unverändert weiter.

2.2.11 Zusammenfassung und Erweiterungen im Rahmen der PG

Zusammenfassend kann festgehalten werden, dass ein Framework geschaffen wurde, das die Kommunikation zwischen Programmierern und Anwendungsexperten erleichtert. Da beide Parteien fast völlig unabhängig voneinander arbeiten können, gibt es weniger Verzögerungen im Ablauf.

Die Möglichkeit der agilen Anpassung durch die Architektur bietet eine auf komplexen Vorgängen basierende Umgebungsentwicklung, die aber für die Anwendungsexperten sehr vereinfacht und domänenspezifisch über den Webbrowser angezeigt wird. Die Anwendungsexperten müssen sich nicht mehr um technische oder abstrakte Komponenten kümmern. Durch das genutzte Metaschema lassen sich beliebige Domänen gestalten und verwalten.

Das DyWA-Framework liegt derzeit selbst als Prototyp vor. Einige Komponenten und Möglichkeiten wurden noch nicht implementiert. So könnte man einige Vorgänge noch automatisieren. Als Beispiel kann man die Einbindung von vorgefertigten Datenbankanfragen (siehe Kap. 6) nennen.

Zudem ist es durch die Architektur auch möglich, Datenstrukturen wie Sets, Stacks oder weitere in das DyWA-Framework mit einzubinden. Dabei kommt es dann natürlich auf die Anforderung an die zu erstellenden Webanwendungen an.

Im Rahmen der Projektgruppe wurden viele Erweiterungen an der DyWA durchgeführt. Die Möglichkeiten des Frameworks wurden durch Plugins erweitert. Dabei wurden folgende Plugins eingefügt:

- DyWA Logic zur Beschreibung zusätzlicher Ausführungssemantiken für Prozesse (Kap. 4)

- DyWA GUI zur Gestaltung einer GUI (Kap. 5)
- DyWA Search zur Erstellung und Persistierung von Datenbankanfragen (Kap. 6)
- DyWA Security um die Anwendung zu sichern (Benutzerrollen, Passwörter, etc.) (Kap. 7)

Die meisten Änderungen an der DyWA traten dabei beim Release der Version 0.5 in Kraft (siehe 2.2.9).

2.3 jABC

Mit dem Framework jABC4 werden Geschäftsprozesse über eine grafische Oberfläche modelliert. Die modellierten Prozesse können in einem nachfolgenden Schritt mit Hilfe der Codegenerierung in ausführbaren Java-Code übersetzt und in andere Anwendungen integriert werden. So können beispielsweise die im jABC modellierten Prozesse in die DyWA integriert und über die Weboberfläche der DyWA ausgeführt werden.

In jABC4 modellierte Prozesse sind gelabelte Kontrollflussgraphen, die um Datenfluss angereichert werden. Die Knoten eines solchen Kontrollflussgraphen stellen die Aktivitäten des Prozessmodells dar. Die Kanten beschreiben *Branches*, mit den abhängig vom Ausgang der vorherigen Aktivität der Übergang des Kontrollflusses zur nächsten Aktivität definiert wird. Die Kommunikation der Aktivitäten eines Prozessmodells untereinander erfolgt mit Hilfe des Datenflusses. Jede Aktivität besitzt eine Menge von Eingabeparametern sowie eine Menge von Branches für den Übergang des Kontrollflusses. Zu jedem dieser Branches der Aktivität existiert eine Menge von Ausgabeparametern. Somit ist durch die Parametrisierung der Aktivitäten die Übergabe von Daten von einer Aktivität zu der nachfolgenden Aktivität möglich. Der Datenfluss wird unter der Berücksichtigung von strenger Typisierung modelliert, sodass der aus den Prozessen generierte Programmcode typsicher ist. Aus formaler Sicht sind die Prozessmodelle in jABC Kripke Transitionssysteme, eine Mischung aus Kripke-Strukturen und gelabelten Transitionssystemen und besitzen eine eindeutig definierte Semantik [19]. Im Umfeld von jABC werden Prozesse als *Service Logic Graphs* (*SLGs*) bezeichnet und die einzelnen Aktivitäten eines SLGs werden als *Service Independent Building Blocks* (*SIBs*) bezeichnet.

Ein Prozess besteht aus einer Menge von SIBs, welche die einzelnen Aktivitäten des Prozesses darstellen. Ein SIB repräsentiert eine einfache oder eine komplexe Aktion. Eine einfache Aktion entspricht einem gekapselten Methodenaufruf (z. B. einer Java-Methode), der ohne Verschachtelungen für die Parameter auskommt. Solches SIBs werden als *atomare Aktivitäten* bezeichnet. Komplexe Aktionen entsprechen einem kompletten Prozessaufruf und werden als *Abstraktionsaktivitäten* bezeichnet. Somit führt die Verwendung komplexer Aktivitäten zu der Modellierung hierarchischer Prozessmodelle. Mit der Wiederverwendung bestehender elementarer Bausteine und Prozesse als Subprozesse kann die Komplexität einzelner Prozesse überschaubar gehalten werden. Zudem ist die Abstraktion von technischen Details möglich.

Mit der Modellierung von Prozessen als reine Kontrollflussgraphen muss jede Variation eines Prozesses explizit modelliert werden. Die Flexibilität der Prozessmodelle kann durch Hinzunahme eines Datenflusses erhöht werden, bei denen neben normalen Datenobjekten auch Prozessinstanzen als Parameter von SIBs genutzt werden können. Diese Prozessinstanzen im Datenfluss können dann über spezielle SIBs ausgeführt werden. Dabei entsteht durch die Verwendung eines Prozesses erster Ordnung[1] als Parameter eines Prozesses zweiter Ordnung[2] und der Möglichkeit der Ausführung des übergebenen Prozesses eine flexible Prozessmodellierung. Folglich müssen nicht alle Varianten eines Prozesses explizit im Voraus modelliert werden, sondern ergeben sich durch die Varianten der übergebenen Prozesse zur Laufzeit der Anwendung. Prozesse, die andere Prozesse als Parameter erhalten, werden als *Prozesse höherer Ordnung* bezeichnet, die Modellierung als *Higher-Order Process Engineering* (*HOPE*). Der Ansatz weißt Ähnlichkeit zu Funktionen höherer Ordnung auf, welche in funktionalen Programmiersprachen wie Haskell verwendet werden können [19].

Der Datenfluss innerhalb eines Prozessmodells wird mit Hilfe eines Ausführungskontextes umgesetzt.

2.3.1 Ausführungskontext von Prozessen

Für die Ausführung eines SLGs und die Kommunikation der Aktivitäten des SLGs untereinander wird ein Ausführungskontext verwendet. Dabei handelt es sich um eine Menge getypter Variablen, die geteilte Ressourcen darstellen und als *Kontextvariablen* bezeichnet werden. Die Werte dieser Variablen können während der Ausführung des SLGs verändert werden, solange die Einschränkungen durch den jeweiligen Datentyp der Variable erfüllt werden. Jede Kontextvariable besitzt einen Namen, unter dem sie innerhalb des Prozessmodells angesprochen werden kann, und einen festen Typ. Bei dem Typ handelt es sich um eine Java-Klasse, ein Interface oder eine Enumeration.

Mit Prozessen höherer Ordnung und der damit verbundenen Notwendigkeit, Prozessinstanzen in Kontextvariablen ablegen zu können, kann der Typ einer Kontextvariable auch ein Graphtyp (*Service Graph* oder *Interface Graph*) sein. Zusätzlich können alle diese Typen für Kontextvariablen (Datentypen und Graphtypen) Typparameter wie bei Java Generics enthalten. Mit der Hinzunahme von Typparametern können wiederverwendbare Prozesse für verschiedene Anwendungsdomänen modelliert werden. Dabei müssen aus technischen Gründen den Typparametern von Java-Datentypen zur Modellierungszeit bereits Typargumente zugeordnet werden. Typparameter können innerhalb des Gültigkeitsbereichs eines Graphen als Typ für Variablen des Ausführungskontextes oder wieder als Typargument parametrisierbarer Java- oder Graphtypen verwendet werden [18, 19].

Der Ausführungskontext eines Prozessmodells besitzt einen lokalen Gültigkeitsbereich (Scope), sodass die Kontextvariablen einer Hierarchieebene nicht in den Subgraphen innerhalb des SLGs sichtbar sind. Damit kann ein Subprozess nicht

[1] Ein Prozess erster Ordnung erhält keine weiteren Prozesse als Parameter.

[2] Ein Prozess zweiter Ordnung erhält u. a. einen Prozess erster Ordnung als Parameter.

auf die Variablen des Ausführungskontextes des Prozesses zugreifen, in den dieser Subprozess eingebettet ist. Außerdem existieren in jABC4 keine globalen Variablen. Folglich erfolgt der Informationsaustausch über Hierarchieebenen hinweg lediglich mit der Parametrisierung der Subprozesse, durch Dependency Injection bzw. der Nutzung von Inversion of Control.

Die Prozessmodelle in jABC bestehen aus verschiedenen SIBs, die zusammen einen Geschäftsprozess der Anwendung beschreiben. Diese SIBs werden im Folgenden näher betrachtet.

2.3.2 Prozesse und SIBs

Ein Prozess in jABC besitzt einen definierten Startpunkt in Form eines SIBs. Dieses Start-SIB wird auch als Input-SIB bezeichnet und definiert die Eingabeparameter des Prozesses. Dabei entsprechen die Eingabeparameter des Prozesses (aktuelle Parameter) den Ausgabeparametern des Start-SIBs (formale Parameter). Anschließend wird bei der Ausführung des Prozesses der Kontrollfluss entlang der gelabelten Kanten abhängig von Ausgang des jeweiligen SIBs verfolgt, bis ein End-SIB (auch als Output-SIB bezeichnet) erreicht wird. Jeder Kontrollflusspfad des Graphen ist durch genau ein Output-SIB abgeschlossen. Über das Output-SIB des Prozesses werden die Ausgabeparameter des Prozesses definiert und der Name des Output-SIBs wird als Branch für die Fortsetzung der Ausführung des Prozesses der darüber liegenden Hierarchieebene genutzt [19].

Eine Aktivität in Form eines SIBs ist parametrisierbar, d. h. sie erhält erforderliche Daten über die Eingabeparameter und liefert Ergebnisse von Berechnungen oder Teilprozessen durch die Ausgabeparameter. Dabei können Eingabeparameter als dynamische Parameter mit Werten der Kontextvariablen belegt werden oder ihnen werden als statische Parameter Konstanten zugewiesen, die zur Modellierungszeit definiert werden. Die Ausgabeparameter von SIBs werden Kontextvariablen zugewiesen, sofern die Daten für weitere Aktivitäten des SLGs benötigt werden. Ein SIB kann mehrere ausgehende, gelabelte Kanten für den Kontrollfluss (Branches) besitzen. Die Ausgabeparameter sind immer an einen Branch gebunden, wobei jeder Branch mehrere Ausgabeparameter besitzen kann. Häufig besitzen SIBs verschiedene Branches, unter anderem für den Erfolgsfall und den Fehlerfall, deren Ausgabeparameter sich voneinander unterscheiden.

Bei den modellierten Prozessen wird zwischen Service Graphs und Interface Graphs unterschieden. Ein Service Graph ist ein Prozess, der neben Input- und Output-SIBs Kanten (und weitere SIBs) für den Kontrollfluss besitzt. Damit ist ein Service Graph die Realisierung einer speziellen Abfolge von Aktivitäten in Form eines Prozesses. Dagegen besteht ein Interface Graph nur aus Input-SIBs und Output-SIBs und definiert lediglich die Schnittstellen für die Nutzung in Prozessen höherer Ordnung. Somit entspricht ein Interface Graph einer Schnittstellendefinition, also einem funktionalen Interface mit genau einer Methodensignatur. Die konkrete Umsetzung der Funktionalität eines Interface Graphs erfolgt über einen oder mehrere Service Graph, der den Interface Graphen implementiert und den Kontrollfluss des Graphen definiert. Interface Graphen sind zudem als Datentyp für Kontextvariablen

und Parameter von SIBs verwendbar und tragen so zur Umsetzung der Modellierungsmöglichkeit für Prozesse höherer Ordnung bei [19].

Bei den SIBs gibt es für einfache Methodenaufrufe atomare Aktivitäten und für Aufrufe von Subgraphen Abstraktionsaktivitäten. Bei atomaren Aktivitäten wird zwischen drei Untertypen unterschieden: *Service SIBs* repräsentieren einen Methodenaufruf, dessen Rückgabewert einen beliebigen Typ außer Boolean und Enumeration darstellen kann. Für Methoden mit booleschen Rückgabetyp gibt es *Decision SIBs* und für Methoden, die Enumerations liefern, existieren *Enumeration SIBs*. Durch den jeweiligen Typ des Rückgabewertes wird auch ein wesentlicher Teil der Menge der Branches für die Fortsetzung des Kontrollflusses bestimmt. Beispielsweise besitzen Decision SIBs für die beiden möglichen Ergebnisse *true* und *false* einen Branch; dazu kommen Branches für die Fehlerbehandlung und für ein fehlendes Ergebnis.

Für die Modellierung von Prozessen höherer Ordnung werden sowohl atomare Aktivitäten als auch Abstraktionsaktivitäten als Graphen modelliert, sodass diese als Parameter von Prozessen höherer Ordnung verwendet werden können. Zudem besteht die Notwendigkeit, Prozessinstanzen in Kontextvariablen ablegen zu können, welche dann als Parameter von anderen Prozessen genutzt werden können. Ein häufiges Anwendungsszenario dabei ist das Erzeugen eines Prozesses, der zunächst in einer Kontextvariable abgespeichert wird und später als Parameter eines Prozesses genutzt wird. Dazu existieren in jABC *Konstruktoraktivitäten*, die Prozesse instanziieren und in Kontextvariablen ablegen können. Konstruktoraktivitäten besitzen keine Eingabeparameter und nur Typparameter, falls der zu instanziierende Service Graph ein generischer Graphtyp ist. Beim Ausführen einer solchen Aktivität wird der Service Graph instanziiert und in einer Kontextvariablen abgelegt, sodass eine Konstruktoraktivität mit einem *new*-Aufruf in Java vergleichbar ist. Eine Prozessinstanz, die in einer Kontextvariable gespeichert ist, kann mit Hilfe des SIBs ausgeführt werden, welches durch den Interface Graphen definiert wird [19].

Neben dem Kontrollfluss ist in jABC4 der Datenfluss für den Austausch von Daten zwischen den Aktivitäten eines Prozesses zu modellieren. Dabei kann die Modellierung des Datenflusses mit Hilfe der Vorkonfiguration von SIBs vereinfacht werden.

2.3.3 Vorkonfiguration von SIBs

Bei der Parametrisierung von SIBs ist für jede Aktivität die komplette Kommunikation mit den Variablen des Ausführungskontextes und die Verwendung statischer Werte manuell zu modellieren. Dies wird bei einer großen Anzahl an SIBs innerhalb eines Prozessmodells schnell unübersichtlich und mühsam. Daher existiert die Möglichkeit, vorkonfigurierte SIBs zu erstellen, die in der Regel auf eine spezifische Anwendungsdomäne zugeschnitten sind und bezüglich des Datenflusses bereits vordefiniert sind.

Mit der Vorkonfiguration von SIBs wird eine Abstraktion von der manuellen Konfiguration von SIBs geschaffen. Vorkonfigurierte SIBs können über spezielle, in jABC modellierte Prozessmodelle als SIB-Bibliotheken für die Modellierung anderer

Prozesse zur Verfügung gestellt werden. Vorkonfigurierte SIBs entstehen dabei aus vorhandenen SIBs, deren Parametrisierung mit Konstanten und Defaultwerten vorgenommen wird. Zusätzlich können die vorkonfigurierten Aktivitäten umbenannt werden, einem spezifischen Package zugeordnet werden und mit einem passenden grafischen Icon für die Darstellung der Aktivität in der jABC-Oberfläche versehen werden [19].

Bei der Verwendung vorkonfigurierter SIBs wird die vordefinierte Belegung der Ein- und Ausgabeparameter der SIBs übernommen und die benötigten Kontextvariablen werden automatisch angelegt. Die Defaultbelegung der Parameter kann anschließend manuell verändert werden.

Mit Hilfe von Plugins kann der Funktionsumfang von jABC zur Modellierung von Prozessen erweitert und auf die Anforderungen einer speziellen Anwendungsdomäne zugeschnitten werden.

2.3.4 jABC Plugins

Plugins in jABC können zur Erweiterung der Basisfunktionalität von jABC eingesetzt werden. Dabei können die Features von Plugins global für alle Anwendungsdomänen verwendet werden oder speziell auf einzelne Domänen zugeschnitten sein. jABC Plugins können auf einzelnen SIBs (*SIB-Level Plugins*) oder auf ganzen SLGs (*SLG-Level Plugins*) operieren [26].

Für jABC sind bereits einige Plugins standardmäßig vorhanden. Darunter fallen unter anderem das *Tracer Plugin* für die Ausführung von Prozessen innerhalb von jABC. Mit dem Tracer kann innerhalb von jABC ein Prozessmodell ausgeführt werden, solange es keine Datentypen der DyWA verwendet (da diese eine spezielle Ausführungsumgebung erfordern). Zudem existieren die beiden Plugins *LocalChecker* und *ModelChecker*. Diese können für die Überprüfung von lokalen oder globalen Bedingungen in Prozessen eingesetzt werden und ermöglichen damit die Umsetzung von Funktionen im Bereich des Model Checkings.

Im nächsten Abschnitt werden die Grundlagen der Codegenerierung erläutert, die auch bei der Umwandlung von jABC Prozessmodellen in ausführbaren Code einer Zielsprache wie Java eingesetzt werden.

2.4 Codegenerierung

Als Codegenerierung wird der Vorgang bezeichnet, aus bestehenden Modellen Code zu generieren, ohne dass ein Programmierer diesen von Hand programmieren muss. Die Möglichkeiten der Codegenerierung reichen von Arbeitserleichterungen, wie dem automatischen generieren von Gettern und Settern bis hin zur vollständigen Generierung ausführbaren Codes. Codegenerierung ist eine der zentralen Aufgabenstellungen dieser PG – in diesem Abschnitt sollen daher einige Grundlagen und Begriffe im Bezug auf Codegenerierung erläutert werden, um die verwendeten Codegeneratoren in das Themengebiet einordnen zu können.

Die Aufgabe ist ähnlich derer von Compilern, welche aus einem in einer Programmiersprache verfasstem Quelltext ausführbare Dateien generieren. Bei modell-

basierten Codegeneratoren, wie sie in der PG verwendet werden sollen, herrschen in dieser Phase andere Voraussetzungen. Da meist in eine Hochsprache wie C++ oder Java übersetzt wird, kann für den Übersetzungsschritt hin zum ausführbaren Code auf bestehende Compiler zurückgegriffen werden. Dementsprechend kann das Kompilieren durch einen Compiler als eine zusätzliche Phase des Übersetzungsprozesses angesehen werden.

2.4.1 Techniken der Codegenerierung

Codegeneratoren haben wie Compiler die Aufgabe, etwas in einer Quellsprache vorliegendes in eine Zielsprache zu übersetzen. Quell- und Zielsprache sind üblicherweise durch die Anwendungsfälle gegeben, lediglich die Implementierungssprache ist frei wählbar. Um z. B. Bootstrapping zu ermöglichen, bietet es sich an, den Codegenerator in der gleichen Programmiersprache zu schreiben, in die er auch übersetzt.

Neben der Entscheidung für die Implementierungssprache gibt es verschiedene Designansätze für die eigentliche Implementierung des Codegenerators. Hier geht es insbesondere um die Ausgabebeschreibung, d. h. die Struktur und das Erscheinungsbild des generierten Codes in der Zielsprache sowie die *Generation Logic*. Letztere bestimmt das Mapping zwischen Quell- und Zielsprache, also wie genau übersetzt wird. Dies kann auch das Erstellen neuer Dateien und Aufteilen des übersetzten Textes auf diese Dateien beinhalten.

Da in der Literatur unterschiedliche Ansätze zur Klassifikation der bestehenden Techniken existieren, wird hier beispielhaft die Klassifikation von Kleppe [17] verwendet. Kleppe unterscheidet wie folgt:

Hard-Coded oder Model-Transformation Hier wird unterschieden, wie die Transformationsregeln definiert sind. Im ersteren Ansatz sind die Transformationen simpel fest im Quellcode verankert.

Die zweite Variante begreift den Codegenerator als Menge von Regeln, die von einem passenden Tool angewandt werden und zur Übersetzung führen. Dies erlaubt eine größere Flexibilität auf Kosten der Effizienz. Je nach verwendetem Tool kann die gleiche Regelmenge verwendet werden, um in unterschiedliche Sprachen zu übersetzen.

Source-Driven oder Target-Driven Translation In diesem Fall wird unterschieden, wie der Übersetzungsprozess organisiert ist. Der Source-Driven-Ansatz stellt die Quelldaten in den Vordergrund. Hier wird das Modell Schritt für Schritt abgearbeitet und übersetzt.

Dementgegen steht der Target-Driven-Ansatz. Hier steht die Zielsprache im Vordergrund. Der Zielcode wird hier sequenziell generiert, das Modell immer dann nach den Informationen gefragt, die gerade benötigt werden.

Concrete oder Abstract Form Target Es ist möglich, entweder direkt in die Syntax der Zielsprache zu übersetzen, sodass diese direkt ausführbar (kompilierbar) ist oder in ihre abstrakte Form. Die abstrakte Form einer Sprache ist im Grunde ihrerseits ein Modell, welches ebenfalls noch übersetzt werden

muss. Hier ist insbesondere vorteilhaft, dass die abstrakte Form nach dem Übersetzungsvorgang noch zur Verfügung steht und so für Erweiterungen und Optimierungen verwendet werden kann, bevor sie in den Zielquelltext übersetzt wird.

Programmierung eines Codegenerators

Grundsätzlich kann ein Codegenerator in einer beliebigen Programmiersprache in einer Textdatei geschrieben werden, sofern eine Schnittstelle zu dem zu übersetzenden Modell existiert. In diesem Fall verwirklicht er automatisch den von Kleppe beschriebenen Ansatz der Hard-Coded Transformation. Die Ausgabe eines solchen Codegenerators wird in der Regel durch simple Ausgabe von Strings in eine oder mehrere Dateien realisiert. Je nach Sprache kann daher die nötige Verwaltung der Strings für zusätzlichen Aufwand und Fehlerquellen, wie die vergessene Maskierung von Sonderzeichen, sorgen.

Diese Art von Codegeneratoren sind für kleine, einfache Projekte meist ausreichend. Je komplexer die Aufgabenstellung wird, umso aufwendiger wird auch die Programmierung des Codegenerators, sodass irgendwann der Aufwand für Programmierung und Wartung den Nutzen des Codegenerators übersteigt. Dies hängt insbesondere damit zusammen, dass die meisten Programmiersprachen wenig Unterstützung für die Verwaltung komplexer Modelle und die Erstellung großer Mengen von Text verfügen.

Regelbasierte Transformation

Neben der eben beschriebenen, festgeschriebenen Übersetzung im Quellcode lassen sich Codegeneratoren auch über Transformationsregeln realisieren. Hier existiert eine Menge von Regeln, die beschreibt, wie jedes einzelne Element aus dem Quellmodell in die Zielsprache zu übersetzen ist. Diese Regeln werden mit Hilfe einer *Transformation Engine* auf das Quellmodell angewandt. Ein vollständiger Codegenerator kann als Kette mehrerer solcher Transformationen verstanden werden, an deren Ende die Transformation in die Zielsprache steht. Das komplexe Problem der Transformation kann so auf viele kleine, leichter zu handhabende, Probleme herunter gebrochen werden. Die regelbasierte Transformation erlaubt es zudem, statt direkt in den Quelltext der Zielsprache, in einem Zwischenschritt zunächst in die abstrakte Form zu transformieren. Wie bereits beschrieben, kann diese Form vor der letztendlichen Übersetzung noch für Erweiterungen verwendet werden.

Template-basierte Codegenerierung

Neben diesen beiden Ansätzen ist es auch möglich, die Codegenerierung Template-basiert erfolgen zu lassen. Ein Template besteht aus statischen und dynamischen Anteilen, welche zusammen mit den Daten aus dem Modell von einer Engine ausgewertet werden, um so den Code zu erzeugen.

Abbildung 2.10 zeigt die grundlegende Funktionsweise der Template-basierten Codegenerierung. Oben links befindet sich ein Template zur Generierung von Quell-

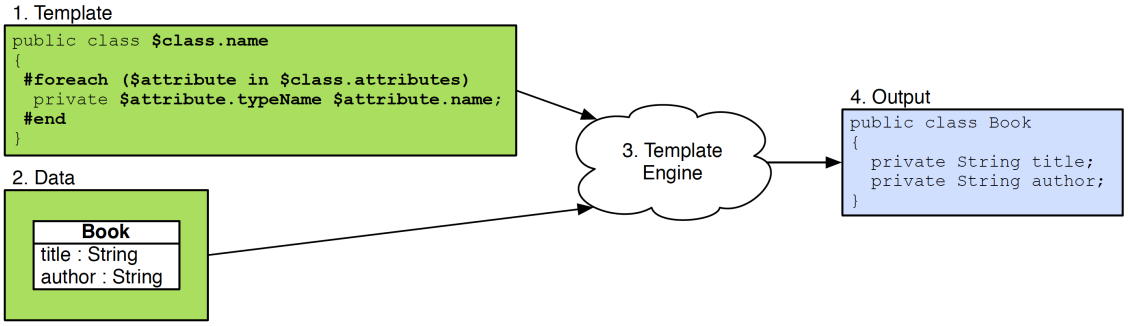

Abbildung 2.10: Template-basierte Codegenerierung [15]

text, darunter eine UML-Klasse (unten links). Die dynamischen Anteile des Templates sind durch die Zeichen # und $ zu erkennen. Die Engine generiert in diesem Beispiel zusätzlich zum statischen Text für jedes Attribut im Diagramm eine eigene Zeile mit passendem Namen und Datentyp und übernimmt den Klassennamen. Es sei angemerkt, dass dieses simple Beispiel keine tatsächliche Funktionalität erzeugt.

Nach Kleppes Klassifizierung fällt die Template-basierte Codegenerierung in den Bereich der *Hard-Coded Transformation* und *Target-Driven Translation*. Verglichen mit der Programmierung eines Codegenerators muss sich der Entwickler nicht mehr um Maskieren von Sonderzeichen kümmern, da Templates kein Quelltext sind.

Nachteilig an dieser Technik ist allerdings, dass die Logik zwischen Teilproblemen im Normalfall nicht mitmodelliert wird. Dieser Kontrollfluss muss dann aufwendig über Zusatzbedingungen in den Regeln des Templates realisiert werden.

Falls die Ausgabe der Codegenerierung in mehreren Dateien erfolgen soll, muss für jede Ausgabedatei ein eigenes Template existieren, da ein Template einer Datei entspricht [15]. In einem solchen Fall muss das Modell für jedes Template gelesen werden, was bei großen Modellen zu Performanceeinbußen führen kann.

Round-Trip-Engineering und Full-Code-Generation

Grundsätzlich können Codegeneratoren in zwei große Gruppen eingeteilt werden. Solche, die vollständigen, ausführbaren Code generieren, und solche, die lediglich Stubs oder andere Arten von unvollständigem Code generieren, welcher vom Entwickler noch vervollständigt werden muss.

Round-Trip-Engineering Mit Round-Trip-Engineering wird der Arbeitsprozess bezeichnet, der entsteht, wenn der verwendete Codegenerator keinen ausführbaren Code generiert. Für den Ansatz der modellbasierten Entwicklung eignet sich diese Variante nicht, hebelt sie doch einen zentralen Vorteil, das Wegfallen der Nachbearbeitung des generierten Quelltextes, aus.

Da sowohl im Quelltext als auch im Modell gearbeitet wird, müssen zwangsläufig auch Änderungen in beide Richtungen, von Modell zu Code und umgekehrt, übernommen werden. Daher auch der Begriff „Round Trip Engineering“. Jeder Entwickler kennt die Probleme, die entstehen, wenn zwei Teammitglieder ohne Absprache Änderungen an der gleichen Datei vornehmen. Ähnlich sieht es

hier aus. Insbesondere muss sichergestellt sein, dass vom Entwickler vorgenommene Änderungen am Code nicht bei einer Neugenerierung aus dem Modell heraus überschrieben oder unbrauchbar gemacht werden. Eine Möglichkeit, solche Probleme zu vermeiden, besteht darin, Änderungen am Quelltext nur in bestimmten Bereichen zu erlauben. Automatisch generierter Code darf nicht verändert, sondern nur ergänzt werden. Ebenfalls muss sich der Entwickler darauf beschränken, keinen Code zu schreiben, der auch vom Codegenerator erstellt worden sein kann.

Eine Anwendung dieser Variante besteht darin, sich aus dem Modell nur die Klassendefinitionen und Funktionsköpfe, beispielsweise aus einem UML-Klassendiagramm, erzeugen zu lassen. Hier muss der Entwickler direkt im Code die leeren Funktionen füllen. Werden jedoch Änderungen an den Funktionen und Klassen oder auch ganz neue Klassen benötigt, dürfen diese nicht einfach so hinzugefügt werden, sondern müssen im Modell gepflegt werden, um die Änderungen dann generieren zu lassen.

Grundsätzlich ist es auch möglich, in beide Richtungen vollständig zu synchronisieren. In der Praxis ist dies jedoch häufig nur möglich, wenn Modell und Programmiersprache auf demselben Abstraktionslevel liegen [15]. Es ist klar, dass die Sinnhaftigkeit, ein Modell zu verwenden, nicht mehr wirklich gegeben ist, soll es doch gerade den Abstraktionsgrad gegenüber der Implementierungssprache erhöhen.

Full-Code-Generation Im Gegensatz zum Round-Trip-Engineering wird bei der Full-Code-Generation, wie der Name schon sagt, vollständiger Code erzeugt. Bei diesem Ansatz sind manuelle Änderungen am erzeugten Quelltext daher auch nicht vorgesehen. Da alle Änderungen auf der Ebene des Modells erfolgen, verschwindet auch die Problematik der Synchronisation von Änderungen. Bei einer Neugenerierung des Codes kann der bisherige Code einfach überschrieben werden, ohne dass Probleme entstehen. Dass nun an nur noch einer Stelle, dem Modell, gearbeitet werden muss, erleichtert den Entwicklungsprozess. Der erzeugte Code ist bei dieser Variante eher als Zwischenschritt oder Nebenprodukt anzusehen. Es ist nötig, um am Ende ein ausführbares Programm zu erhalten, aber ohne tatsächliche Bedeutung für die Entwicklung. Analog dazu ist es grundsätzlich auch möglich eine Abstraktionsebene tiefer den, vom Compiler einer Hochsprache generierten, Maschinencode händisch zu bearbeiten. Während eine solche Nachbearbeitung früher durchaus üblich war, sind heutige Compiler so effizient, dass der Maschinencode nicht mehr verändert werden muss.

Dazu sei gesagt, dass mitnichten immer ein sofort ausführbares Programm am Ende der Full-Code-Generation stehen muss. Je nach Projekt kann es sinnvoll oder nötig sein, nur Teile des Projekts automatisch generieren zu lassen, während andere Teile von Hand geschrieben werden. Erst nach Zusammenführung von generiertem und handgeschriebenem Code lässt sich das Programm dann ausführen. Im Gegensatz zum Round-Trip-Engineering werden die generierten Quelltexte jedoch nicht mehr verändert.

Ob Full-Code-Generation möglich ist, hängt vor allem von der, durch sie erzeugten Programmiersprache ab. Hierzu müssen einerseits genügend Informationen in der Sprache vorhanden sein, um die Erzeugung vollständigen Codeszu ermöglichen. Gleichzeitig darf dadurch der Abstraktionslevel der Sprache nicht so weit in die Höhe gehen, dass er auf dem gleichen Level liegt wie das Modell, aus dem der Code generiert wird. Auch hier würde die modellbasierte Entwicklung keinen Vorteil mehr bieten, da direkt in der Programmiersprache entwickelt werden könnte.

Das Ziel der Projektgruppe soll mit Full-Code-Generation für domänenspezifische Bestandteile erreicht werden, da es nur so möglich ist, Änderungen an der domänenspezifischen Anwendung vorzunehmen, ohne einen Programmierer hinzuziehen zu müssen.

2.4.2 Vorhandene Generatoren

Die DyWA und das jABC enthalten bereits einige Codegeneratoren, die für den Export des Datenmodells aus der DyWA und den Import der jABC-Prozessmodelle in die DyWA eingesetzt werden.

Der Codegenerator für den Export des Datenmodells aus der DyWA ist als ein Maven-Plugin realisiert, welches in Kombination der Programmiersprachen Java und Scala nach dem Prinzip der Template-basierten Codegenerierung funktioniert. Dieser Codegenerator ist direkt im Code implementiert. Ähnlich verhält es sich für den Codegenerator, welcher die Controller für die Prozesse generiert, damit diese in der DyWA ausgeführt werden können. Anders sieht es bei dem Codegenerator aus, der die modellierten jABC-Prozesse in ausführbaren Java-Code übersetzt. Dieser Generator ist selbst als ein jABC-Prozess modelliert und verwendet spezielle SIBs für die Nutzung von Templates. Die Sammlung von Prozessen, die den Codegenerator bildet, wird mit Hilfe des bestehenden Codegenerators für Prozesse übersetzt und ist dann als Maven-Plugin nutzbar. Somit werden neue Codegeneratoren für Prozesse nach dem Prinzip des Bootstrapping gewonnen.

2.5 Bioinformatik

Die Bioinformatik (engl. *bioinformatics*, auch *computational biology*) ist eine interdisziplinäre Wissenschaft, die Probleme aus den Lebenswissenschaften mit theoretischen computergestützten Methoden löst. Sie hat zu grundlegenden Erkenntnissen der modernen Biologie und Medizin beigetragen. Bekanntheit in den Medien erreichte die Bioinformatik in erster Linie 2001 mit ihrem wesentlichen Beitrag zur Sequenzierung des menschlichen Genoms.

Bioinformatik ist ein weit gefächertes Forschungsgebiet, sowohl was die Problemstellungen als auch was die angewandten Methoden angeht. Wesentliche Gebiete der Bioinformatik sind die Verwaltung und Integration biologischer Daten. Da Bioinformatik unentbehrlich ist, um Daten in großem Maßstab zu analysieren, bildet sie einen wesentlichen Pfeiler der Systembiologie.

Der Bioinformatik wird im englischen Sprachraum oft die computational biology gegenübergestellt, die einen weiteren Bereich als die klassische Bioinformatik abdeckt, meist benutzt man beide Begriffe jedoch synonym.

2.5.1 Datenverwaltung

Die rasch wachsende Menge biologischer Daten wie z.B. DNA- und Proteinsequenzen, 3D-Proteinstrukturen oder Interaktionen biologischer Moleküle stellen besondere Anforderungen an die Handhabung dieser Daten. Ein wichtiges Problem der Bioinformatik besteht daher in der Datenaufbereitung und Speicherung in geeignet indizierten und verknüpften biologischen Datenbanken. Die Vorteile liegen dabei in der einheitlichen Struktur, der leichteren Durchsuchbarkeit und der Automatisierbarkeit von Analysen durch Software.

Der Umfang der einzelnen Datenbanken wächst teilweise exponentiell [8]. Auch die Anzahl einschlägiger Datenbanken wächst beständig (über 350 weltweit) [7]. Bei der Suche nach relevanten Informationen werden daher häufig Bioinformatik-Meta-Suchmaschinen (Bioinformatik-Harvester, Entrez, EBI SRS) benutzt.

Die Vielfalt an weltweit verfügbaren Datenbanken führt oft zu redundanter und damit fehleranfälliger Datenhaltung, zumal DNA-Sequenzen teils in Fragmenten, teils in vollständig assemblierten Genomen vorliegen. Idealerweise würde die Speicherung von Genom- und Proteindaten eine Rekonstruktion der Regelwerke eines gesamten Organismus erlauben. An der dazu erforderlichen Abbildung von identifizierten Proteinen auf die sie codierenden Gene und umgekehrt, an den Verknüpfungen untereinander zur Repräsentation ihrer Wechselwirkungen sowie an der Zuordnung von Proteinen zu metabolischen und regulatorischen Pfaden wird intensiv gearbeitet.

Eine weitere Aufgabe bei der Datenintegration ist das Erstellen von kontrollierten Vokabularien und Ontologien, die eine Zuordnung von Funktionsbezeichnungen quer durch alle Ebenen ermöglichen. Das Gene Ontology-Konsortium (GO) versucht derzeit, eine konsistente Nomenklatur für die molekulare Funktion, den biologischen Prozess und die Zelllokalisation von Gen-Produkten zu erstellen.

2.5.2 Workflows

Ein Arbeitsablauf (engl. *workflow*) ist eine definierte Abfolge von Aktivitäten in einem Arbeitssystem einer Organisation. Dabei werden verschiedene Komplexitäten nach der Granularität ihrer Beschreibung unterschieden. Meist wird ein Arbeitsablauf als Typ im Modell entworfen und zur Ausführung als Instanz mit aktuellen Daten versorgt.

- Ein *Arbeitsablauf* ist eine inhaltlich abgeschlossene, zeitlich und sachlogisch zusammenhängende Kette von Funktionen oder Diensten, die zur Bearbeitung einer betriebswirtschaftlich relevanten Entität notwendig sind und deren Übergänge beispielsweise von einem Informationssystem versorgt oder auch gesteuert werden.

- Ein *Arbeitsauftrag* (engl. *job*) ist ein Abschnitt eines Arbeitsablaufs mit einem oder über mehreren Arbeitsschritten oder Verrichtungen (engl. *task*) eines Auftrags. Dabei wird jeder Arbeitsschritt von einem Akteur (engl. *worker* oder *actor*) an einem Arbeitsplatz (engl. *work station*) und mit Ressourcen allein oder kooperativ erledigt.

- Ein *Arbeitsschritt* (engl. *task*) ist eine einzelne geschlossene Verrichtung innerhalb einer Kette solcher Verrichtungen oder in einem Arbeitsauftrag, die einzeln beschrieben ist, aber meist im Zusammenhang eines Auftrags beauftragt und von einem Akteur allein oder kooperativ erledigt wird.

Der Arbeitsablauf betont dabei die operational-technische Sicht auf die konkreten Prozesse mit zugehörigen Ressourcen, während der in der Definition nahestehende Geschäftsprozess die Abstraktion mit betriebswirtschaftlichen Faktoren wie Kosten und Erlösen betont. Dabei kann ein Workflow-Management-System den Ablauf in mehrfachem Kontext unterstützen, z.B. die notwendigen Daten bereitstellen. Internationale industrielle Gremien wie die *Workflow Management Coalition* (*WfMC*) und *Object Management Group* (*OMG*) haben dazu Standards wie *Business Process Analytics Format* (*BPAF*), *Business Process Model and Notation* (*BPMN*) und *Business Process Modeling Language* (*BPML*) entwickelt.

Die Ziele der Arbeitsablaufsteuerung sind unter anderen:

- Eine klare Vorgabe
- Vermeidung von Wartezeiten
- Vermeidung von temporaler oder modaler Überlastung
- Zuordnung von Ressourcen, wenn sie benötigt werden
- Konstante Belastung der beteiligten Arbeitnehmer
- Konstanter Ausstoß an Produkten
- Mögliche (Teil-) Automatisierung der Ausführung
- Weniger eine Dokumentation für eine Organisation oder deren Mitarbeiter

In Abgrenzung vom Geschäftsprozess wird dabei detailliert auf die operative Ebene eingegangen, indem die Arbeitsablauf-Beschreibung (Workflow-Definition) den Prozess in Bestandteile gliedert, die auf IT-Ebene eindeutig sind und in deterministischer Beziehung zueinander stehen.

Arbeitsablaufmodelle sollen helfen, die Unterstützung durch verschiedene Anwendungen und Datenbanken zu nutzen und die Arbeitsabläufe der Organisation sicherzustellen. Hierzu wird ein Workflow-Management-System eingesetzt: Das ist ein Softwaresystem, das den Entwurf und Anwendung von Modellen für Workflows ermöglicht, indem es die Workflow-Instanzen nach einem vorgegebenen, in der Applikation abgebildeten Schema steuert und dazu benötigte Daten bereitstellt oder weitere Applikationen startet.

2.6 Verwendung der Plugins

In diesem Abschnitt soll kurz erklärt werden, wie die von der PG erstellten Bausteine genutzt werden können.
Voraussetzung für die Nutzung ist die Verwendung von Java 8 und Maven (Version <=3.0.5 oder >=3.2.3), zusätzlich wird Wildfly in der Version 9 benötigt (mögliche höhere Versionen von Wildfly standen zum Zeitpunkt der Erstellung dieses Berichts noch nicht zur Verfügung). Außerdem muss PostgreSQL eingerichtet werden. Zum Beginn der PG war PostgreSQL in Version 9.3 aktuell, Version 9.4 kann ebenfalls verwendet werden.

Wildfly

Der Wildfly Application Server wird einfach von `http://wildfly.org/downloads/` heruntergeladen und entpackt.
Zusätzlich muss dem Wildfly noch ein Modul zur Verbindung mit PostgreSQL hinzugefügt werden. Dazu wird unter `http://jdbc.postgresql.org/download.html` das jdbc-jar-File in der aktuellsten Version heruntergeladen. Bei uns fand das JDBC „9.3-1102 JDBC 41“ Verwendung. Im Wildfly-Hauptordner, ab jetzt unter ${WILDFLY_HOME} geführt, muss der Ordner ${WILDFLY_HOME}/modules/system/layers/base/org/postgresql/main/ erstellt werden und das zuvor heruntergeladene jar-File in diesen Ordner kopiert werden. Anschließend wird in dem gerade erstellten Ordner die Datei *module.xml* erstellt. In diese Datei wird folgender Inhalt kopiert:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0"
   name="org.postgresql">
    <resources>
        <resource-root
           path="postgresql-9.3-1102.jdbc41.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

Damit ist die Vorbereitung des Wildfly Servers abgeschlossen.

Einrichtung PostgreSQL

Neben dem Wildfly muss PostgreSQL auf dem System installiert werden. PostgreSQL wird als Datenbank für die DyWA verwendet. Um die PostgreSQL-Datenbank im Zusammenspiel mit der DyWA zu verwenden, wird in PostgreSQL ein Nutzer *sa*

mit dem Passwort *sa* und eine Datenbank mit dem Namen *dywa* angelegt. Damit wären die grundlegenden Voraussetzungen zur Verwendung der PostgreSQL-Datenbank durch die DyWA geschaffen.

DyWA und PG-Bausteine

Ausgehend von einer Wildfly-Installation ohne Deployments (War-Files unter ${WILDFLY_HOME}/standalone/deployments/ ggf. löschen) und einer leeren *dywa*-Datenbank kann nun mit der Einrichtung der DyWA und der von der PG erstellten Bausteine begonnen werden.
Zuerst muss das Git-Repository der PG zur Verfügung stehen. Dann kann in dem Verzeichnis, welches das Repository enthält, mit der Einrichtung der Bausteine begonnen werden. Zuerst wird die DyWA installiert. Dazu muss unter *dywa-config/* mvn clean install -Dwildfly.path="<path to wildfly>" aufgerufen werden. Hierbei ist <path to wildfly> durch den Pfad zum Ordner mit dem Wildfly-Server zu ersetzen. Diese Ersetzung ist in späteren Schritten ebenfalls durchzuführen.
Ist dieser Aufruf mit SUCCESS erfolgreich beendet, so müsste sich nun in ${WILDFLY_HOME}/standalone/deployments/ die Dateien *dywa.war* und *xadisk.rar* befinden.
Nun wird zum ersten Mal der Wildfly Server gestartet. Dazu in ${WILDFLY_HOME}/bin/ *standalone.sh* bzw. *standalone.bat*, je nach Betriebssystem ausführen. Beim ersten Start der DyWA werden die Tabellen von der DyWA angelegt. Die DyWA sollte unter `http://localhost:8080/dywa/` nach erfolgreichem Start des Wildfly erreichbar sein. Der Server kann vor der Installation der weiteren PG-Bausteine erst einmal wieder gestoppt werden.
Im Anschluss daran werden die Bausteine der PG installiert. Dazu wird jeweils mvn clean install -Dwildfly.path="<path to wildfly>" in den folgenden Verzeichnissen aufgerufen:

1. dywa-common
2. dywa-logic
3. dywa-search
4. dywa-security
5. dywa-gui

Einzig die Reihenfolge von GUI und Security kann hierbei beliebig vertauscht werden. Die Reihenfolge der anderen Aufrufe ist auf Grund von Abhängigkeiten in dieser Reihenfolge festgelegt.
Nach erfolgreicher Installation kann der Wildfly Server erneut gestartet werden und enthält nun die Bausteine der PG als Webplugins.
Nach der Modellierung von Typen in der DyWA können diese für Modellierungszwecke im jABC bereit gestellt werden. Zugleich werden auch bereits modellierte Suchanfragen für die Applikation vorbereitet. Dazu muss unter dywa-testapp mvn

clean install -prepareModeling aufgerufen werden.
Nach der Prozessmodellierung im jABC folgt der Aufruf mvn clean install -Pfinish Modeling -Dwildfly.path="<path to wildfly>" unter *dywa-testapp/*. Sobald die Securitymodellierung und die GUI-Modellierung in den DyWA-Plugins abgeschlossen ist, wird der Bau der Testapp durch den Aufruf mvn clean install -PfinishWebapp -Dwildfly.path="<path to wildfly>" unter *dywa-testapp/* abgeschlossen.
Anschließend steht die fertige Applikation auf dem Wildfly-Server zur Ausführung bereit.

Kapitel 3

Zielsetzung und Vorgehensweise

Dieses Kapitel gibt einen Überblick über die Zielsetzung der Projektgruppe in Hinblick auf die entwickelte Anwendung sowie über die erforderlichen Bausteine, mit denen diese Ziele erreicht werden.

3.1 Zielsetzung

Die zu entwickelnde Webanwendung im Bereich der Biomedizin zur Erfassung und Verarbeitung soll neben dem modellierten Datenmodell der Domäne und den modellierten Prozessen auch auf einer modellierten Weboberfläche aufbauen. Zudem müssen Funktionen wie die Überprüfung von Zugriffskontrollen integriert werden. Dazu ist das vorhandene Framework, bestehend aus DyWA und jABC, erweitert worden. Die Erweiterungen ergänzen die DyWA in Form von Web-Plugins. Es gibt vier Web-Plugins, eins von jeder Untergruppe der PG, konkret also ein Web-Plugin für die Modellierung von Suchen, ein Plugin zur Beschreibung von Security-Anforderungen, ein Plugin zur Untergruppe „Logic“ und ein Plugin zur Modellierung der Weboberfläche.

Die Einsatzmöglichkeiten der neuen Bausteine des Frameworks werden anhand einer Beispielanwendung aus dem Bereich der Biomedizin demonstriert. Dazu werden mehrere Abläufe aus der Biomedizin in Prozessmodellen erfasst, die auf einem zuvor erstellten Domänenmodell arbeiten. Neben der Erfassung von Daten mit Hilfe von Prozessen steht dabei insbesondere die Verarbeitung dieser Daten im Vordergrund. Beispielsweise wird die Berechnung von Statistiken und Korrelationen aus den erfassten Daten demonstriert. Hinzu kommt die Modellierung einer zu der Anwendung passenden Weboberfläche. Näheres dazu findet sich in Abschnitt 3.3.

Durch die Erweiterung des Frameworks zur Entwicklung domänenspezifischer Anwendungen ändert sich der in Abbildung 2.1 dargestellte Ablauf. Der neue Ablauf ist in Abbildung 3.1 dargestellt. Nach der Modellierung der Domäne mit Hilfe der DyWA und vordefinierter Suchen auf den Typen der DyWA erfolgt der Export der Datentypen für die Prozesse. Im jABC werden die Prozesse für die Geschäftslogik sowie die Prozesse für Security-Überprüfungen unter Nutzung der domänenspezifischen Datentypen modelliert. Im Gegensatz zum ursprünglichen Framework erfolgt nun der Export der Prozesse, um diese im GUI-Builder neben den Datentypen und

Suchen nutzen zu können. Im GUI-Builder wird die Weboberfläche der domänenspezifischen Anwendung (wie z. B. Masken für die Anzeige von Daten oder das Ausführen für Prozesse) modelliert. Hinzu kommt die Modellierung von Zugriffskontrollen für Prozessmodelle und Bestandteile der GUI der Webanwendung. In einem finalen Generierungsschritt wird dann ein Prototyp der domänenspezifischen Anwendung mit Weboberfläche und integrierten Zugriffskontrollen generiert.

Durch die Verwendung bestehender und neuer Codegeneratoren bleibt dabei die Möglichkeit bestehen, zu jedem Zeitpunkt der Anwendungsentwicklung Teile der Anwendung wie z. B. die modellierten Prozesse zu verändern oder zu erweitern und anschließend die GUI für die veränderten Prozesse zu modellieren. Somit können domänenspezifische Anwendungen weiterhin über mehrere Iterationsschritte aufbauend auf einem elementaren Basiszustand der Anwendung hinweg entwickelt werden.

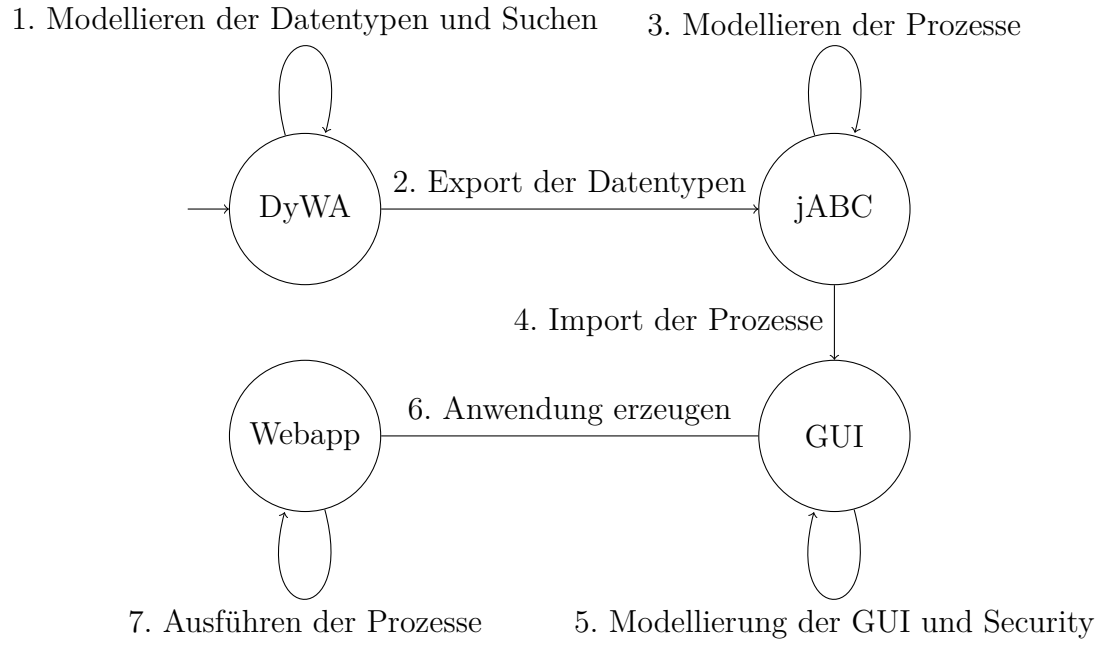

Abbildung 3.1: Erweitertes Framework zur Modellierung von Anwendungen

3.2 Framework-Komponenten zum Erreichen der Ziele

Für das Erreichen des gesetzten Ziels werden mehrere neue Bausteine benötigt, die in das vorhandene Framework integriert werden. Diese Bausteine sind in der folgenden Auflistung aufgeführt:

- Erweiterung der Modellierungsmöglichkeiten in jABC-Prozessmodellen
- Bereitstellung von Suchfunktionen für den Anwender sowie die interne Verwendung

- Schaffung einer Oberfläche zur Modellierung der domänenspezifischen Weboberfläche und Erstellen von Codegeneratoren für die modellierte GUI
- Integration von Zugriffskontrollmechanismen in die domänenspezifische Anwendung

Durch die benötigten Bausteine ergibt die Aufteilung der Projektgruppenmitglieder in die vier Kleingruppen „Logic“, „Search“, „GUI“ und „Security“. Die Details zu den Anforderungen sowie die Realisierungen der einzelnen Bausteine befinden sich in den jeweiligen Abschnitten.

3.3 Beispielanwendung

Zu Beginn der Projektgruppe war das Ziel, dass nach dem PG-Zeitraum eine Beispielanwendung mit der von der PG erarbeiteten Software modelliert werden kann. Unter Berücksichtigung des Anwendungsbereichs der prozessorientierten Verarbeitung biomedizinischer Daten hat sich die PG einige Anwendungsfälle überlegt, die die erarbeiteten Funktionen und Möglichkeiten nutzen und in ihrer Anwendung demonstrieren können. Die Beispielanwendung soll möglichst die Nutzung aller Funktionen abbilden, welche im Rahmen der PG erarbeitet worden sind. Dazu werden mehrere Abläufe aus der Biomedizin in Prozessmodellen erfasst, die auf einem zuvor erstellten Domänenmodell arbeiten.

Kapitel 4

Logic Builder

Wir gehen in diesem Kapitel auf die Implementierung von Bausteinen für erweiterte Funktionen in Prozessmodellen ein, die bei der Prozessmodellierung eingesetzt werden können. Dafür analysieren wir zunächst die Anforderungen und gehen danach darauf ein, welche Features wir bisher realisiert haben, wie die internen Strukturen und Abläufe dieser Features umgesetzt sind und geben einen Ausblick auf Features, welche nach Abschluss der PG noch implementiert werden können.

4.1 Anforderungsanalyse

Im Bereich der Biomedizin werden oft umfangreiche Analysen und Studien durchgeführt, um neue Erkenntnisse über Ursachen für Krankheiten oder Symptome zu erhalten. Hierbei kann es sich um verschiedene Arten der Erhebung von Daten handeln, wie zum Beispiel

- Fragebögen,
- Messreihen,
- Studien,
- Statistische Analysen und Auswertungen sowie
- Experimente.

Um die Erfassung von Daten mit Hilfe von Prozessmodellen zu ermöglichen und zu vereinfachen, werden neben den bestehenden Möglichkeiten von jABC-Prozessen weitere Features benötigt. Bei den Erweiterungen stehen die Erfassung von Daten mit Prozessen und das Erzeugen von Statistiken mit Hilfe von Prozessen im Vordergrund. Bisher fehlen Möglichkeiten in Prozessmodellen, um beispielsweise Messreihen über einen längeren Zeitraum hinweg zu erfassen.

Im Folgenden werden die Anforderungen aus dem Bereich der Biomedizin, die die Prozessmodelle betreffen, näher vorgestellt. Dabei werden die Anforderungen anhand verschiedener Anwendungsfälle kategorisiert.

4.1.1 Erweiterte SIB-Bibliotheken für Prozessmodelle

Wie bereits in Kapitel 2 beschrieben, stellen SIBs die zentrale Komponente in Form von Aktivitäten dar, aus denen ein Prozessmodell besteht. Zur Modellierung von Prozessen aus dem Bereich der Biomedizin werden Bibliotheken von SIBs benötigt, mit denen einige der in dem Anwendungsbereich benötigten elementaren Funktionen zur Verfügung gestellt werden. Aus den spezifischen Anforderungen der Biomedizin ergibt sich die Notwendigkeit von SIBs, welche u. a. die folgenden Funktionen realisieren:

- Erzeugung und Verarbeitung von Excel-Dokumenten,
- Versand von E-Mails und
- Verwendung von statistischen Funkionen für die Berechnung von Statistiken und Korrelationen sowie das Erzeugen grafischer Darstellungen für Statistiken

Um eine möglichst einfache Modellierung der Prozesse mithilfe der neuen SIBs in einer domänenspezifischen Anwendung zu ermöglichen, werden diese Funktionen als vorkonfigurierte SIBs zur Verfügung gestellt.

4.1.2 Verwendung von Timern in Prozessmodellen

Im Umfeld der Biomedizin werden Messreihen in zeitlichen Intervallen durchgeführt oder eine zuvor entnommene Gewebeprobe wird erst nach einer bestimmten Zeitspanne untersucht. Solche Beispiele zeigen, dass wir den Umgang mit Zeitspannen innerhalb von Prozessen benötigen. Dabei werden die Zeitspannen in Prozessen durch Timer erfasst. Für Timer können die folgenden Anwendungsfälle unterschieden werden:

- **Timer für den Start von Prozessen**: Es soll möglich sein, Prozesse aus der GUI zeitverzögert oder periodisch zu starten. Somit werden u. a. Prozesse für periodisch generierte Berichte wie Abrechnungen unterstützt.
- **Timer zur Unterbrechung von Prozessen**: Eine weitere Anwendung von Timern ist die Überbrückung einer Zeitspanne innerhalb eines Prozesses, z. B. wenn eine Probe eingelagert und nach einer bestimmten Zeit analysiert wird. Dazu muss ein laufender Prozess unterbrochen und nach Ablauf eines Timers automatisch fortgesetzt werden können.
- **Start von Timern für andere Prozesse**: Es soll außerdem möglich sein, dass in Prozessen Timer für andere Prozesse gestartet werden. Somit kann der erste Anwendungsfall auch innerhalb von Prozessen modelliert werden.
- **Kombination von Timern mit anderen Aktivitäten**: Timer können als Deadlines mit anderen Aktivitäten in Prozessen kombiniert werden. Beispielsweise kann die Erfassung von Messdaten durch einen Benutzer, die nach einer bestimmten Zeitspanne (z. B. einer Woche) nicht erledigt wird, mit einem Timeout abgebrochen werden.

4.1.3 Benutzereingaben in laufenden Prozessinstanzen

Es ist bisher möglich, Benutzereingaben beim Start eines Prozesses als Startparameter mitzugeben und im Prozess zu verwenden. Die vorhandenen Modellierungsfeatures bieten allerdings nicht die Möglichkeit, Benutzereingaben während eines laufenden Prozesses nachzuliefern.

Wenn ein Prozess in der DyWA bzw. der domänenspezifischen Webanwendung gestartet wird und Benutzereingaben benötigt, erscheint zunächst ein Dialog, in dem diese Daten einzugeben sind. Erst nach dem Ausfüllen dieses Dialoges wird der Prozess gestartet. Danach können nach bisherigem Stand innerhalb der laufenden Prozessinstanz keine weiteren Benutzereingaben erfolgen oder manuelle Entscheidungen über die Fortsetzung der Prozessinstanz getroffen werden.

Falls schrittweise Daten in Prozessen eingegeben werden sollen, müssen bisher mehrere Prozesse manuell nacheinander ausgeführt werden, die jeweils mit der Ausgabe des vorherigen Prozesses weiterarbeiten und zusätzliche Parameter für die Benutzereingaben erhalten. Dies schränkt jedoch die Prozessmodellierung ein. Daher ist die Integration von Benutzereingaben innerhalb laufender Prozessinstanzen mit mehr Flexibilität bei der Prozessmodellierung verbunden.

Wir unterscheiden zwischen zwei Arten von Interaktionen zwischen Benutzern und Prozessen:

- Eingaben ohne Einfluss auf den Prozessverlauf: Darunter fällt die Eingabe einzelner Messungen im Rahmen einer Messreihe über mehrere Dialogfenster.
- Eingaben mit Einfluss auf den Prozessverlauf: Eingaben können Einfluss darauf haben, wie der Prozess weiter fortgesetzt wird. Ein Beispiel dafür ist ein Dialog, bei dem der User auswählen muss, welche Aktion als nächstes auszuführen ist. Denkbar ist auch, dass beim Eintragen von bestimmten Werten (z. B. Blutwerte oberhalb eines bestimmten Schwellwertes) automatisch ein neuer Dialog erscheint und zu weiteren Untersuchungen anleitet.

Zudem ist die Bündelung mehrerer Benutzerinteraktionen als *Wizards* hilfreich, sodass Anwender mit Hilfe eines Prozesses durch komplexere Folgen von Dateneingaben geführt werden.

4.1.4 Persistieren von Prozessparametern und Prozessinstanzen

Mit der Unterbrechung von Prozessen durch das Warten auf den Ablauf von Timern oder die Eingabe von Daten durch Benutzer stellt sich die Frage, wie mit diesen Prozessen umzugehen ist. Wenn unterbrochene Prozesse im System weiterlaufen, entsteht eine schlecht skalierbare Webanwendung und pausierte Prozesse verursachen Performanceprobleme, da sie Ressourcen verschwenden und blockieren. Somit soll der Status unterbrochener Prozesse gespeichert werden, sodass die laufenden Prozessinstanzen beendet werden können. Eine unterbrochene Prozessinstanz muss persistiert werden, damit die Ausführung nach Ablauf des Timers oder der erfolgten Dateneingabe des Benutzers mit dem gleichen Ausführungskontext an der gleichen

Stelle fortgesetzt werden kann. Beim zeitverzögerten, periodischen Start von Prozessen müssen lediglich die Benutzerdaten der Startparameter der Prozesse gespeichert werden.

In beiden Fällen sind Daten von Datentypen zu speichern, die von der DyWA unterstützt werden – also die unterstützten primitiven Datentypen sowie die domänenspezifischen Datentypen. Bei der Unterbrechung von Prozessen muss neben dem Ausführungskontext der Einsprungspunkt für die Fortsetzung der Prozessinstanz gespeichert werden.

4.1.5 Langlebige Prozesse

Für Prozesse, deren Ausführung längere Laufzeiten in Anspruch nehmen, bestehen mehrere Anforderungen. Darunter fällt die nebenläufige Ausführung dieser Prozesse, da der Start solcher Prozesse über die Webanwendung mit dem Warten auf eine Rückmeldung nicht benutzerfreundlich ist. Mit der Berechnung komplexer Statistiken auf großen Datenmengen und die Erstellung statistischer Auswertungen über Prozesse besteht ein Anwendungsfall langlebiger Prozesse im Bereich der Biomedizin.

In Verbindung mit Zugriffskontrollmechanismen sind zudem Interaktionen verschiedener Benutzer mit einer Prozessinstanz vorstellbar. Bei der Genehmigung eines Beschaffungsantrages in einem Forschungsinstitut wird z. B. der Antrag von einem Anwender gestellt und muss anschließend von einem zweiten Anwender genehmigt werden. Die Interaktionen verschiedener Benutzer in Prozessen verbinden die Ausführung der Prozesslogik mit der Zugriffskontrolle.

4.1.6 Vorbereitung von jABC für die Anwendungsdomäne

Mit der Bereitstellung vorkonfigurierter jABC-Projekte soll die Nutzung von jABC durch Benutzer aus dem Bereich der Biomedizin vereinfacht werden. Dazu kann u. a. ein Basissatz von Prozessen aus der Biomedizin bereitgestellt werden oder die Nutzung von vorkonfigurierten, auf die Anwendungsdomäne ausgerichtete SIBs ermöglicht werden.

Im folgenden Abschnitt wird die grobe Architektur der entwickelten Artefakte beschrieben, mit denen einige der aufgeführten Anforderungen umgesetzt werden.

4.2 Architektur

In diesem Abschnitt wird die grobe Architektur der Komponenten beschrieben, die sich aus den bisher umgesetzten Anforderungen der Anwendungsdomäne Biomedizin (siehe Abschnitt 4.1) ergeben haben. Die Details zu den einzelnen Komponenten sind in den folgenden Abschnitten zu finden.

Das Zusammenspiel der Komponenten für die Ausführung von Prozessen ist in Abbildung 4.1 dargestellt. Dabei visualisieren die Kanten die Kommunikation bzw. die Zugriffe der Komponenten untereinander. Die Komponenten sind teilweise direkt nutzbare Bausteine, teilweise werden diese aber auch erst durch vorhandene oder

erstellte Codegeneratoren beim Export der Prozessmodelle aus jABC generiert. Die

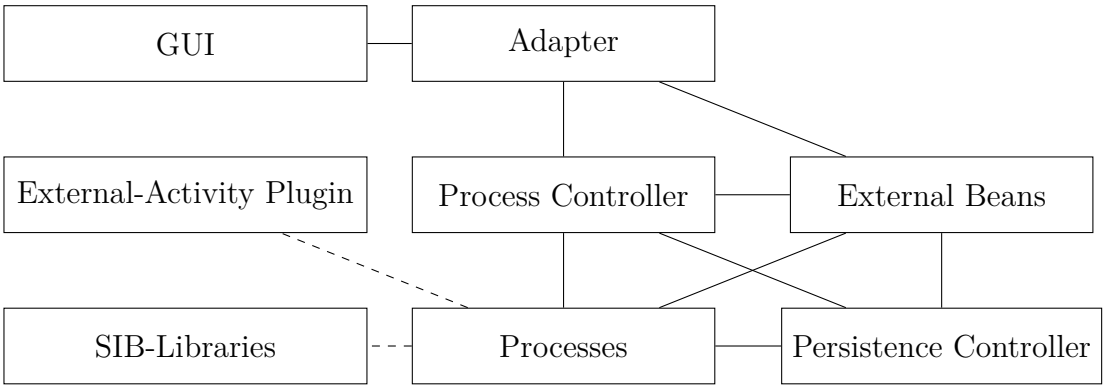

Abbildung 4.1: Übersicht der Komponenten für erweiterte Features in Prozessen

einzelnen Komponenten realisieren dabei die folgenden Funktionen:

GUI Weboberfläche der DyWA bzw. generierte Oberfläche der domänenspezifischen Anwendung, welche den Zugriff auf die Prozesse ermöglicht

Adapter Anbindung der generierten Controller für Prozesse an die GUI als eine Art Vermittler

Process Controller Verarbeitung und Ausführung von Prozessen in Verbindung mit Aufrufketten von Handlern (Chaining)

Processes Prozesse enthalten die Geschäftslogik der domänenspezifischen Anwendung und werden in jABC modelliert

SIB-Libraries Komponenten in Form von SIBs, die in Prozessen bei der Modellierung genutzt werden können und beispielsweise die Verarbeitung von Excel-Dokumenten ermöglichen

External-Activity Plugin jABC-Plugin für neue SIB-Typen (u. a. für Timer und Benutzerinteraktionen in Prozessen) und sogenannte SIB Cartridges, welche für die Behandlung dieser SIB-Typen bei der Codegenerierung verwendet werden

Persistence Controller Controller für den Zugriff auf Datenbank-Entitäten zur Persistierung unterbrochener Prozessinstanzen und Startparameter von Timer-gesteuerten Prozessen

External Beans Beans, die Funktionen wie Timer oder die asynchrone Ausführung von Prozessen bereitstellen

Mit den entwickelten Komponenten werden einige der analysierten Anforderungen erfüllt und die neu entwickelten Funktionen in die Weboberfläche der DyWA integriert. Beim Zugriff aus der domänenspezifischen Webanwendung heraus ist die

Adapter-Komponente nicht zwingend erforderlich, da eine generierte GUI direkt auf die generierten Prozess-Controller zugreifen kann.

Einige der vorgestellten Komponenten werden mit Hilfe von Codegeneratoren auf der Basis der modellierten Datentypen und Prozesse generiert, da das Datenmodell und die Prozess-API für die Komponenten benötigt werden oder sie auf andere Komponenten zugreifen, die generiert werden. Unter den generierten Bausteinen befinden sich die Implementierungen der Adapter, der komplette Baustein der Prozess Controller mit dem Chaining, die Controller für die Persistenz und die externen Beans. Dagegen sind die SIB-Bibliotheken, das ExternalActivity Plugin für jABC und die Datenbank-Entitäten für die Persistenz fest und werden nicht generiert.

Im folgenden Abschnitt beschreiben wir die erstellten SIB-Bibliotheken und gehen auf die neu entwickelten SIBs der einzelnen Anwendungsbereiche ein.

4.3 Erweiterte SIB-Bibliotheken für Prozesse

Wir haben in der Anforderungsanalyse gesehen, dass wir erweiterte SIB-Bibliotheken benötigen, um Funktionen wie den E-Mail-Versand oder die Verarbeitung von Excel-Dokumenten in Prozessen zu ermöglichen. Dafür nutzen wir verschiedene verfügbare Java-Frameworks und stellen die Funktionalitäten als Services in jABC bereit. Mit Hilfe eines Prozessgraphen für die Vorkonfiguration wird zudem die Nutzung dieser Services in Hinblick auf den Datenfluss vereinfacht. Die SIB-Bibliotheken werden als Maven-Artefakt zur Verfügung gestellt, sodass sie in jABC-Projekten als Abhängigkeit definiert und genutzt werden können.

4.3.1 Excel

Die Manipulation von Excel-Dokumenten steht im Zentrum der SIBs für die Verarbeitung von Excel-Dokumenten. Wir haben Services entwickelt, die elementare Funktionen zum Lesen und Schreiben von Excel-Dokumenten bereitstellen. Für die Realisierung der Services verwenden wir das Java-Framework Apache POI [1], welches eine umfassende Java-API für das Lesen und Schreiben von Microsoft Office Dokumenten zur Verfügung stellt.

Mit Hilfe der elementaren Service kann ein komplexer Ablauf wie das Erzeugen einer Patientenakte als Aneinanderreihung dieser Funktionen in einem Prozess modelliert werden. Bisher können Excel-Dokumente wie folgt in Prozessen genutzt werden:

- Erzeugen von neuen Excel-Dokumenten bzw. von Tabellenblättern innerhalb von Excel-Dokumenten
- Schreiben verschiedener Daten (einzelne Objekte, Listen sowie ein- und zweidimensionale Arrays) in Tabellenblätter
- Einlesen von Excel-Dokumenten in Java-Datenstrukturen

[1] `http://poi.apache.org/`

- Nutzung der beiden Excel-Formate *xls* und *xlsx*

Der letzte Aspekt der Auflistung wird mit zwei verschiedenen Java-Klassen umgesetzt, welche von einer gemeinsamen, abstrakten Java-Klasse erben. Diese abstrakte Klasse stellt die Services für die Excel-Dokumente zur Verfügung. Mit den beiden unterschiedlichen Subklassen erfolgt lediglich die Unterscheidung für das Erzeugen der Excel-Dokumente in den verschiedenen Formaten.

4.3.2 GNU R

Für statistische Auswertungen benutzen wir die Programmiersprache GNU R[2], welche Funktionen für statische Berechnungen und die Erzeugung statischer Auswertungen bereitstellt. Wir stellen daher Services zur Verfügung, welche auf diese Funktionen von R zurückgreifen. Dabei liegt der Fokus auf der Ausführung der Prozesse über die Weboberfläche der DyWA oder der domänenspezifischen Anwendung und weniger auf der direkten Ausführung der Prozesse in jABC mit Hilfe des Tracer-Plugins.

Für die Anbindung der Statistik-Programmiersprache R in Java-Programme existieren verschiedene frei verfügbare Frameworks, darunter

- JRI bzw. rJava (`http://www.rforge.net/rJava/`)
- Renjin (`http://www.renjin.org/`)
- Rserve (`http://www.rforge.net/Rserve/`)

Jedoch erfüllen nicht alle der aufgeführten Frameworks unsere Anforderungen. JRI bzw. rJava haben Schwierigkeiten mit dem Multi-Threading-Umfeld von Webservern, auf welches wir durch die Webanwendung allerdings angewiesen sind. Renjin überträgt R-Funktionen in die Java Virtual Machine (kurz JVM), unterstützt in der aktuellen Version aber noch keine grafischen Funktionen. Somit entfallen diese beiden Lösungen für unseren Anwendungsfall.

Dagegen deckt Rserve, eine Client-Server-basierte Lösung, alle benötigten Funktionen ab. Bei Rserve wird R über einen Server außerhalb der JVM ausgeführt. Mit einen Java-Client werden Anfragen in Form von R-Befehlen übermittelt und auf dem Server ausgewertet. Folglich muss neben dem Webserver auch der R-Server auf einer Maschine laufen, wenn Funktionen aus R in Prozessmodellen verwendet werden. Ein Vorteil dieser Lösung ist, dass der R-Server später getrennt auf anderen Maschinen laufen kann, sodass die Anwendung besser skaliert.

Wie bereits bei den SIBs für die Manipulation von Excel-Dokumenten, stellen wir auch in Bezug auf R lediglich einige Basisfunktionen zur Verfügung, z. B. für die Berechnung von Mittelwert oder Varianz. Darüber hinaus ist es mit dem aktuellen Stand möglich, R-Skripte über mehrere SIBs hintereinander auf dem R-Server auszuführen. Dies beinhaltet das Laden von R-Skripten, das Setzen von Werten für Parameter und die Rückgabe der Ergebnisse einzelner R-Befehle aus Basis des

[2] `http://www.r-project.org/`

geladenen Skriptes. Sobald genauere Anforderungen bezüglich statistischer Auswertungen in der Biomedizin bekannt sind, werden zusätzlich benötigten Funktionen implementiert.

4.3.3 E-Mail

Zum Versand von E-Mails aus Prozessen heraus werden Services bereitgestellt, mit denen E-Mails versendet werden können. Aus technischer Sicht erfolgt die Umsetzung auf der Basis der Java Mail API, einem Standard aus Java EE sowie dem Paket Apache Commons Email. Dieses stellt Funktionen bereit, die auf der Java Mail API aufbauen, aber stärker von technischen Details abstrahieren. Bei Verwendung von Services für den E-Mail-Versand werden zunächst die Login-Daten für einen Mailserver angegeben werden. Anschließend können aus dem Mailkonto heraus E-Mails (ggf. mit Anhängen) an mehrere Empfänger versendet werden.

4.3.4 Vorkonfiguration der Services als SIBs

Damit die Modellierung des Datenflusses bei der Verwendung der bereitgestellten Services weniger aufwändig ist, existiert ein Prozessgraph für die Vorkonfiguration [19]. Dieser Prozessgraph enthält insbesondere die Services für die Verarbeitung von Excel-Dokumenten und den Versand von E-Mails. Zusätzlich werden mit der Vorkonfiguration die entwickelten Services verschiedenen Packages zugeordnet, sodass sie leichter aufzufinden sind und die Nutzung durch den Anwender vereinfacht wird. Beispielsweise werden die Services für den schreibenden und den lesenden Zugriff auf Excel-Dokumente in zwei verschiedenen Packages gruppiert. Der Vorkonfigurationsgraph ist genauso wie die SIB-Bibliotheken in ein Maven-Artefakt eingebunden, sodass er in jABC-Projekte verwendet werden kann.

Im folgenden Abschnitt befassen wir uns mit der Umsetzung von Timern und Benutzerinteraktionen innerhalb von Prozessen.

4.4 Timer und Benutzerinteraktionen in Prozessen

In diesem Abschnitt stellen wir dar, wie Prozessunterbrechungen beispielsweise für Timer oder Benutzerinteraktionen realisiert werden. Der grundlegende Ansatz ist in beiden Fällen identisch, da die laufende Prozessinstanz unterbrochen und persistiert werden muss. Bei Timern wird die Prozessinstanz nach Ablauf des Timers automatisch fortgesetzt. Bei einer Benutzerinteraktion wird die Prozessinstanz fortgesetzt, wenn die Eingabe durch den Benutzer erfolgt oder die Deadline abläuft. Die Unterschiede solcher Prozessunterbrechungen liegen im Wesentlichen darin, durch welche internen oder vom Anwender angestoßenen Aktionen eine pausierte Prozessinstanz fortgesetzt wird.

Im Folgenden werden zwei mögliche Ansätze beschrieben, mit denen Unterbrechungen in Prozessmodellen umgesetzt werden können. Dabei benutzen wir die Kom-

ponenten, die wir in Abbildung 4.1 vorgestellt haben. Wir beschreiben ebenfalls, welche Nachteile der erste Ansatz beinhaltet und weshalb wir uns dafür entschieden haben, den zweiten Ansatz zu verwenden.

4.4.1 Realisierung der Unterbrechungen: Erster Ansatz

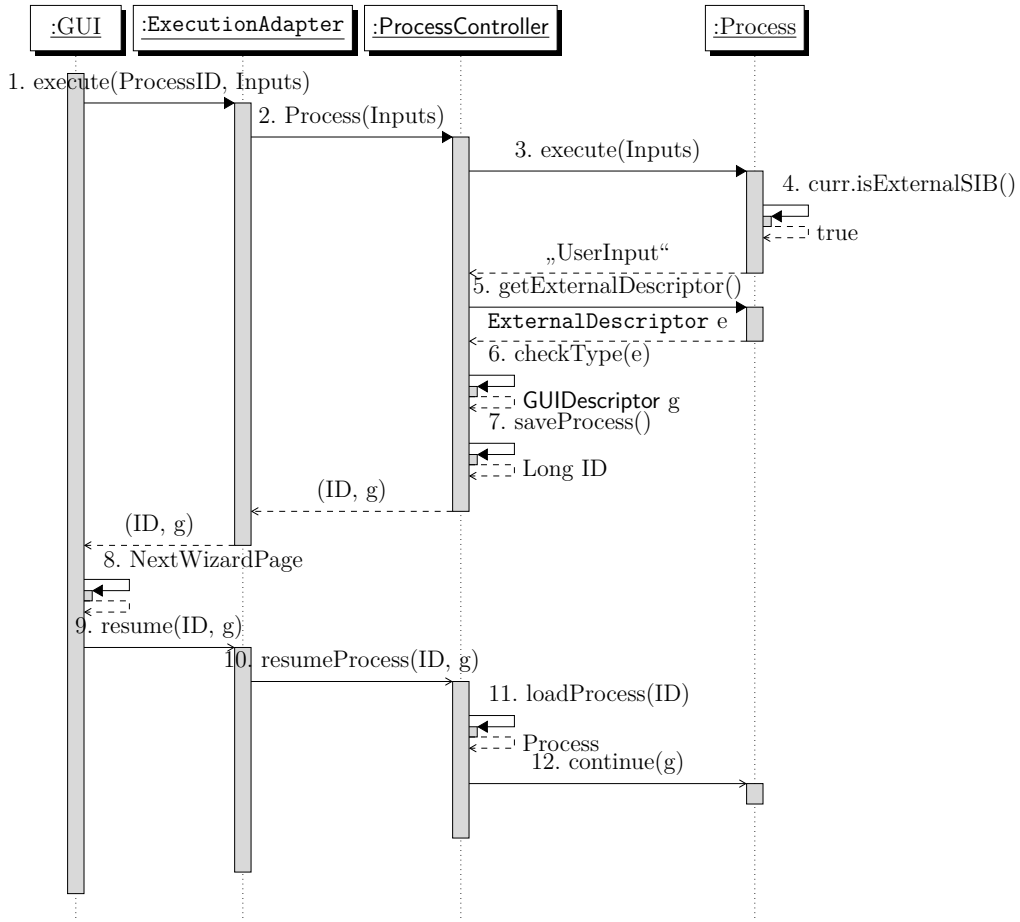

Abbildung 4.2: Erster Ansatz der Realisierung von Unterbrechungen in Prozessen

In Abbildung 4.2 wird der erste Ansatz graphisch dargestellt. Die modellierten Prozesse werden aus der DyWA bzw. der domänenspezifischen Anwendung heraus gestartet[3]. Dazu erfolgt von der GUI aus der Aufruf *execute* auf dem `ExecutionAdapter` unter der Angabe eines Prozess-Deskriptors und den Benutzereingaben für die Startparameter des Prozesses (siehe Kante 1). Die generierten ProcessController besitzen für jeden Prozess eine eigene Methode, wobei der Methodenname dem Prozessnamen entspricht. Der `ExecutionAdapter` ruft die zum Prozess gehörende Methode auf dem jeweiligen ProcessController auf (siehe Kante 2), welcher die Prozessausführung startet (siehe Kante 3). Im generierten Prozesscode wird in einer Schleife vom Start-SIB ausgehend der Reihe nach der Graph durchlaufen. Dabei wird für das aktuelle SIB jeweils überprüft, ob es ein SIB mit einer externen Funktion ist. Ein

[3]Der Start von Prozessen über (periodische) Timer erfolgt direkt über die ProcessController.

externes SIB ist ein SIB, das mit einer externen Aktion assoziiert ist, also z. B. auf den Ablauf eines Timer oder eine Benutzereingabe wartet. Ist das aktuelle SIB ein solches externes SIB (siehe Kante 4), signalisiert dies der Prozess dem ProcessController über die Rückgabe. Auf diese Rückgabe reagiert der ProcessController, indem er sich vom Prozess das `ExternalDescriptor`-Objekt holt (siehe Kanten 5 und 6). Das Deskriptor-Objekt enthält die benötigten Informationen für den Umgang mit der Unterbrechung. Also z. B. Information über den Timer oder über benötigte Eingaben eines Anwenders. Im letzteren Fall beschreibt der `ExternalDescriptor` (hier GUIDescriptor), welche Eingabefelder in der GUI anzuzeigen sind. Das Sequenzdiagramm beschreibt einen Fall, bei dem weitere Benutzereingaben erwartet werden.

Im nächsten Schritt wird der Prozess persistiert (siehe Kante 7). Dabei wird der Prozesszustand in der Datenbank abgespeichert und eine Referenz als eindeutige ID, zusammen mit dem GUIDescriptor über den `ExecutionAdapter` an die GUI zurückgegeben. Die GUI ermittelt aus dem Deskriptor, dass weitere Benutzereingaben benötigt werden und zeigt die dazugehörigen Eingabefelder an (siehe Kante 8). Sobald das Formular von Benutzer abgeschickt wird, setzt die GUI den Prozess über den `ExecutionAdapter` fort. Dazu enthält der `ExecutionAdapter` eine *resume*-Methode und die ProcessController *resumeProzess*-Methoden für die Prozesse (siehe Kanten 9 und 10). Der Unterschied zu den ursprünglichen *execute*-Methoden besteht darin, dass neben der ID des gespeicherten Prozesszustandes die Daten der externen Aktivität übergeben werden. Die gespeicherte Prozessinstanz wird im ProcessController aus der Datenbank geladen und an dem SIB fortgesetzt, an dem zuvor die Unterbrechung stattfand.

Nachteile Ein Nachteil bei der Verwendung der beschriebenen, allgemeinen Deskriptor-Objekte ist der Verlust der Typsicherheit. Die Typsicherheit geht verloren, wenn wir zum Fortsetzen eines Prozesses Deskriptor-Objekte übergeben. Dies ist insofern als schwerwiegendes Problem anzusehen, da jABC4 maßgeblich auf Typsicherheit setzt und der aus den modellierten Prozessen generierte Code ebenfalls diesem Paradigma folgt. Folglich führt die Umsetzung von Unterbrechungen nach dem vorgestellten Prinzip dazu, dass ein Teil der Typsicherheit aufgegeben wird. Daher wurde der Ansatz des Chainings für die Umsetzung gewählt, dessen Prinzip in Kapitel 2 bereits beschrieben wurde. Im Folgenden gehen wir kurz auf die Implementierung ein.

4.4.2 Chaining

Wie bereits im Grundlagenteil erklärt, stellt Chaining einen zentralen Mechanismus bei der Ausführung von Prozessen in der DyWA dar.

Implementierung von Chains Aus Sicht der Implementierung von Chains und Handlern sind mehrere Aspekte zu berücksichtigen: Für jede Chain existiert ein Handler in Form einer abstrakten Klasse, welche die Signatur der zu implementierenden Methoden vorgibt und die *forward*-Methode implementiert. Alle Handler für diese Chain müssen diesen abstrakten Handler erweitern. Außerdem müssen alle neu

hinzugefügten Handler als CDI Beans realisiert werden, sodass diese beim Aufbauen der Chain mit Hilfe einer BeanManager-Instanz gefunden und in die Chain integriert werden. Da der Code für die Prozessmodelle zur Ausführung in der DyWA generiert wird, werden die Implementierungen aller Handler ebenfalls generiert. Dies geschieht über Codegeneratoren in Form von Maven-Plugins, die Java- und Scala-Code kombinieren und nach dem Prinzip der Template-basierten Codegenerierung arbeiten. Das entwickelte Maven-Plugin für die Generierung von Handlern für Prozessunterbrechungen wird beim Import der jABC-Prozesse in die DyWA bzw. den GUI Builder aufgerufen.

Vorteile des Chainings Der Vorteil des Chainings liegt in der Flexibilität und Erweiterbarkeit. Es ist mit diesem Ansatz möglich, neue Funktionen bei der Prozessausführung umzusetzen, ohne die bisherigen Komponenten der DyWA oder der Chain verändern zu müssen. So kann zum Beispiel ein Security Handler in die Chain integriert werden, der vor bestimmten Aktionen die Rechte des aktuellen Anwenders überprüft und ggf. einen Aufruf gar nicht erst weiterleitet. Chaining bietet zudem die Möglichkeit, das Prinzip des *Separations of Concerns* umzusetzen, indem für jeden neuen Aspekt ein zusätzlicher Handler in die Chains eingefügt wird.

4.4.3 Zweiter Ansatz der Realisierung

In Abbildung 4.3 wird der zweite Ansatz und damit der aktuelle Stand der Realisierung von Unterbrechungen dargestellt. Aus der GUI heraus wird dabei der `ExecutionAdapter` über die Methode *execute* aufgerufen, wobei der Prozess und die Benutzereingaben als Parameter übergeben werden (Kante 1). Der `ExecutionAdapter` ruft eine Methode für den Prozess auf einem ProcessController auf, wobei wie im ersten Ansatz der Name des Prozesses dem Namen der Methode entspricht. Beim Erzeugen einer Instanz des SetupHandlers innerhalb der Methode des ProcessControllers wird auf dieser Instanz die *setup*-Methode aufgerufen. Danach startet der ProcessController die Chain, indem er die *handle*-Methode auf dem SetupHandler aufruft (Kante 5). Es folgt eine Kette von *forward*-Methodenaufrufen der Handler, die wir im Detail in Kapitel 2 beschrieben haben. Im letzten Kettenglied ruft der DefaultHandler die *execute*-Methode auf der Prozessinstanz auf (Kante 8), welche über die SIBs des Graphen iteriert. Dabei ruft der Prozess jeweils die *execute*-Methode der einzelnen (generierten) SIB-Container auf, wobei der Ausführungskontext übergeben wird und überprüft jeweils, ob es sich bei dem SIB um ein EndSIB handelt. Ein EndSIB ist ein Output-SIB des Prozesses oder ein Unterbrechungs-SIB, welches die Prozessausführung vorerst beendet. Im dargestellten Fall kommt ein Unterbrechungs-SIB für einen Timer vor, welches nicht den Branch „success“ sondern den Pseudo-Branch[4] „waitForTimerGraph“ zurückgibt (Kante 9). Da es sich um ein EndSIB handelt (Kante 10), übergibt der Prozess den Prozessstatus an den DefaultHandler. Dieser reicht das Ergebnis in Form eines ProcessResult-Objektes zurück, welches von dem

[4]Ein Pseudo-Branch existiert nicht innerhalb eines Prozessmodells, sondern wird nur in dem generierten Code des Prozesses für die Behandlung von Unterbrechungen eingeführt.

ExternalHandler überprüft wird. In dem ProcessResult-Objekt ist der Status der Prozessausführung als Strings und die Rückgabe des Success-Branches des Prozesses als Value enthalten (falls der Success-Branch erreicht wird). Falls der Success-Branch keine Rückgaben hat, hat der Value des ProcessResult den Typ Void, ansonsten den Typen des ersten Output-Parameters des Success-Branches. Der ExternalHandler persistiert bei einer Prozessunterbrechung die Prozessinstanz über die Controller der Persistenz (Kante 12), holt sich die Dauer der Unterbrechung von den Prozess (Kante 13) und übergibt die Daten an ein ResumeProcessBean (Kante 14). Anschließend wird das ProcessResult-Objekt über die Handler, den ProcessController und den `ExecutionAdapter` an die GUI zurückgegeben. Nachdem der Timer abgelaufen ist, ruft das ResumeProcessBean die *resume*-Methode auf dem ProcessController auf (Kante 15) und übergibt die Prozess-ID, mit dessen Hilfe der Prozess geladen (Kante 16) und fortgesetzt wird (Kante 17).

Wir haben in diesem Diagramm die Aufrufe innerhalb der Chain aus Platzgründen nur unvollständig angegeben. Neben der Auslassung einiger Methodenaufrufe für das erstmalige Ausführen des Prozesses ist insbesondere für die Fortsetzung des Prozesses die Verwendung der Chain komplett ausgelassen. Durch die verschiedenen Parameter von Methoden (Startparameter des Prozesses und Parameter zum Fortsetzen des Prozesses) besitzt jede Methode zum Ausführen oder Fortsetzen des Prozesses eine eigene Chain, sodass die Typsicherheit erhalten bleibt. Dies ist außerdem erforderlich, da sich die DefaultHandler der Chains voneinander unterscheiden, denn für jeden möglichen Einsprungspunkt in einer Prozessinstanz existiert eine eigene *resume*-Methode im generierten Code des Prozesses.

4.4.4 Neue SIB-Typen für Prozessunterbrechungen

Um Prozessunterbrechungen (z. B. Timer oder Benutzerinteraktion) bei der Modellierung von Prozessen in jABC umsetzen zu können, existieren neue, zusätzliche SIB-Typen. Damit diese neuen SIB-Typen bei der Codegenerierung aus Prozessmodellen gesondert behandelt werden, werden zusätzlich für die SIB-Typen SIB-Cartridges angelegt. Ein solches Cartridge sorgt dafür, dass der für den jeweiligen SIB-Typ spezifische Code in der generierten Klasse des Prozesses eingefügt wird und das SIB so eine andere Ausführungssemantik erhält.

Neue SIB-Typen Wie bereits angesprochen, unterscheiden wir bei Unterbrechungen zunächst in erster Linie zwischen Timern und Benutzerinteraktionen. Daher benötigen wir die folgenden neuen SIB-Typen:

- **TimerGraph**: SIB-Typ für eine Unterbrechung durch einen Timer für den aktuellen Prozess oder den Start von Timern für andere Prozesse
- **UserTask**: SIB-Typ für Benutzereingaben innerhalb von Prozessen

Die Aktivität eines Prozesses, die direkt nach einer Unterbrechung ausgeführt werden soll, wird als Unterprozess im Prozessgraphen eingefügt. Somit werden die beiden neuen SIB-Typen auf Unterprozesse angewendet. Das bedeutet praktisch, dass

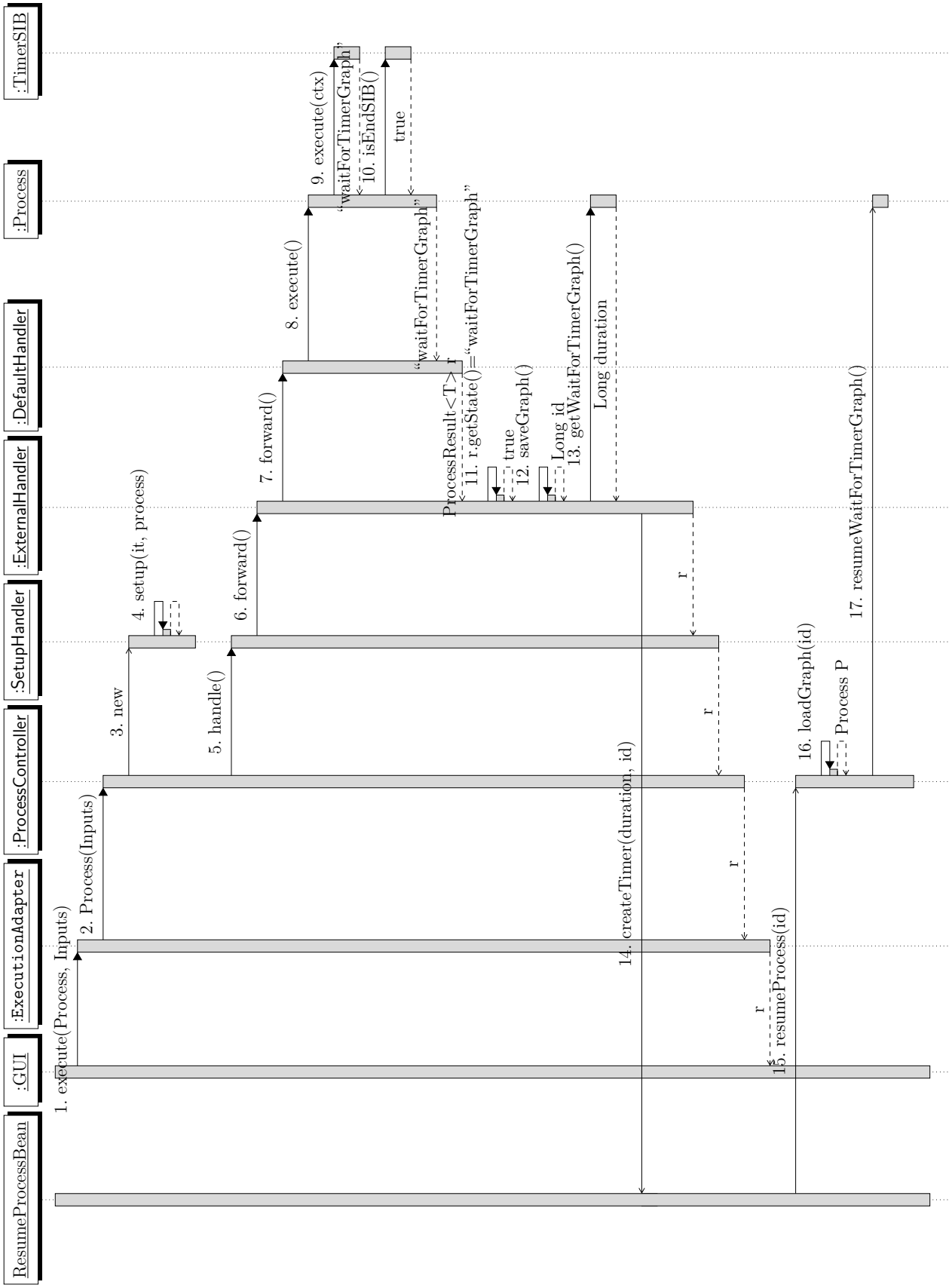

Abbildung 4.3: Zweiter Ansatz der Realisierung von Unterbrechungen in Prozessen

man einen vorhandenen Subgraphen zu einem TimerGraph bzw. einer UserTask machen kann.

Ein UserTask-SIB besitzt neben den Parametern des Subgraphen einen zusätzlichen Parameter, mit dem eine optionale Deadline für die Benutzerinteraktion modelliert werden kann. Die Angabe der Dauer der Deadline erfolgt in Sekunden, wobei ungültige Werte (Werte kleiner gleich null oder der undefinierte Wert null) zum Deaktivieren der Deadline führen. Im Falle, dass die Deadline vor der Eingabe der Benutzerdaten abläuft, wird der Kontrollfluss des Prozesses im Branch „timeout“ fortgesetzt. Falls der Untergraph bereits einen solchen Branch besitzt, wird dieser überschrieben und in jABC wird ein Warnhinweis angezeigt. Falls die Ausführung des Untergraphen (als generierter Code) den Branch „timeout“ liefert, wird eine Exception geworfen und die Prozessausführung direkt beendet. Die Daten, die von einem Benutzer im Rahmen der Benutzerinteraktion einzugeben sind, ergeben sich aus den Eingabeparametern des Subgraphen. Alle Eingabeparameter eines UserTask-SIBs (ohne die Deadlinedauer), die als dynamische Parameter nicht mit Kontextvariablen belegt sind, werden als Eingaben von dem Benutzer erwartet.

Ein TimerGraph-SIB besitzt neben den Parametern des Subprozesses zwei zusätzliche Eingabeparameter. Damit wird zum einen der Typ des Timers (also ob dieser sich auf das Warten innerhalb des aktuellen Prozesses oder das Starten eines Timers für einen anderem Prozess bezieht) und zum anderen die Dauer des Timers (Angabe in Sekunden) festgelegt. Diese zusätzlichen Parameter werden für das Anlegen des Timers ausgewertet und in dem generierten Code des SIBs entsprechend berücksichtigt.

jABC-Plugin Um die neu entwickelten SIB-Typen innerhalb von jABC verwenden zu können, haben wir ein jABC-Plugin entwickelt, welches als *ExternalActivity-Plugin* bezeichnet wird. Mithilfe dieses Plugins ist es möglich, die entsprechenden Drop Cartridges (UserTask und TimerGraph) über den Graph Browser in jABC auszuwählen und vorhandene Graphen per Drag & Drop auf die Modellierungsfläche zu ziehen. Dieser herein gezogene Graph wird zu einem Unterprozess und ein SIB des jeweiligen Typs.

Die neuen SIB-Typen sind durch ein Icon in einer der Ecken des SIBs in der Modellierungsfläche erkennbar. TimerGraph-SIBs sind an einem Icon in Form einer Uhr erkennbar, UserTask-SIBs an einem Icon eines Formulars.

Codegenerator-Bestandteile Neben den Java-Klassen existieren für die neuen SIB-Typen zusätzlich SIB-Cartidges für die Codegenerierung. Ein SIB-Cartridge enthält einen jABC-Prozess, welcher Java-Code liefert, der für die SIBs dieses SIB-Typs in einem Prozess eingefügt wird. Der aus dem SIB generierte Code enthält den SIB-Container für das SIB und ggf. weitere Bestandteile wie *resume*-Methoden für den generierten Prozess. Der Prozess des SIB-Cartridges erhält eine Repräsentation des zu verarbeitenden SIBs (SIB-Pojo) als Parameter und liefert einen String als Rückgabe, der Java-Code enthält. Ein solcher Prozess muss einen speziellen Interface-Graphen implementieren, um bei der Codegenerierung berücksichtigt zu

werden und wird anhand des Typs des Eingabeparameters vom Codegenerator für Prozesse auf SIBs des jeweiligen Typs angewendet.

Die Prozessmodelle der Cartridges werden zunächst mit Hilfe des Codegenerators für Prozessmodelle in ausführbaren Code übersetzt und anschließend als Bestandteil des gleichen Codegenerators auf die domänenspezifischen Prozesse angewendet. Damit wird auch für die Prozessmodelle der SIB-Cartridges das Prinzip des Bootstrappings angewendet.

4.4.5 Erweiterungen bei der Einführung neuer SIB-Typen

Für die Implementierung neuer SIB-Typen sind eine Reihe von Artefakten zu implementieren, die im Folgenden kurz vorgestellt werden. Dabei sind die Artefakte unabhängig von der Funktionalität des neuen SIB-Typs erforderlich.

- **Klassen des SIB-Typs**, die als Teil eines jABC-Plugins eingebunden werden. Dafür existieren jeweils Interfaces oder (abstrakte) Superklassen, die zu implementieren oder zu erweitern sind. Unter den Klassen des SIB-Typs sind die
  - eigentliche Klasse des SIB-Typs,
  - die Klasse des SIB-Pojos und
  - eine Klasse eines Drop Cartridges
- **SIB-Cartridge**, das ein Prozessmodell enthält, welches den Container des SIBs und eventuell zusätzliche Methoden innerhalb eines generierten Prozesses erzeugt.
- ggf. **Handler** für das Chaining bei Prozessaufrufen, die wie alle Handler generiert werden müssen.

Im nachfolgenden Abschnitt gehen wir auf die Nebenläufigkeit innerhalb von Prozessen ein.

4.5 Nebenläufigkeit innerhalb von Prozessen

Im Folgenden stellen wir dar, wie Teile eines Prozesses nebenläufig ausgeführt werden können. Es soll also möglich sein, einen jABC-Prozess so zu modellieren, dass an einem bestimmten Punkt in einem Prozess (Masterprozess) mehrere Unterprozesse (Threads) entstehen und parallel ausgeführt werden. Erst wenn all diese Threads erfolgreich beendet wurden, wird der übrige Teil des Masterprozesses weiter ausgeführt.

Wir erläutern die technische Umsetzung anhand der Abbildung 4.4, welche das Schema eines jABC-Graphen mit Nebenläufigkeit darstellt. Alles was vor dem Knoten *Fork* steht, wird wie bisher ausgeführt. Dort können also auch unter anderem Timer oder Benutzerinteraktionen vorkommen. Sobald das SIB *Fork* ausgeführt wird, wird der Kontext des Masterprozesses, also der Inhalt der Kontextvariablen in die

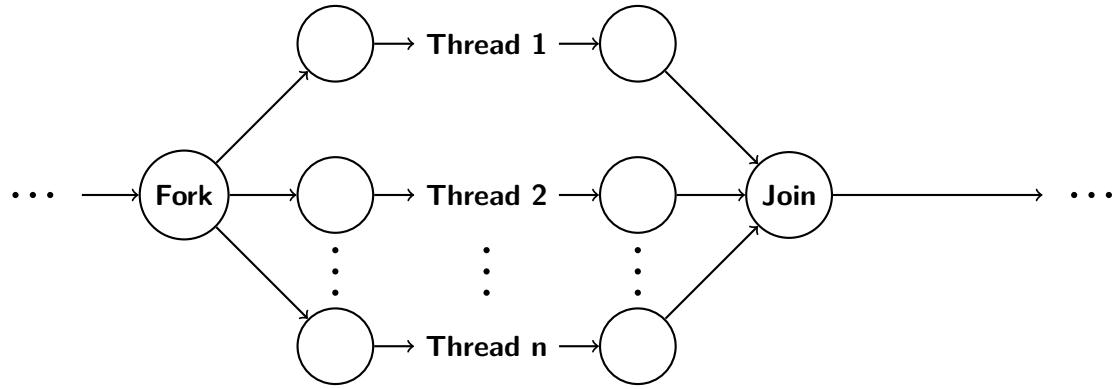

Abbildung 4.4: Schema eines nebenläufigen Prozesses

Datenbank abgespeichert und es werden n weitere Prozesse (Threads) gestartet, dessen Startknoten jeweils ein Folgeknoten von *Fork* ist und dessen Endknoten jeweils ein Vorgängerknoten des Knotens *Join* ist.

Java EE7 Concurrency bietet bereits eine Umsetzung von mehreren nebenläufigen Threads und könnte mit Request Scoped Beans realisiert werden. Allerdings scheitert dieser Ansatz daran, dass dabei neue Java Threads erzeugt werden, welche den ursprünglichen Scope nicht mehr beinhalten. Somit ist zum Beispiel die Datenbankverbindung in diesen Threads nicht mehr vorhanden. Stattdessen werden zur Umsetzung zwei statische Queues mit jeweils einer Message Driven Bean als Listener benutzt.

Technische Umsetzung Wenn der Knoten *Fork* ausgeführt wird, wird der Masterprozess persistiert und es wird eine Bean gestartet. Diese soll ihn wieder fortsetzen, sobald alle Prozess-Threads erfolgreich beendet sind. Außerdem wird eine JMS-Nachricht in eine Queue (fork-queue) geschickt, für die eine Message Driven Bean existiert, die bei Ankunft einer solchen Nachricht alle Threads startet. Sobald ein Thread erfolgreich beendet wurde, schickt dieser eine JMS-Nachricht an eine andere Queue (join-queue), für welche es ebenfalls einen Listener in Form einer Message Driven Bean gibt. Da bei der Generierung der Bean bereits der Prozess modelliert war, also festgelegt war, wie viele Threads es gibt, kann die Bean reagieren, sobald alle Threads eine Nachricht geschickt haben. Der Masterthread wird dann somit wieder geladen und mit dem gegebenenfalls veränderten Kontext ab dem *Join*-Knoten weiter ausgeführt.

Behebung potentieller Fehler In der Datenbank gibt es zu jedem laufenden Prozess eine Referenz auf den dazugehörigen Ausführungskontext. Im Regelfall ist ein Kontext also nur einem Prozess zugeordnet. In diesem Fall referenzieren allerdings neben dem Masterprozess nun auch alle n Threads in der Datenbank auf den selben Kontext. Dazu kommt, dass ein Prozess in jABC so modelliert werden kann, dass mehrere Threads die selben Kontextvariablen verändern. Dies kann dazu führen, dass Race Conditions entstehen, die bei dem verwendeten Optimistic Locking zu entsprechenden Fehlern führen. Liest ein Thread eine Variable, speichert er da-

bei das dazugehörige Versionsattribut, welches sich jedesmal verändert, wenn der Inhalt der Variable geschrieben wird. Will nun der selbe Thread den Inhalt dieser Variable verändern, wird zunächst das Versionsattribut überprüft. Hat sich dieses in der Zwischenzeit (zwischen Lesen und Schreiben) verändert, wird eine Exception geworfen. Das passiert genau dann, wenn ein anderer Thread in der Zwischenzeit die Variable und damit das Versionsattribut verändert hat. Diese Exception wird allerdings aufgefangen, indem der Thread, der diese Exception wirft, nochmal mit aktuellen Versionsnummern neugestartet wird. Das sorgt dafür, dass höchstens n mal ein Thread neugestartet werden muss, bevor alle Threads erfolgreich beendet sind. Somit wird diese Exception in jedem Fall intern behandelt ohne dass der Benutzer etwas davon bemerkt.

4.6 Nachrichten zwischen Prozessen

Ein weiteres umgesetztes Feature ist der Nachrichtenaustausch zwischen Prozessen. Damit können Prozesse an andere Prozesse Nachrichten verschicken oder auf Nachrichten warten. Die Nachrichten können dabei Daten enthalten oder lediglich als Trigger benutzt werden, um persistierte Prozesse aufzuwecken. Die verwendete Technologie für den Nachrichtenaustausch ist JMS, ein Feature von Java EE7. Nun wird zuerst erläutert, wie Nachrichten versendet und danach, wie sie empfangen werden können.

Nachrichten versenden Um innerhalb eines Prozesses eine Nachricht zu versenden, muss beim Modellieren kein Service Graph, sondern ein Interface Graph als DropCartridge *SendMessageGraph* verwendet werden. Der Interface Graph muss vorher erstellt werden, denn die Eingangsparameter des Interfaces bestimmen die Datentypen der zu übermittelnden Daten. Soll also zum Beispiel eine Nachricht verschickt werden, die einen Namen und ein Alter enthält, muss vorher ein Interface Graph mit den Eingangsparametertypen String und Integer erstellt werden. In jABC erscheint zusätzlich noch ein weiterer Parameter vom Typ String, welcher den sogenannten Filter darstellt. Nachrichten können nur an einen Prozess mit dem selben Filter verschickt werden, damit die Möglichkeit eines gezielten Nachrichtenaustausches besteht. Der Code eines *SendMessageGraph* wird so generiert, dass eine JMS Nachricht versendet wird, welche die übergebenen Daten enthält.

Nachrichten empfangen Das Empfangen von Nachrichten ist als Prozessunterbrechung realisiert. Ein DropCartridge vom Typ *ReceiveMessageGraph* wird auf einen Graph, also einen Unterprozess, angewendet, analog zu der Vorgehensweise bei Benutzereingaben oder Timern. Der Unterprozess muss das Interface realisieren, das zur Nachricht passt, welche empfangen werden soll. Sobald der Prozess bei diesem Knoten angekommen ist, wird persistiert, wie es bereits für die anderen Prozessunterbrechungen beschrieben wurde. Es existiert für jeden Graphen vom Typ *ReceiveMessageGraph* eine Bean, welche auf Nachrichten mit passendem Filter reagiert. Sobald einen solche Nachricht empfangen wurde, setzt die Bean den Prozess

fort und übergibt die Daten der Nachricht als Eingangsparameter. Sollten mehrere Prozesse mit passendem Filter (und realisiertem Interface) auf eine Nachricht warten, werden alle fortgesetzt. Sollte kein Prozess auf eine versendete Nachricht warten, wird diese nicht gespeichert, sondern verworfen.

Im nachfolgenden Abschnitt gehen wir auf die Realisierung der Persistierung von Prozessdaten ein.

4.7 Persistierung von Prozessdaten

Die Notwendigkeit der persistenten Zwischenspeicherung von Daten für die Ausführung von Prozessen taucht an verschiedenen Stellen der erweiterten Features in Prozessmodellen auf:

- Beim zeitlich verzögerten Start von Prozessen über das Eintreten externer Ereignisse (wie z. B. der Ablauf von Timern) müssen die Startparameter eines Prozesses bis zum tatsächlichen Start des Prozesses persistiert werden.

- Bei der Unterbrechung laufender Prozessinstanzen (wie z. B. für Userinteraktionen oder Timer in Prozessen) ist die Prozessinstanz für die spätere Fortsetzung zu persistieren. Dies betrifft insbesondere die Werte des Ausführungskontextes und die Möglichkeit der Wiederaufnahme des Prozesses.

Somit gibt es zwei wichtige Anwendungsfälle, in denen Daten von Prozessen zwischengespeichert werden müssen.

Umsetzung der Persistierung In beiden Anwendungsfällen basiert die Persistierung auf einem zentralen Grundsatz: Daten primitiver Datentypen wie Strings oder Integer und Listen primitiver Datentypen wie Listen von Strings werden direkt mit ihren jeweiligen Wert(en) gespeichert. Die Objekte domänenspezifischer Datentypen aus der DyWA dagegen werden lediglich mit Hilfe ihrer ID gespeichert und zum Zeitpunkt der Prozessausführung aus der DyWA geladen und verwendet. Damit werden im zweiten Anwendungsfall Änderungen an den domänenspezifischen Objekten verworfen, die vor der Persistierung nicht gesichert wurden. Bei der Speicherung von Prozessinstanzen können auch Prozessinstanzen, die in Kontextvariablen abgelegt sind, über ein rekursives Verfahren persistiert werden [19].

Mit der Entscheidung, die verwendeten DyWA-Objekte beim Start eines Prozesses nach einem Timerablauf oder bei der Fortsetzung des Prozesses z. B. nach einer Userinteraktion neu aus der DyWA zu laden, wird das Überschreiben von Daten mit veralteten Versionen verhindert. So ist es nicht möglich, dass durch fortgesetzte Prozesse die (manuellen) Änderungen an DyWA-Objekten überschrieben werden, die parallel zur Unterbrechung einer Prozessinstanz vorgenommen werden.

Einschränkungen Mit der umgesetzten Lösung können die Datentypen persistiert werden, die das Java-Interface *Serializable* implementieren, (Listen von) domänenspezifischen Objekten sind oder selbst wieder Prozesse (mit den genannten Ein-

schränkungen) sind. Alle weiteren Datentypen werden nicht gespeichert und müssen beim Fortsetzen von Prozessen anderweitig neu gesetzt werden.

Unterschiede Die beiden Anwendungsfälle für die Zwischenspeicherung von Daten für Prozesse unterscheiden sich darin, dass bei Timern vor Prozessen mit der verwendeten Technologie der EJB Timer die Startparameter des Prozesses in einem serialisierbaren Objekt vom Webserver (auf der Festplatte) gespeichert werden. Die Zwischenspeicherung unterbrochener Prozessinstanzen erfolgt dagegen über die Datenbank der DyWA. Dazu existieren folgende Datenbank-Entitäten (Abbildung 4.5): Für eine gespeicherte Prozessinstanz wird ein Objekt der Klasse ProcessState erzeugt. Der ProcessState besitzt eine Referenz zu einem Objekt der Klasse ProcessContext, welche für die verschiedenen Typen persistierbarer Kontextvariablen (serialisierbare Objekte, domänenspezifische Objekte, Listen domänenspezifischer Objekte und Prozessinstanzen) jeweils eine Map enthält. In der Map wird der Name der Kontextvariable und, dem bei der Unterbrechung vorliegenden, Wert der Variablen (bei DyWA-Objekten deren ID) abgelegt. Somit repräsentiert die Klasse ProcessContext den Ausführungskontext einer unterbrochenen Prozessinstanz zum Zeitpunkt der Unterbrechung. Zusätzlich kann in der gespeicherten Prozessinstanz ein Objekt der Klasse TimeoutEvent mit Deadline-Informationen sowie ein Objekt der Klasse MessageEvent für das Warten auf Nachrichten abgespeichert werden. Bei den Klassen TimeoutEvent und MessageEvent handelt es sich um keine echte Datenbank-Entitäten, sondern um Embedabbles, da auf diese Daten immer nur über die Prozessinstanz zugegriffen wird. Mit Hilfe der Informationen einer Entität des Typs ProcessState ist das Fortsetzen einer Prozessinstanz möglich, da neben dem Typ des Prozesses (in Form eines Class-Objektes) auch die Methode zum Fortsetzen des Prozesses abgelegt wird. Die Deadline einer Prozessinstanz enthält neben einem Identifier für den Timer den Namen einer weiteren *resume*-Methode des Prozesses.

Umsetzung der Persistierung Der Controller für die Persistierung unterbrochener Prozessinstanzen und Startparametern wird generiert, da Java-Klassen benötigt werden, die erst mit dem Export der domänenspezifischen Datenklassen generiert werden. Innerhalb des Controllers ist die Umwandlung der Startparameter bzw. des Ausführungskontextes in persistierbare Objekte und zurück als generischer Code implementiert. Also wird nicht für jedes Prozessmodell ein spezifischer Codeschnipsel generiert, sondern die Umwandlung erfolgt auf Basis der Reflection API und Überprüfungen der Klassen von Daten. Die Entscheidung für die generische Lösung liegt in der Einfachheit der Implementierung, da sie durch die geringe Anzahl unterstützter Datentypen sehr kompakt ist.

Wie bereits angesprochen, existieren u. a. Einschränkung bei der Persistierung von Prozessdaten. Im nächsten Abschnitt gehen wir auf alle Einschränkungen, die sich bei der Verwendung der Komponenten ergeben, ein.

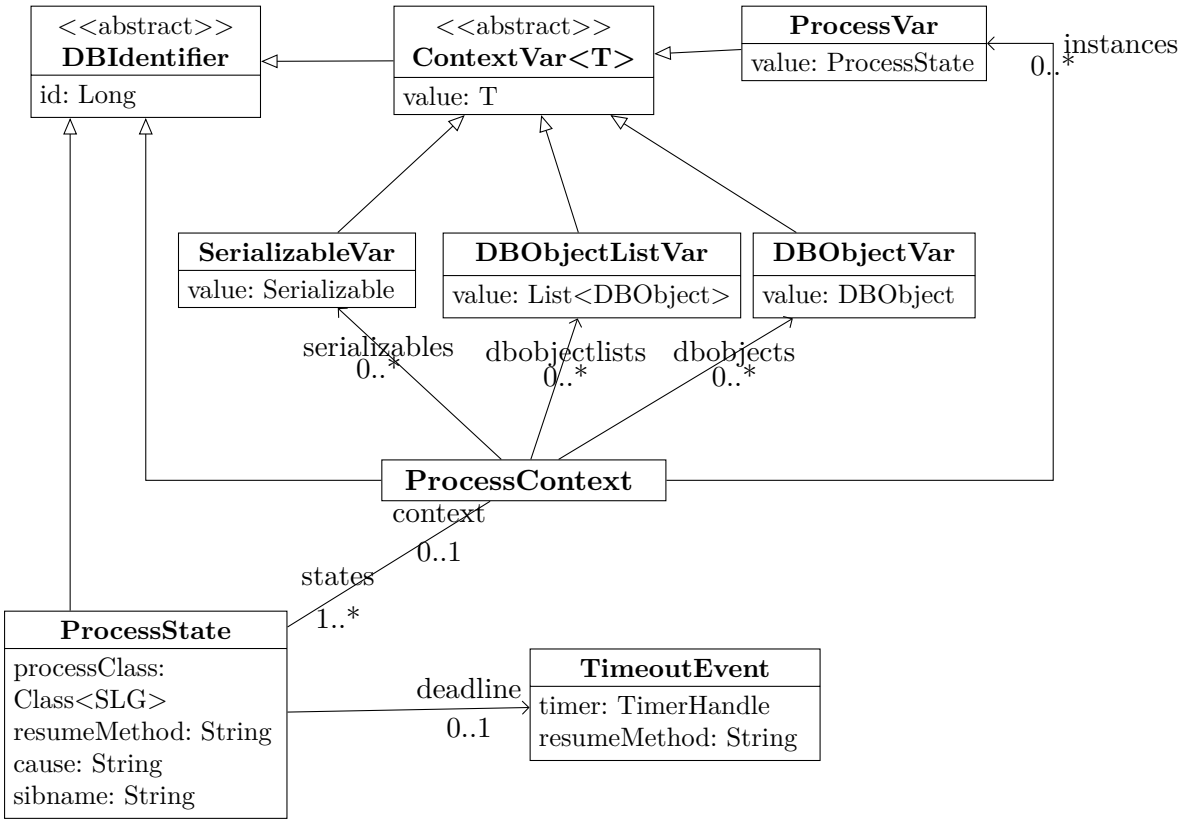

Abbildung 4.5: UML-Klassendiagramm der Entitäten für die Persistenz von Prozessen

4.8 Einschränkungen bei Verwendung der Komponenten

Bei der Verwendung der von uns realisierten Komponenten, die wir in diesem Kapitel vorgestellt haben, treten teilweise Beschränkungen auf. Dies betrifft zum einen die Persistierung der Prozessinstanzen bei Unterbrechungen und zum anderen die Nutzung der neu eingeführten SIB-Typen in Prozessmodellen.

Einschränkung bei der Persistierung Die Einschränkung bei der Persistierung der Prozessinstanzen liegt in der Beschränkung der Speicherung auf die in Abschnitt 4.7 beschriebenen Datentypen, wie z. B. primitive Datentypen oder domänenspezifischen Datentypen der DyWA. Alle andere Datentypen, die nicht in diese Kategorien fallen, können aktuell nicht gespeichert werden. Somit gehen zum Beispiel nicht-serialisierbare Beans, die in Prozessen injiziert werden, beim Fortsetzen von Prozessen nach Unterbrechungen verloren und müssen ggf. manuell neu gesetzt werden.

Einschränkung bei der Nutzung der SIB-Typen Auch die Nutzung der neu eingeführten SIB-Typen innerhalb von Prozessmodellen unterliegt einer Beschränkung. Aktuell können die neuen SIB-Typen, die eine Unterbrechung des Prozesses zur Folge haben, nur in Top-Level-Prozessen verwendet werden. Eine Verwendung innerhalb von Subprozessen ist hingegen nicht möglich.

Diese Einschränkung basiert einerseits auf der Verwendung der Pseudo-Branches für SIBs mit Unterbrechungen wie TimerGraph-SIBs. Ein Prozess, der einen Subprozess mit einem TimerGraph-SIB enthält, erhält bei der Ausführung den Pseudo-Branch des TimerGraph-SIBs zurück. Anschließend wird für diesen Pseudo-Branch ein Nachfolger gesucht, der jedoch nicht existiert, sodass die Prozessausführung mit dem Werfen einer Exception abgebrochen wird.

Andererseits ist die Beschränkung durch das Fortsetzen des Prozesses bedingt, da es bisher nicht möglich ist, Prozesse über mehrere Hierarchieebenen fortsetzen zu können. Ein Prozess besitzt bislang keine *resume*-Methoden, mit denen eine *resume*-Methode eines Subprozesses aufgerufen werden kann. Aktuell wird in den *resume*-Methoden lediglich die Logik der *execute*-Methode des SIB-Containers ausgeführt und anschließend die Prozessausführung über eine Schleife für die nachfolgenden SIBs durchgeführt.

Aus der Sicht der Persistenz von Prozessinstanzen ist die Verwendung von Unterbrechungen in Subprozessen durchaus möglich, da Prozessstacks mit den bekannten Einschränkungen gesichert werden können. Jedoch scheitert die Nutzung an dem aus Prozessen generierten Code für die Ausführung.

Im letzten Abschnitt fassen wir unseren jetzigen Entwicklungsstand zusammen und geben einen Ausblick auf weitere mögliche Features.

4.9 Zusammenfassung

Der innerhalb der Projektgruppe erreichte Entwicklungsstand erfüllt die vorgestellten Anforderungen. So ist es z. B. möglich, in jABC Unterbrechungen für Timer oder Benutzerinteraktionen zu modellieren. Außerdem können in Prozessen nebenläufige Ausführungsstränge realisiert werden und verschiedene Prozessinstanzen können mit Nachrichten miteinander kommunizieren, die als Trigger zum Fortsetzen oder für den Datenaustausch genutzt werden können. Mit Hilfe der erstellten Codegeneratoren können diese erweiterten Funktionen für Prozesse in einer domänenspezifischen Webanwendung genutzt werden oder durch Administratoren über das entwickelte DyWA Logic Plugin kontrolliert werden. Darüber hinaus wurden eigene SIB-Bibliotheken für verschiedene Services wie die Verarbeitung von Excel-Dokumenten oder die Berechnung von Statistiken mit GNU R implementiert.

4.10 Ausblick

Ziele für die weitere Entwicklung können einerseits darin bestehen, die aktuell existierenden Einschränkungen der Implementierung (wie die Beschränkung alternativer SIB-Semantiken auf Top-Level-Prozesse) zu erweitern oder ganz aufzuheben. Außerdem können noch weitere Features wie eine Batchausführung mehrerer Prozesse oder eine stärkere Kopplung von Business-Prozessen mit den Security-Funktionen der PG implementiert werden. Beispielsweise ist es gut vorstellbar, dass ein Security-Prozess für die Absicherung einer Benutzerinteraktion Zugriff auf den persistierten Ausführungskontext des unterbrochenen Prozesses erhält und so komplexere Zugriffskontrollen auf Basis der Daten des Ausführungskontextes ermöglicht werden. In Verbindung mit der GUI-Modellierung können für Prozesse noch Erweiterungen für die Nutzung mehrerer Benutzerinteraktionen als Wizard implementiert werden. Insbesondere ist dabei die Verbindung der einzelnen GUI-Formulare für die Benutzerinteraktionen zu erweitern.

Kapitel 5

GUI

Die GUI-Teilgruppe hat das Ziel, eine Web-Applikation zu entwickeln, mit der auf Basis der DyWA-Typen und der generierten Prozesse ein Frontend für eine domänenspezifische Anwendung modelliert werden kann (s. Abschnitt 5.3.1). Das erstellte domänenspezifische Frontend-Modell wird durch Anwendung eines Codegenerators zu einer eigenständigen Webanwendung (s. Abschnitt 5.3.3).

5.1 Anforderungsanalyse

Im folgenden Abschnitt wird die Aufgabe der GUI-Gruppe näher spezifiziert und anhand eines Use-Case-Szenarios beispielhaft erläutert.

5.1.1 Aufgabenbeschreibung

Zunächst wird der grobe Ablauf der Erstellung einer domänenspezifischen Anwendung beschrieben. In Anlehnung daran wird die konkrete Aufgabe der GUI-Gruppe erläutert.

Genereller Ablauf

Die in diesem Kapitel vorgestellte Anwendung soll im Kontext eines biomedizinischen Anwendungsbereiches Mediziner und andere Domänenexperten dabei unterstützen, auf ihre Bedürfnisse zugeschnittene Webapplikationen zu erstellen. Es sei den Medizinern möglich, eine Anforderungsanalyse und eine grobe Struktur der Applikation zu entwickeln, ohne dabei über detailliertes Wissen über die notwendigen Schritte und Fähigkeiten zur Programmierung solch einer Anwendung zu verfügen. Der Umgang mit Objekten, Typen, etc. wird jedoch vorausgesetzt. Das bedeutet, dass die Mediziner Grundlagenwissen der Modellierung, allerdings kein detailliertes Wissen der Programmierung benötigen, um eine domänenspezifische Anwendung zu entwickeln.

Zur Entwicklung der Anwendung, werden im Rahmen der genutzten Tools zunächst in der DyWA die benötigten Typen als Metamodell angelegt. Dies wird durch den Codegenerator in native Klassen überführt, womit die Klassen anschließend im

jABC genutzt werden können. Im jABC werden Abläufe als Prozesse grafisch modelliert und infolge einer erneuten Iteration des Codegenerators ebenfalls in native Klassen überführt.

Die nativen Klassen (Typen und Prozesse) können anschließend in einer Webanwendung (GUI-Builder, s. Abschnitt 5.3.1) genutzt werden, um einzelne Webseiten (Pages) anzulegen, die CRUD-Operationen auf den Instanzen der Typen zulassen bzw. die Prozessformulare grafisch umsetzen. Durch den GUI-Codegenerator wird anschließend eine fertige Webanwendung generiert, die die Typen, Prozesse und GUI-Elemente in einer Tapestry-Webanwendung bündelt (s. Abb. 5.1).

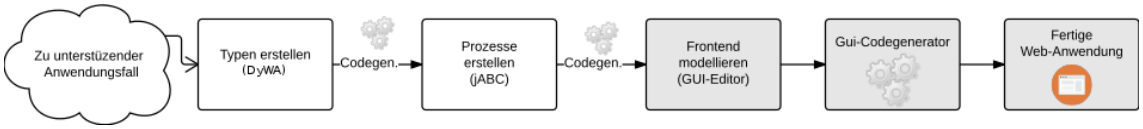

Abbildung 5.1: Vereinfachte Übersicht des Entwicklungsprozesses.

5.1.2 Beispielanwendung

Als beispielhafte Anwendung wird angenommen, dass eine Arztpraxis eine webgestützte Verwaltung ihrer Patienten benötigt. Dazu werden zunächst die Rollen Arzt (mit zwei Instanzen), Arzthelfer (zwei Instanzen) und Blutspezialist (eine Instanz) benötigt. Für Patienten (zehn Instanzen) sind persönliche Daten (Blutgruppe, Größe, Geburtsdatum, etc.) anzulegen. Darüber hinaus soll es möglich sein, für jeden Benutzer beliebig viele Befunde wie auch Blutproben zu erstellen. Des Weiteren soll kontinuierlich das Gewicht des Patienten erfasst werden können. Eine mögliche Implementierung einiger benötigten Typen wird in Abbildung 5.2 dargestellt.

Struktur

Um diese Struktur anzulegen, wird zunächst die DyWA genutzt. Es werden die Objekte und die Klassenstruktur aufgebaut (s. Abb. 5.3). Nach der Codegenerierung können mittels jABC die benötigten Prozesse, wie etwa eine Untersuchung (Gewichtsmessung mit Erstellung eines Befundes), erstellt werden. Durch eine erneute Codegenerierung liegen an dieser Stelle sowohl die Typen als auch die Prozesse als native Java-Klassen vor.

Interaktionsmöglichkeiten

Dem Arzt soll es möglich sein, den Patienten vollständig zu verwalten, mit Ausnahme der Blutwerte, die nur vom Blutspezialisten eingetragen und geändert werden dürfen. Dem Blutspezialisten wiederum dürfen nur Name und Geburtsdatum des Patienten angezeigt werden. Die Arzthelferin kann Patienten anlegen und das Gewicht sowie die Größe eintragen bzw. bearbeiten.

Zur Verwaltung des Patienten gehören:

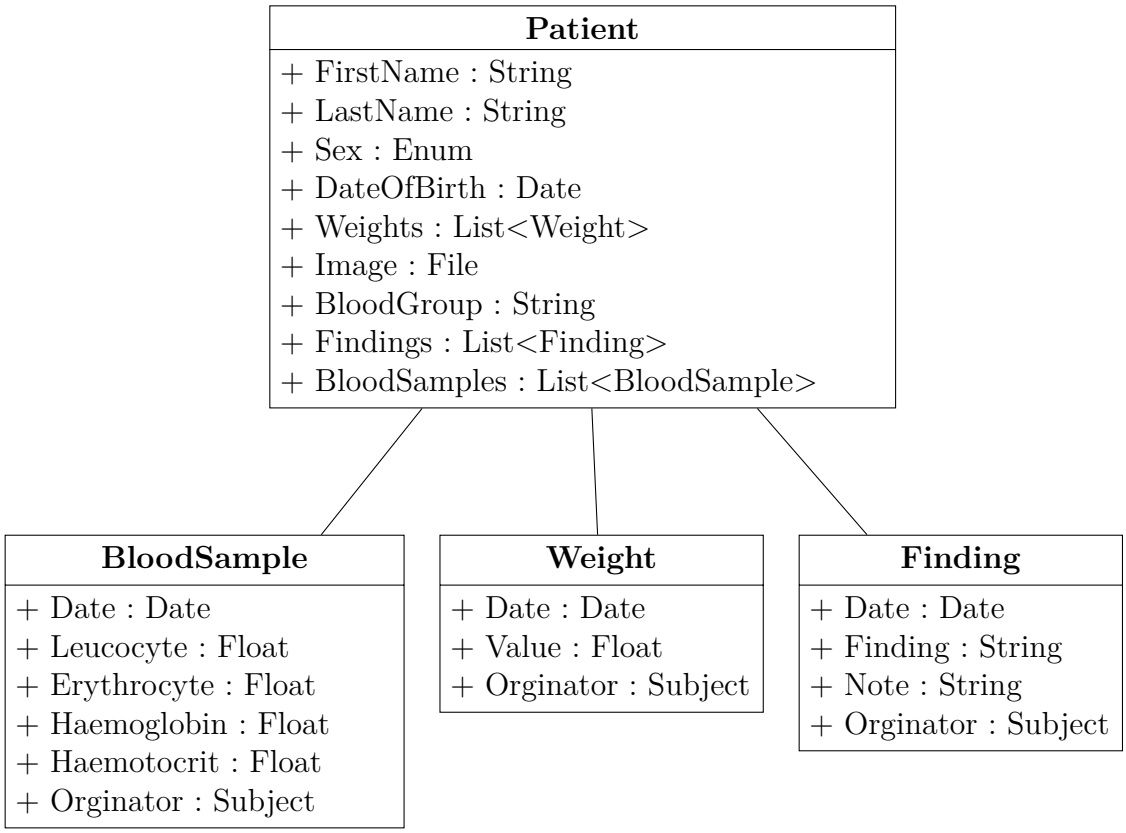

Abbildung 5.2: Beispielhaftes Klassendiagramm für einen Patienten.

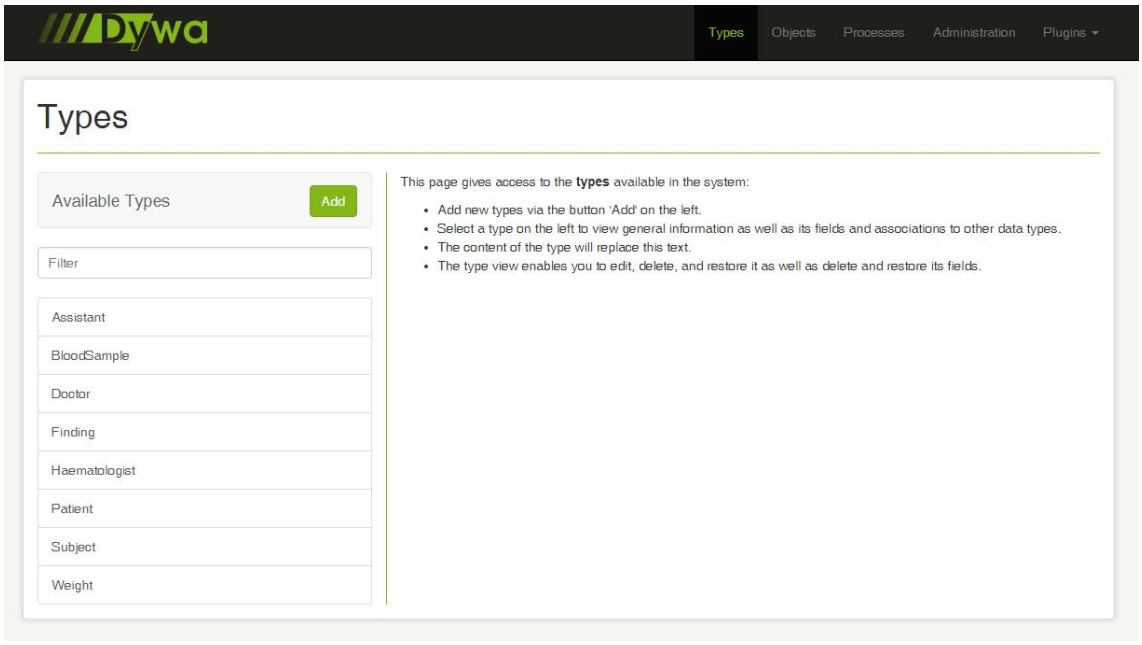

Abbildung 5.3: DyWA Implementierung der benötigten Typen für das Use-Case-Szenario.

- das Anlegen: Name, Größe, Geburtsdatum, Geschlecht, Patientenbild (durch Ärzte oder Klinikpersonal)

- das Eintragen von Werten: Gewicht, Blutwerte (Blutspezialist) und Befunde (Arzt)

- das Bearbeiten von Werten: Befunde, Gewicht, Patientenbild (durch den zuständigen Arzt)

- die Analyse: Patientenübersicht und Gewichtsstatistik (durch Spezialisten/Ärzte)

Das Eintragen von Blutwerden ist verbunden mit einem Prozess:

1. Auswahl des Patienten

2. Abnahme und Einsendung des Bluts

3. Auswertung des Bluts durch Blutspezialist (mehrere Timer, da unterschiedliche Wert-Extraktionen notwendig sind)

4. Eintragung von Blutwerten durch Blutspezialist

5. Analyse der Werte durch zwei unabhängige Ärzte (parallel)

6. Besprechung der Werte mit dem Patienten und einem Arzt

Da Alarme eine zentrale Bedeutung in der Erweiterung von jABC spielen, sind fünf verschiedene Alarme vorgesehen:

- Wenn der Patient ein Jahr nicht bei der Untersuchung war (Datum des letzten Befundes)
  → Nachricht an den Patient (E-Mail) zur Erstellung eines Termins

- Wenn dem Patient zwei Jahre keine Blutprobe entnommen wurde
  → Mitteilung an den Arzt

- Wenn die Blutwerte zwei Wochen lang nicht besprochen/eingesehen wurden
  → Mitteilung an den Arzt

- Wenn die Blutwerte nach fünf Tagen nicht durch Blutspezialist erstellt wurden
  → Mitteilung an den Blutspezialist

- Tägliche Statistik per E-Mail an den Arzt über die Anzahl der behandelten Patienten (ggf. Excel)

Erstellung der Weboberfläche

Den nächsten Schritt bildet die Erstellung der Weboberfläche, welche die benötigten Interaktionen (Anlegen eines Patienten, Aufruf eines Befunds, Eintragen eines Gewichts etc.) ermöglicht. Dazu wird zunächst ein neues Layout angelegt, welches den Kopfbereich der Website festlegt. Danach wird eine neue Page (Webseite) angelegt, der Typ ausgewählt, der dargestellt werden soll, und ein Name für die Seite vergeben. Daraufhin wird das Layout mittels SplitContainern (s. Abb. 5.7) erstellt. Nachdem das grobe Layout angelegt wurde, kann für die einzelnen Bereiche eine entsprechende Komponente (Tabelle, Freitext, etc.) hinzugefügt werden, die zur Anzeige von Elementen (Objekten) des zuvor festgelegten Typs dient. Dabei wird die komplette Objektstruktur des Typs angezeigt. Zur Auswahl stehen für die Elemente jedoch nur die Objekte, die sinnvoll mit der Komponente angezeigt werden können.

Das initiale Layout (s. Abb. 5.4) wird durch horizontale und vertikale Teilung beeinflusst und so zu einer aufgeteilten Seite (s. Abb. 5.5). Die einzelnen Komponenten (DynamicText, Image, Grid, Chart ..) besitzen immer einen Titel, welcher intern zur Identifizierung verwendet wird, und einen Namen, welcher in der domänenspezifischen Anwendung als Überschrift genutzt wird. Zusätzlich sind die notwendigen komponentenspezifischen Elemente, wie z. B. der Text bei Markdown- und DynamicText- Components, oder die Auswahl der anzuzeigenden Attribute des Objekts als entsprechende Spalten bei einer Tabelle (Grid), enthalten. Die Komponenten, aus der die spätere Anwendung besteht, werden im Kapitel 5.3 beschrieben. Die Anwendung soll schlussendlich wie in Abb. 5.6 aussehen.

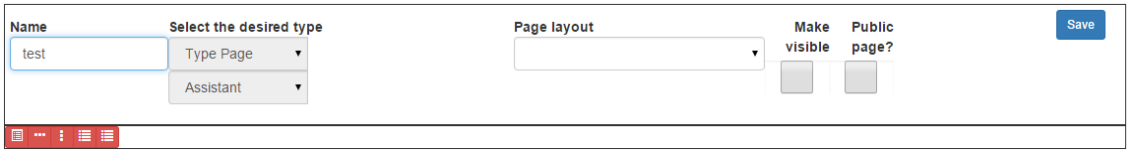

Abbildung 5.4: Initiale Seite ohne Aufteilung

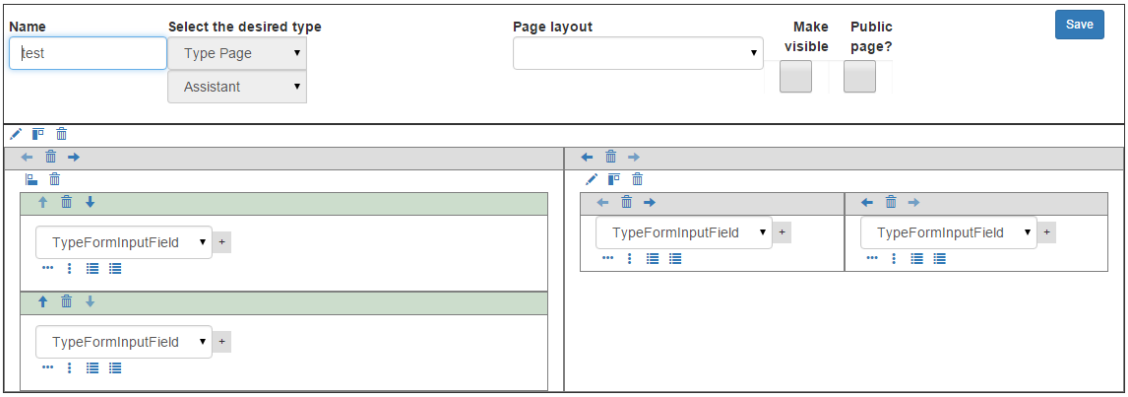

Abbildung 5.5: Seite mit Aufteilung

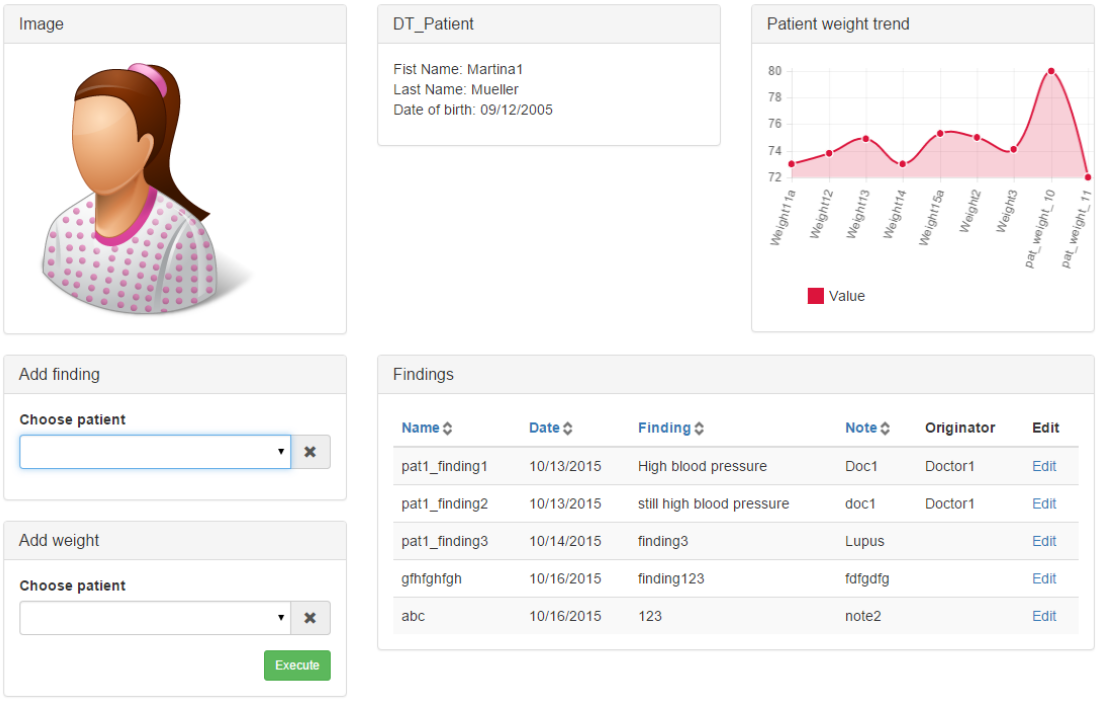

Abbildung 5.6: Beispielanwendung

Die erstellte Oberfläche muss in einer Datenbank gespeichert werden. Dazu wird die in Abschnitt 5.3 vorgestellte Architektur genutzt. Nach der Erstellung der domänenspezifischen GUI wird mittels des Code-Generators (s. Abschnitt 5.3.3) eine lauffähige Webanwendung generiert, welche auf einem Webserver deployed werden kann. Erwähnenswert ist an dieser Stelle zudem das Draft-Feature, welche es ermöglicht, Seiten zu erstellen, diese allerdings nicht zu veröffentlichen. Wird der „Make Visible“-Haken nicht gesetzt, wird die jeweilige Seiten zwar im Metamodell gespeichert, allerdings nicht in die domänenspezifische Anwendung generiert. So lassen sich auch Entwürfe zu einzelnen Seiten erstellen. Ein Vorteil des Entwicklungsprozesses und der verwendeten Tools ist die Live-Anpassungsmöglichkeit. Das bedeutet, dass während des Einsatzes der Anwendung die Typen, die Prozesse und die Oberfläche angepasst werden können.

5.2 Technologieauswahl

Dieser Abschnitt beschreibt die zur Realisierung verwendeten Technologien und die Gründe für ihre Auswahl. Da die finale Webanwendung vollständig generiert wird, war die Auswahl der Technologie unabhängig von den bisher in der DyWA verwendeten Technologien und konnte aus sämtlichen verfügbaren Frameworks erfolgen.

5.2.1 ZK

ZK ist ein Framework zur Entwicklung von Webanwendungen.[4] Die grundlegende Arbeitsweise des Frameworks erinnert eher an klassische Desktop- als an Webanwendungen. Das User Interface wird in XUL[1]-Dokumenten vorgegeben. Durch Userinteraktionen werden Events an eine dazugehörige Java-Klasse geschickt, welche die Verarbeitung und Manipulation der Seite übernimmt.

Potenzielle Vorteile des ZK-Frameworks sind zum einen die direkte Einbindung von Bootstrap (siehe Sektion 5.2.4) und zum anderen die Möglichkeit, Webanwendungen ohne Kenntnisse in Ajax und Javascript zu entwickeln. ZK besitzt dafür zahlreiche vorgefertigte und leicht zu integrierende Komponenten, welche die Funktionalitäten von Ajax und Javascript [9] implementieren.

Als nachteilig erweist sich die Lizenzierung. Während die Hauptkomponente unter der unproblematischen LGPL[2] lizenziert ist, sind die Erweiterungen des Frameworks kritisch. Diese sind, wie die Komponente zur Erstellung von Charts, entweder nur als kommerzielle Variante verfügbar und daher generell zur Verwendung in der Projektgruppe ungeeignet oder aber unter der GPL[3] lizenziert. Dadurch müsste ggf. das gesamte Ergebnis der Projektgruppe auch unter der GPL veröffentlicht werden. Letztendlich gab die unklare Lizenzierung einiger, wahrscheinlich aber benötigter Teilkomponenten den Ausschlag dafür, von der Verwendung von ZK abzusehen.

5.2.2 AngularJS

AngularJS [27] ist ein unter anderem von Google entwickeltes Framework. Anders als ZK und Apache Tapestry basiert AngularJS auf Javascript. Es realisiert das Entwurfsmuster Model-View-ViewModel (MVVM) und ist eines der wenigen Javascript-Frameworks, die dieses Entwurfsmuster realisieren und gleichzeitig Dependency Injection unterstützen.

Im Vergleich zu anderen Frameworks ist AngularJS-Code sehr kompakt. Da jedoch die Zielanwendung vollständig generiert wird, ist die Größe des Codes im Endeffekt irrelevant.

Probleme gab es bei der Einbindung zusätzlicher Bibliotheken. Hier war eine einfache Einbindung nicht immer möglich und erforderte das Schreiben eines Adapters. Da geplant ist, weitere Features wie zum Beispiel Graphen durch Nutzung zusätzlicher Bibliotheken zu realisieren, könnte dies die Auswahl entweder einschränken oder den Generierungsprozess erschweren.

Die Unwägbarkeiten in Bezug auf die Verwendung zusätzlicher Bibliotheken führten schließlich zu der Entscheidung, AngularJS nicht zu verwenden.

---

[1] **XUL:** XML User Interface Language, eine auf XML-basierende Beschreibungssprache für grafische Benutzeroberflächen. Vgl [4]

[2] GNU Lesser General Public License

[3] GNU General Public License

5.2.3 Apache Tapestry

Genau wie ZK ist auch Apache Tapestry ein Framework zur Entwicklung von Webanwendungen [23]. Da auch die DyWA mit Tapestry realisiert ist, war es naheliegend, Tapestry auf die Eignung zum Generieren von Code zu evaluieren. Apache Tapestry ist ein auf Java basierendes Open Source Framework unter der Apache Lizenz 2.0. Probleme und Unklarheiten in Bezug auf die Verwendung können daher ausgeschlossen werden.

Als Vorteile im Vergleich zu ZK hat sich hier die Nähe zur bestehenden Technologie erwiesen. Darüber hinaus soll die Oberfläche zum Erstellen der GUI in die DyWA eingebunden werden. Auch hier ist es vorteilhaft, dass bei der Verwendung von Tapestry alle Teile mit der gleichen Technologie realisiert sind. So können Teile des Designs übernommen werden, sodass kein zusätzlicher Aufwand entsteht, um eine optisch konsistente Einbindung zu gewährleisten. Tapestry erlaubt ebenfalls die Verwendung von Bootstrap und bindet es mit der Version 5.4 (derzeit noch Beta) sogar standardmäßig ein. Aus diesem Grund wird das Framework auch in der GUI der hier beschriebenen Webanwendung verwendet.

Als nachteilig kristallisierten sich vor allem die aus unserer Sicht schwierig zu verstehende Dokumentation und die damit verbundene steile Lernkurve heraus. Die Entwicklung von kleinen Testanwendungen während der Evaluation gestaltete sich mit Tapestry am schwierigsten. Snippets aus der Dokumentation ließen sich oft nur schwierig in die Beispielanwendung übernehmen.

5.2.4 Bootstrap

Die generierten Seiten sollen natürlich nicht nur funktional, sondern auch optisch ansprechend und unabhängig von dem verwendeten System sein. Die Vielzahl möglicher Kombinationen von Betriebssystem, Browser und Bildschirmauflösung zusätzlich in die Codegenerierung direkt einfließen zu lassen, könnte zu einer stark steigenden Komplexität führen. Durch die Verwendung von Bootstrap kann diese Aufgabe an das Framework abgeben werden.[4]

Insbesondere die Unterstützung von Responsive Web Design (RWD) gab hier den Ausschlag für die Verwendung. Mit Bootstrap soll sichergestellt werden, dass die generierte Anwendung auf einer Vielzahl von Zielsystemen, inklusive mobilen Systemen wie Tablets, gut bedienbar ist. Darüber hinaus wird Bootstrap von jedem modernen Browser unterstützt, was das Ziel, auf möglichst vielen Systemen lauffähig zu sein, unterstützt. Auch Bootstrap ist unter der für das Projekt gut verträglichen Apache 2.0-Lizenz lizenziert und kann daher verwendet werden.

5.2.5 Scala

Zu der Realisierung des Codegenerators wird auf einen in Java und Scala geschriebenen Template-basierten Codegenerator zurückgegriffen, welcher als Maven-Plugin realisiert ist und so in das Hauptprojekt eingebunden werden kann. Eine genaue

[4] Vgl: `http://holdirbootstrap.de/`

Beschreibung der Funktionsweise findet sich im Abschnitt 5.3.3. Der bereits vorhandene Codegenerator zu der Generierung der domänenspezifischen Klassen basiert ebenfalls auf diesem Konzept.

5.2.6 Entscheidung

Die Entscheidung für ein Framework fiel zu Gunsten von Apache Tapestry aus. Anders als die anderen untersuchten Frameworks gab es keinen schwerwiegenden Nachteil, der gegen die Verwendung spricht. Die steilere Lernkurve wird hierfür in Kauf genommen.

Der entwickelte Codegenerator zur Generierung der domänenspezifischen Anwendung orientiert sich an den bereits im DyWA-Framwork verwendeten Codegeneratoren. So wird innerhalb des Projekts eine konsistente Technologiewahl gewährleistet, wodurch eine verbesserte Zusammenarbeit der Projektgruppenteilnehmer ermöglicht wird.

5.3 Architektur

Im folgenden Kapitel soll der Aufbau des GUI-Builders und seine Verwendung erläutert werden. Dazu wird zunächst die Architektur anhand eines UML-Diagrammes vorgestellt, später wird dann gezeigt, wie mit Hilfe der einzelnen Komponenten komplexere Views erzeugt werden können.

5.3.1 GUI-Builder

Aufbau

Die folgende Aufzählung dient der Erklärung des Klassendiagramms in Abbildung 5.7.

- **Pages:** Die Pages sind die Objekte, die die einzelnen Komponenten der View enthalten und stellen einzelne Tapestry-Seiten dar. Es wird dabei zwischen verschiedenen Typen von Pages unterschieden. Diese sind im Einzelnen:
  - General Pages: Hierbei handelt es sich um statische Pages. Dynamische Inhalte wie Objekte aus der Datenbank können damit nicht dargestellt werden. Anwendung findet diese Art von Pages vor allem als Startseite einer Applikation.
  - Type Pages: Diesen Pages wird ein Datentyp aus der DyWA zugeordnet. Sie dienen dazu, Objekte des zugeordneten Datentyps anzuzeigen und zu editieren.
  - Process Pages: Äquivalent zu den Type Pages, nur dass diesen Pages ein Prozess anstelle eines Datentyps zugeordnet wird. Die Page wird dann für das Starten des Prozesses verwendet.

  - Layout: Jedem anderen Typen von Pages kann ein Layout zugeordnet werden. Dabei handelt es sich um immer wiederkehrende Gestaltungselemente wie beispielsweise Menüs, die auf mehreren Pages verwendet werden sollen. Layouts sind rein statische Pages.

- **Component:** Dies ist die abstrakte Oberklasse für alle GUI-Elemente. Diese GUI-Elemente können entweder Components oder Container sein.

- **Container:** Hierbei handelt es sich um die abstrakte Oberklasse für alle Komponenten einer Page, die Unterelemente haben können. An dieser Stelle werden bereits grundlegende Designelemente wie z. B. Abstände definiert, die später im Generierungsprozess auf CSS Klassen gemappt werden können. Ein Container kann entweder ein CompoundContainer oder ein SimpleContanier sein.

- **CompoundContainer:** Sie enthalten entweder mehrere weitere Container oder Komponenten. Fügt man mehrere Komponenten in den Container ein, so muss man durch Verwendung der HSplit- bzw. VSplitContainer-Klasse angeben, wie man den Container optisch unterteilen möchte. Durch die Verwendung dieses Konstrukts entsteht eine Baumstruktur, die später die einzelnen Komponenten zusammen fügt.

- **SimpleContanier:** Dieser Container enthält genau eine innere Component und ist in verschiedene Typen unterteilt.

- **ProcessFormContainer:** Eine konkrete Klasse, die von SimpleContainer erbt und die mit der inneren Component ein Formular für ein Prozess bündelt.

- **TypeFormContainer:** Eine konkrete Klasse, die von SimpleContainer erbt und die mit der inneren Component ein Formular für ein DBType bündelt.

- **BasicComponent:** Dies ist eine abstrakte Oberklasse für konkrete View-Komponenten zur Darstellung von (teils statischen) Elementen wie beispielsweise Text, Integer-Zahlen, Date-Objekten, Listen, Links zu einer anderen Page, etc.. Für die einzelnen Einsatzbereiche der Component gibt es jeweils eigene Unterklassen, die von dieser Oberklasse erben und die über entsprechende spezifische Attribute verfügen.

Verwendung

In der DyWA-Oberfläche des Projektes der GUI-Gruppe existiert ein Menü-Eintrag *View-Builder*, der es ermöglicht, für jeden DBType, den die DyWA kennt, mehrere Components zu erstellen. Für die primitiven Datentypen gibt es vorkonfigurierte BasicComponents (siehe oben), die ebenfalls benutzt werden können. Da jede Component über einen Namen und einen entsprechenden DBType verfügt, ist der Aspekt der Wiederverwendung realisiert. Wie bereits im vorherigen Abschnitt erläutert, wird jede Component in einen Container eingesetzt, der wiederum die Verbindung zur Page herstellt und verschiedene Components miteinander verbinden kann.

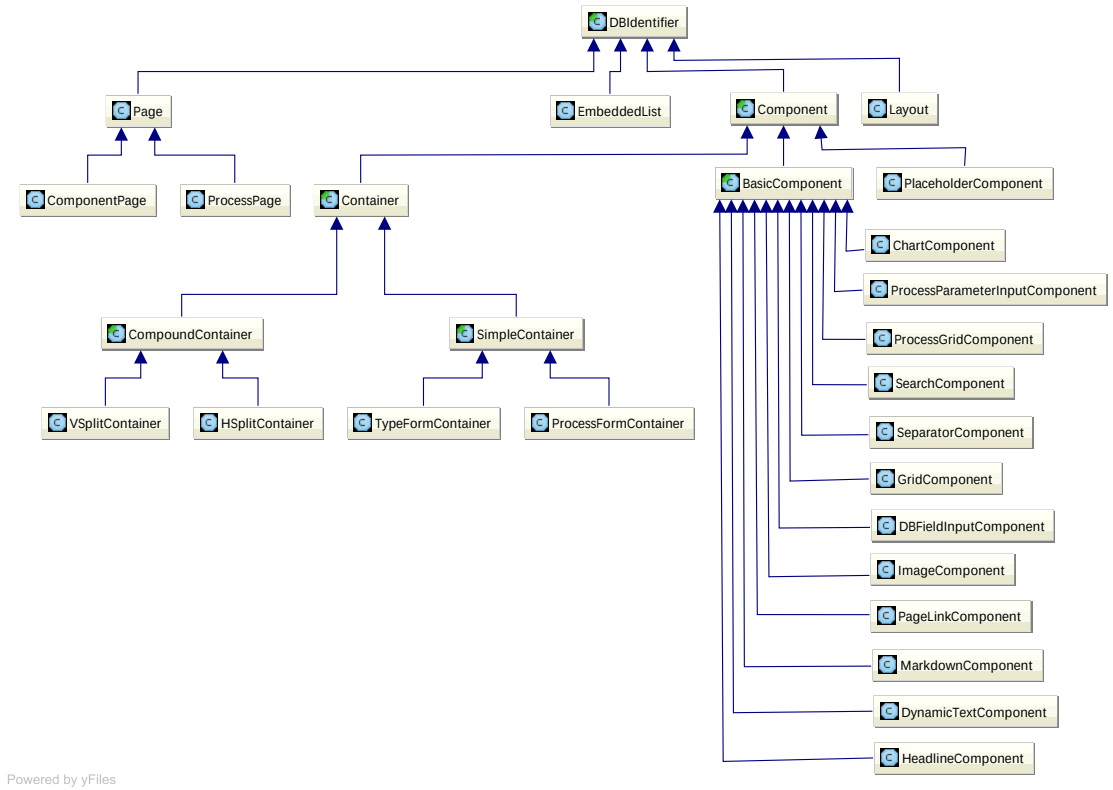

Abbildung 5.7: Klassendiagramm der Architektur

Container lassen sich beliebig verschachteln und unterteilen. Durch das Erzeugen von HSplit- bzw. VSplitContainer lassen sich Pages optisch unterteilen. Eine grobe Skizze, wie dies aussehen kann, zeigt Abbildung 5.8.

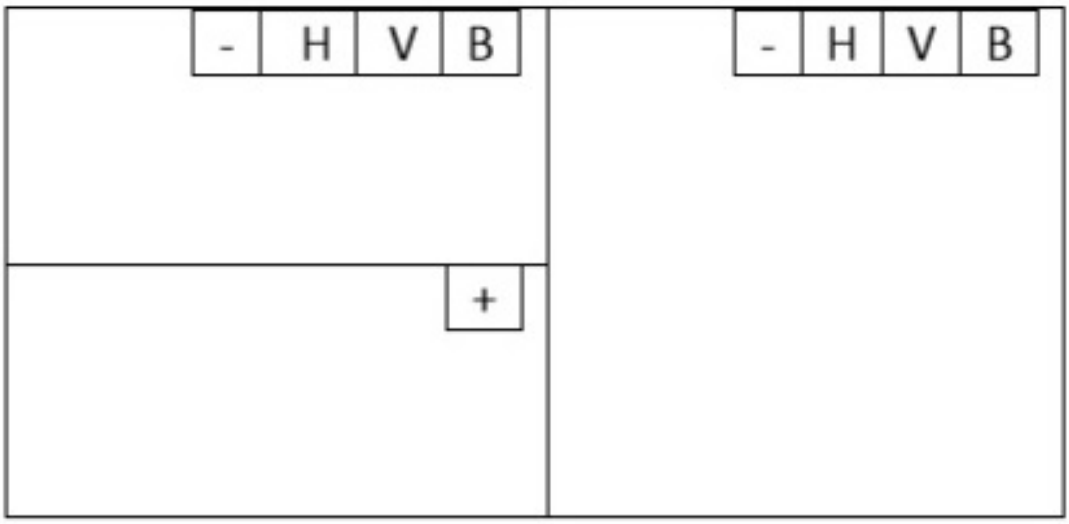

Abbildung 5.8: Skizze des Aufbaus einer View.

Hier enthält die Page einen „Ober-Container“, der wiederum einen VSplitContainer enthält. Auf der rechten Seite kann nun sofort eine Component eingesetzt werden, links wurde durch Nutzung eines HSplitContainers die View noch weiter unterteilt. In diesem Konstrukt könnten nun drei Components dargestellt werden. Wie eine konkrete Beispiel-View aussehen kann, zeigt Abbildung 5.9. Hier wurde ein Container in insgesamt fünf „Unter-Container“ unterteilt, die jeweils ein Element enthalten. Zu sehen ist eine ImageComponent zum Darstellen eines Bildes, drei Components zum Darstellen eines Patienten bzw. einer Adresse sowie eine Grid-Component mit einer Liste von Befunden.

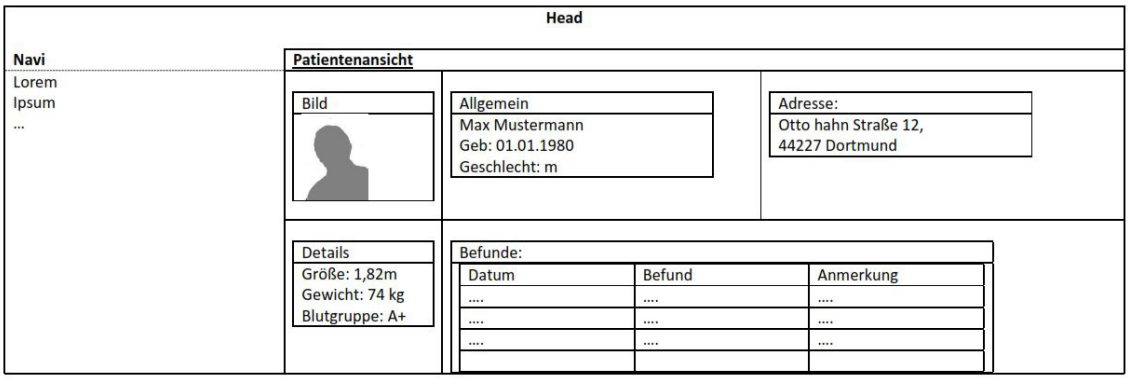

Abbildung 5.9: Beispielhafter Aufbau einer View.

Da die Verknüpfung von Komponenten zu einer einzelnen Page ausschließlich über Container geschieht, hat jede Komponente selbst keinerlei Informationen über

die anderen in der Page enthaltenen Komponenten. Eine Komponente „ViewAddress“ selbst hat folglich nur die Information, dass sie ein Objekt vom Typ „Address“ darstellen kann. So sind auch die einzelnen Components wiederverwertbar. Die Navigation durch die Page erfolgt durch die entsprechenden Attribute der Container. Der Nutzer legt diese Pages in einem Editor an, dessen Funktionsweise im folgenden Abschnitt erläutert werden soll.

Der Page Editor

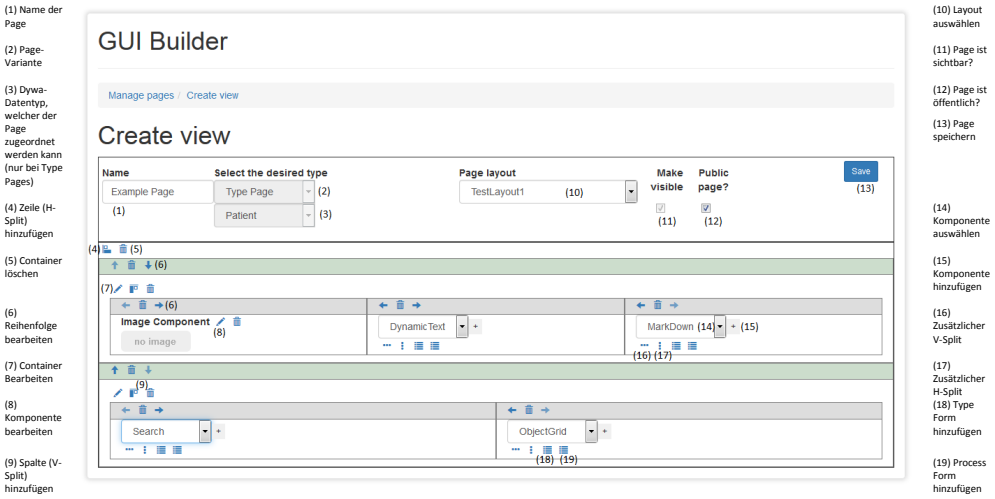

Abbildung 5.10: Der GuiBuilder

Wie bereits in Abb. 5.9 zu sehen ist, werden Pages von „außen“ nach „innen“ aufgebaut. Zuerst wird der grobe Rahmen bestehend aus den entsprechenden Containern erzeugt. Es folgen die benötigten Components sowie deren Inhalt. Auf diese Art und Weise werden die Pages auch im Frontend erzeugt. Es wurde zu diesem Zweck ein Editor entwickelt, der es ermöglicht, Pages zu erstellen und zu bearbeiten. Wie in Abbildung 5.10 zu sehen, lassen sich Pages „zusammenklicken“. Auch hier sollte von „außen“ nach „innen“ gearbeitet werden. Das bedeutet, dass zunächst der grobe Aufbau der Page durch hinzufügen und unterteilen der einzelnen Container definiert werden muss. Dadurch erstellt der Nutzer eine Art Tabelle, deren Zellen dann mit einzelnen Komponenten befüllt werden kann. Dies geschieht durch eine Auswahl in der entsprechenden Dropdown-Box und einem anschließenden Klick auf den dazugehörigen Plus-Button. Welche Komponenten zur Verfügung stehen, kann dabei je

nach Art der Page variieren. Auch die Container selbst lassen sich gesondert konfigurieren (z.B. Anpassung der Breite). Dazu muss lediglich auf den entsprechenden Button geklickt werden.

Um die einzelnen Komponenten konfigurieren zu können, wurden jeweils eigene Editoren entwickelt, die durch einen Klick auf das jeweilige „Bearbeiten“ -Icon der einzelnen Komponenten aufgerufen werden können. Bei den meisten handelt es sich um einfache Formulare, andere sind komplexer. Vor allem letztere sollen nun vorgestellt und erläutert werden.

Der Markdown-Editor

Der Markdown[5]-Editor dient zum Erstellen und Editieren von Markdown-Components. Dies sind Komponenten, die statischen Text enthalten. Dies können z.B. Informationstexte oder rechtliche Anmerkungen, die unabhängig von einzelnen Objekten oder Typen immer identisch bleiben sein. Der Editor besteht hauptsächlich aus einer Textarea, in der der jeweilige Text eingetragen werden kann, und einigen Schaltflächen für die Formatierung. Dabei handelt es sich um keine Eigenentwicklung. Vielmehr wurde auf einen unter Open-Source-Lizenz bereitgestellten Markdown-Bootstrap-Editor[6] zurückgegriffen. Für Dokumentation bezüglich dieses Editors sei auf das entsprechende Handbuch verwiesen.

Zur Editorseite gehört noch ein Preview-Feld, sowie die Möglichkeit eine Überschrift für die Komponente (Title) und einen internen Namen (Component Name) festzulegen. Das Preview-Feld dient dazu, den Text formatiert anzuzeigen. Das Übersetzen des Markdown-Codes übernimmt ebenfalls eine fertige Bibliothek[7]. Um dieses Feld zu aktualisieren ist ein Klick auf den Preview-Button erforderlich. Der Save-Button speichert die Seite und springt zurück zum Page-Editor.

Der Dynamic-Text-Editor

Genau wie die Markdown-Component dient auch die Dynamic-Text-Component zum Darstellen von Texten. Entsprechend ist auch die Oberfläche des Editors nahezu identisch aufgebaut (siehe Abbildung 5.11) und soll deshalb an dieser Stelle nicht erneut erklärt werden. Im Unterschied zu rein statischen Texten, die in der Markdown-Component gespeichert werden, kann die Dynamic-Text-Component auch dynamische Texte, wie z.B. Attribute von Objekten, enthalten. In der Editoransicht werden diese zunächst durch Platzhalter repräsentiert, die durch spitze Klammern erkennbar sind. Soll z.B. an einer Stelle im Text der Vorname eines Patienten erscheinen, so muss im Text an der entsprechenden Stelle <Vorname> stehen. Es ist sowohl möglich, die Platzhalter selbst in den Text-Editor einzugeben, als auch über das Dropdown-Menü an der Unterseite des Editors per Mausklick hinzuzufügen. Wird

[5]**Markdown:** Auszeichnungssprache, die ohne weitere Konvertierung leicht lesbar ist. Als Auszeichnungselemente werdenAuszeichnungsarten verwendet, die in Plaintext und E-Mails üblich sind. Vgl: `http://markdown.de/`

[6]Toopay Bootstrap Markdowneditor: `http://www.codingdrama.com/bootstrap-markdown/`

[7]markdown4j: `https://www.versioneye.com/java/org.commonjava.googlecode.markdown4j:markdown4j/2.2-cj-1.0`

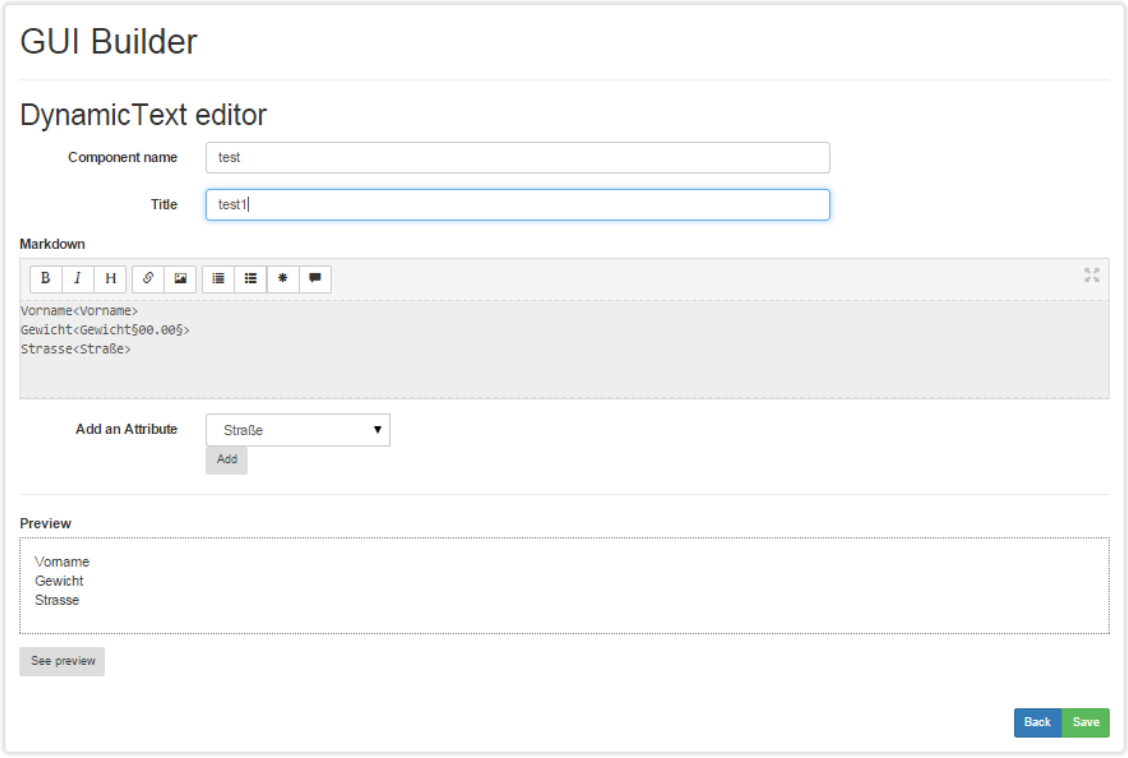

Abbildung 5.11: Ansicht des Dynamic-Text-Editors

ein Platzhalter eingegeben, dessen Bezeichnung innerhalb der spitzen Klammern nicht mit dem Namen eines der zur Verfügung stehenden Attribute korrespondiert, wird dieser später ignoriert.

Es ist wichtig, dass ausschließlich Attribute des Typs auswählbar sind, die zu dem Typen der Page gehören, deren Teil die jeweilige Dynamic-Text-Component ist. Das bedeutet: Ist die Page vom Typ Patient, sind auch nur Attribute eines Patienten auswählbar. Möchte man Daten eines anderen Typs auf einer Page anzeigen, ist es notwendig, eine andere Page mittels eines Page-Links anzuhängen.

Hat ein Datentyp Attribute, die selbst wiederum ein komplexer Datentyp sind, ist allerdings möglich, auch dessen Attribute in den Dynamic-Text einzufügen. Ein mögliches Beispiel wäre ein Objekt vom Typ Patient, das über ein Attribut „Adresse“ verfügt, welches seinerseits wiederum die Attribute „Strasse“, „Hausnummer“, „PLZ“ und „Ort“ aufweist. Auch diese vier Attribute stehen dann in einer Dynamic-Text-Component mit dem Typen Patienten zur Verfügung. Allerdings ist diese Verkettung von Untertypen auf eine Tiefe von zwei Unterebenen beschränkt. Die Attribute eines komplexen Attributes von Adresse werden folglich noch erkannt, komplexe Attribute dieses Untertyps werden aber ignoriert.

Für Fließkommazahlen ist es darüber hinaus noch möglich, Formatstrings mit anzugeben. Diese müssen zwischen zwei Paragraphensymbolen innerhalb eines Platzhalters stehen. So gibt <Gewicht§00.00§> in der späteren, generierten Anwendung ein Gewicht mit zwei Stellen vor und zwei Stellen nach dem Komma an. Die Formatstrings werden beim Speichern in einer gesonderten Map abgelegt, die alle Format-

strings einer Component in Verbindung mit dem jeweiligen Attribut enthält. Wird für ein Attribut kein Formatstring angegeben, enthält die Map auch keinen Eintrag für das jeweilige Attribut.

Bevor die Texte in die Datenbank geschrieben werden, werden die Platzhalter bereits zum ersten Mal ersetzt. Anstelle des Attributnamens wird *fieldX* in die spitzen Klammern geschrieben, wobei X für in Index des jeweiligen Attributes in der Fieldlist des jeweiligen Typs steht. Für den Anwender ist dieser Schritt nicht zu sehen und auch nicht relevant. Allerdings handelt es sich bei dem fieldX-Format um das Eingabeformat für Platzhalter, welches der Codegenerator benötigt. Das Umwandeln der Platzhalter geschieht im Programmcode in den Methoden *MaskValue* und *DemaskValue* in der Java-Klasse zur Page vom Editieren von Dynamic-Text-Components.

Der Edit-Grid-Component-Editor

Die GridComponent dient der tabellarischen Anzeige von Listenobjekten eines Typen. Die Oberfläche besteht aus einem Textfeld für den Namen der Komponente sowie einem weiteren Textfeld, in dem der Typ der Page steht. Es kann dann ein Attribut des Typen, das gleichzeitig eine Liste ist, ausgewählt werden. Mit dem „select“ -Button wird die Auswahl bestätigt und es erscheinen die Attribute der ausgewählten Listenobjekte. Nun kann mittels Auswahlboxen entschieden werden, welche Attribute in der Komponente, also in der Tabelle, angezeigt werden sollen. Des Weiteren ist auswählbar, welche Operationen von der tabellarischen Sicht aus angezeigt werden sollen. So kann das Löschen, Editieren oder Anzeigen einzelner Attribute ermöglicht werden. Handelt es sich Beispielsweise um eine Page des Typen Patient und der Patient hat als Attribut eine Liste von Blutwerten, so kann „Blutwert“ selektiert werden. Es erscheinen die Attribute des Blutwert-Typs. Nun kann ausgewählt werden, welche dieser Attribute in der Tabelle als Spalten erscheinen sollen.

ChartComponent

Mit Hilfe der ChartComponent soll es dem Anwender ermöglicht werden, einfach und schnell Diagramme verschiedener Art zu erstellen und in der Webapplikation zu nutzen. Es stehen Linien-, Balken-, Kreis- und Sterndiagramme zur Verfügung. Als Datenquelle können Listen von Datentypen gewählt werden, die Attribute mit Double- oder Long-Feldern besitzen. Eine andere Datenquelle stellen Prozesse dar, mit denen durch die Nutzung von Statistikfunktionen (mehr dazu in Abschnitt 4.3.2) komplexe Zusammenhänge erfasst und geeignet dargestellt werden können. Zur Umsetzung wurde auf Chart.js [3] zurückgegriffen, da es unter MIT Lizenz steht und für unsere Aufgaben ausreichend dimensioniert erscheint. Als Alternative bietet sich das mächtigere d3.js [2] an. Auf Grund des großen Umfanges haben wir uns aber gegen dieses Tool entschieden, da die grundlegende Arbeit genauso gut mit einem kleineren Tool gemacht werden kann. Die dritte Alternative zur Erstellung von Charts war Highcharts [12], welches besonders durch seine einfache Integration in Tapestry auffiel. Allerdings hat uns die Lizenzierung davon abgehalten. Eine Anpassung des

verwendeten Tools ist allerdings auch zu einem späteren Zeitpunkt möglich, da die verwendeten Datenquellen erhalten bleiben und nur der Codegenerator entsprechend angepasst werden muss. Dieses sollte allerdings weniger aufwendig sein.

5.3.2 Formulare

Formulare sind meist ein zentraler Bestandteil jeder (Web-)Anwendung. Sie dienen dazu, Eingaben des Nutzers entgegen zu nehmen und diese zu verarbeiten. Formulare stellen somit die gängigste Art und Weise der Interaktion zwischen dem System und dem Nutzer dar. Aus diesem Grund ermöglicht der GUI-Builder auch die Einbindung von Formularen in eine Page. Unterschieden wird dabei zwischen den „Type Forms“ und den „Process Forms“ . Diese sind aufrufbar durch die Buttons 18 und 19 in Abbildung 5.10. Prozess-Formulare können in jeder Art von Page verwendet werden, Formulare für Typen dagegen nur auf ComponentPages.

Die dazugehörige Editoroberfläche ist sehr einfach gehalten und soll an dieser Stelle nicht weiter thematisiert werden. Wichtig zu erwähnen ist allerdings, dass es sich bei diesen Formularen zunächst um spezielle Container handelt. Diese werden (wie andere Container auch) in eine Page eingebunden und können anschließend mit beliebigen Components befüllt werden. Für Nutzereingaben sind hier die DBFieldInput bzw. die ProcessParameterInputComponent wichtig. Ersteres ist dabei für Type Forms gedacht, letztere für Process Forms. Der Unterschied zwischen den beiden Typen von Formularen liegt in ihrer Funktion. Type Forms sind an einen bestimmten Datentyp gebunden. Nutzereingaben befüllen hier in der Regel die jeweiligen DBFields des Typs. Diese Art von Formularen kann zum Beispiel dazu genutzt werden, um einen neuen Patienten anzulegen und in der Datenbank zu speichern.

Process Forms hingegen werden an einen zuvor definierten Prozess gekoppelt. Hier werden mit den Nutzereingaben die Parameter dieses Prozesses belegt. Pages, die Process Forms enthalten, dienen folglich dazu einen Prozess anzustoßen. Ein Beispiel kann das Durchführen einer Untersuchung sein, bei der der Anwender zunächst einige Daten eingeben muss. Beide Formulartypen können über die „Container editieren“ -Schaltfläche (Nummer 7 in Abbildung 5.10) konfiguriert werden.

5.3.3 GUI-Codegenerator

Da Scala und Java miteinander kompatibel sind, kann bei der Generierung auf die Vorteile beider Programmiersprachen zurückgegriffen werden. Die Sammlung und Vorbereitung der Daten findet Java-basiert statt. Die letztendliche Codegenerierung wird Template-basiert von Scala-Funktionen übernommen.

Um einen möglichst hohen Grad der Anpassung durch den Benutzer zu gewährleisten und gleichzeitig die Templates so einfach wie möglich zu halten wird darauf verzichtet, ein großes Template für alle denkbaren Seiten zu erstellen. Dies hätte entweder die Möglichkeiten des Benutzers stark eingeschränkt oder enormen Aufwand zur Folge gehabt. Stattdessen wird die Speicherung der nötigen Daten in einer Baumstruktur ausgenutzt. Hierzu wird für jede GUI-Klasse je ein Template für die Java-Klasse und die .tml-Datei erstellt. Dieses Template enthält, je nach Klasse, ent-

weder einen bloßen Rahmen für die Kindobjekte (dies sind insbesondere die Containerklassen) oder funktionale Codeabschnitte (für die Components). Die funktionalen Codeabschnitte wären ohne den passenden Rahmen nicht funktionsfähig.

Während der Codegenerierung wird der Baum mit einem *Divide and Conquer*-Algorithmus durchlaufen, sodass jeder Knoten nur seinen eigenen Code generiert. Dadurch bleibt auch die Struktur des Dokuments vollständig erhalten.

Einfaches Beispiel

Es soll eine simple Seite generiert werden, welche vertikal geteilt ist. Im linken Teil werden die Attribute eines Objekts angezeigt, die rechte Seite enthält einen simplen, statischen Text.

Eine solche Seite wird als ComponentPage, welche einen VSplitContainer enthält, gespeichert. Der VSplitContainer wiederum besitzt eine DynamicTextComponent (für die linke Seite) und eine MarkdownComponent (für die rechte Seite). Abbildung 5.12 zeigt, wie die eben beschrieben Beispielseite im GUI-Editor aussehen könnte.

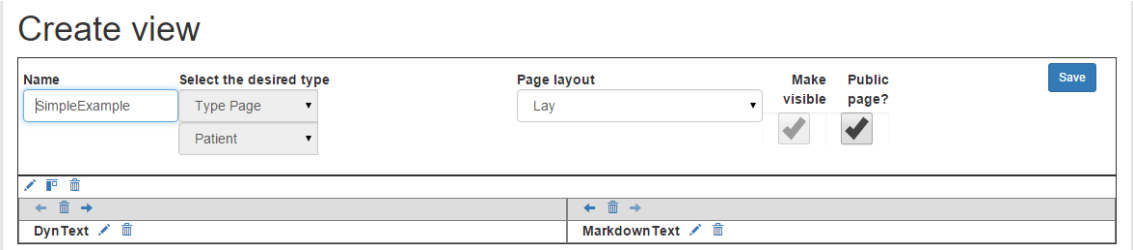

Abbildung 5.12: Beispielseite im GUI-Editor

Wird nun die Seite generiert, beginnt der Generierungsprozess bei der Wurzel des Baumes, der ComponentPage, und generiert aus ihr den groben Rahmen der Seite. Das sind insbesondere der Seitentitel, öffnende und schließende HTML-Tags sowie Imports für benötigte Tapestry-Klassen.

Die noch leere Mitte wird durch den VSplitContainer aufgefüllt. Hier wird nun ein inhaltsloser Rahmen für die Kindkomponenten generiert, in welchen der Code der Kindkomponenten eingefügt wird. Dadurch existiert nun das Layout der Seite die, bis auf den Titel, noch leer ist. Als reine Layout-Komponente generiert diese keinen neuen Java-Code.

Zum Schluss werden die DynamicTextComponent und MarkdownComponent generiert. Letztere besitzt ebenfalls keine Logik, sondern generiert auch nur Text zum Einfügen in die noch leere rechte Teilkomponente des VSplitContainers. Die DynamicTextComponent hingegen generiert zusätzlich noch einige Zeilen Java-Code, um Zugriff auf das gewünschte Objekt zu erhalten.

5.4 Zusammenfassung

Das Ziel, dem Benutzer eine grafische Oberfläche zur Modellierung einer Anwendung zur Verfügung zu stellen, wurde erreicht. Der neu entwickelte GUI-Editor ermöglicht

es dem Domänenexperten ohne Programmierkenntnisse eine Anwendung zu entwickeln. Hierbei kann er / sie auf zahlreiche Gestaltungskomponenten zurückgreifen, um Layout und die Darstellung der Daten an die Bedürfnisse der Endbenutzer anzupassen. Eine Absicherung der einzelnen Komponenten ist über die Einbindung der in Kapitel 7 beschrieben Funktionen möglich. Auch das Ausführen der in jABC modellierten Prozesse wurde ermöglicht.

Der ebenfalls neu entwickelte Codegenerator erlaubt es, die im GUI-Editor modellierten Daten zu einer Apache Tapestry Webanwendung zu generieren. Hierzu werden die, an anderer Stelle ebenfalls generierten, domänenspezifschen Klassen und Controller verwendet, sodass die generierte Webanwendung nicht (mehr) direkt auf dem Metaschema der DyWA arbeitet.

5.5 Ausblick

Es ist gelungen einen voll funktionsfähigen GUI-Editor zu entwickeln, welcher dem Endnutzer alle bisher verfügbaren Features zur Verfügung stellt. Künftige Erweiterungen haben daher vor allem das Ziel, dem Nutzer bestehende Funktionen auf andere Weise zur Verfügung zu stellen oder die allgemeine Bedienbarkeit zu erleichtern und so ein angenehmeres und effizienteres Arbeiten zu ermöglichen.

5.5.1 Erweiterung der Suchfunktionalität

Zusätzlich soll es möglich sein, die vordefinierten Suchen in der generierten Anwendung über Suchfelder vervollständigen oder verändern zu können. So soll dem Anwender ermöglicht werden, bei einer Suche, welche alle männlichen Patienten mit einem Körpergewicht größer als 90 kg zurückliefert, das Suchgewicht selbst an seine Bedürfnisse anpassen zu können. Stand jetzt muss für eine solche Funktion die Suche selbst vom Administrator angepasst werden.

5.5.2 Zusätzliche Verwendungsmöglichkeiten für Prozesergebnisse

Prozessergebnisse sollen in einer späteren Version auch als Platzhalter in DynamicTextComponents und Spalten in GridComponents verwendet werden können. Ein Anwendungsbeispiel ist die Berechnung des BMI (Body Mass Index) einer Person. Hierzu werden sowohl die Körpergröße als auch das Gewicht benötigt. Stand jetzt kann hierfür eine weiteres Attribut im zugehörigen Datentyp hinzugefügt werden, welche jedoch manuell bei jeder Änderung des Gewichts angepasst werden muss.

5.5.3 Mehrsprachigkeit

Eine weitere denkbare Erweiterung für zukünftige Versionen ist die Möglichkeit der Internationalisierung der statischen Informationen, damit die effiziente Zusammenarbeit in unterschiedlichen Sprachen ermöglicht wird.

5.5.4 Wiederverwendung von Komponenten

Um die Benutzung des GUI-Editors zu vereinfachen und beschleunigen sollte es irgendwann möglich sein, bereits modellierte GUI-Komponenten in neuen Seiten wiederzuverwenden. So ist es zum Beispiel denkbar, das Seitenlayout inklusive enthaltener Komponenten einmalig festzulegen und für alle Seiten zu verwenden. So muss lediglich der Inhalt neu erstellt werden. Im aktuellen Entwicklungsstand muss jede Seite vollständig von Grund auf erstellt werden.

Kapitel 6

Suche

In diesem Kapitel werden die Ergebnisse der DyWA Search Gruppe vorgestellt, um domänenspezifische Suchanfragen in einer modellierten Anwendung bereitzustellen. Zunächst erfolgt die Analyse der Anforderungen, bevor die umgesetzten Lösungen zum Erreichen dieser Anforderungen näher erläutert werden.

6.1 Anforderungsanalyse

In diesem Abschnitt werden zunächst die anfänglichen Anforderungen bezüglich der Suche in einer domänenspezifischen Anwendung erläutert.

6.1.1 Aufgabenbeschreibung

Im ersten Teil der Projektgruppen-Phase bestand die Aufgabe darin, sich mit der aktuellen Struktur der Datenbank und ihrer Verbindung zur domänenspezifischen Anwendung vertraut zu machen. Hierzu implementierte die Gruppe zunächst eine einfache Suchseite mit Filterfunktionen. Im zweiten Semester wurde eine domänenspezifische API (**Search-API**) implementiert, die es ermöglicht, Datenbankabfragen zu erstellen, die mit den domänenspezifischen Klassen arbeiten und nicht mehr direkt mit dem Metamodell der DyWA in Beziehung stehen. In erster Linie soll die API das Metaschema verstecken und eine Verbindung zwischen der domänenspezifischen Anwendung bzw. ihre Klassen und dem Metaschema der DyWA schaffen. Ziel ist es, das Backend somit zu modularisieren und die zugrundeliegende Datenbank austauschbar zu machen. Weiterhin wurde mit Hilfe dieser neuen API eine Erweiterung eingebaut, die es dem Domänenexperten erlaubt, Datenbankabfragen über eine GUI zu erstellen und abzuspeichern. Diese Datenbank-Abfragen werden dem Domänenexperten als **Suchen** bereitgestellt. Diese Suchen können auf Seiten der Anwendung verwendet werden, um die Objekte der Datenbank in der domänenspezifischen Anwendung zu filtern und darzustellen.

6.2 Verwendete Technologien

In diesem Abschnitt werden Technologien erläutert, die hauptsächlich verwendet werden, um die Implementierung umzusetzen.

6.2.1 JPA Criteria API

Um Datenbankabfragen mit Hilfe von existierenden Klassen durchführen zu können, nutzen wir die *JPA Criteria API* [10]. Mit dieser API wird es ermöglicht, einfache und auch komplexere Datenbankabfragen in Objekten zu kapseln und zu verschachteln. In Listing 6.1 sieht man eine Beispielfunktion, die als Ergebnis Objekte des Typs `DBObject` liefert. Die Funktion erhält als Eingabeparameter einen Such-String und die ID eines `DBField`s. Durch die Criteria-API kann eine simple Abfrage mit einer Join-Operation über die Tabellen, die zu den Klassen `DBObject` und `DBField` gehören, erzeugt werden.

Listing 6.1: Datenbankabfrage mit Join-Operation

```
public Set<DBObject> getObjectsByFieldAndQuery(int fieldID, String searchQuery)
    {
  CriteriaBuilder builder = this.entityManager.getCriteriaBuilder();
  CriteriaQuery<DBObject> query = builder.createQuery(DBObject.class);
  Root<DBObject> dbObjectRoot = query.from(DBObject.class);
  Join<DBObject, DBProperties> dbPropertiesJoin =
      dbObjectRoot.join(DBObject_.properties);
  Join<DBProperties, DBStringProperty> dbStringPropertyJoin =
      dbPropertiesJoin.join(DBProperties_.stringProperties);

  query.select(dbObjectRoot)
    .where(
      builder.equal(dbStringPropertyJoin.get(DBStringProperty_.key), fieldID),
      builder.like(builder.lower(dbStringPropertyJoin.get(DBStringProperty_.value)),
          ("%" + searchQuery + "%").toLowerCase()));

  return new HashSet<>(this.entityManager.createQuery(query).getResultList());
}
```

6.2.2 Hibernate Search

Hibernate Search [11] ist eine API, die es ermöglicht, Volltext-Suchen auf Datenbanken mittels Hibernate durchzuführen. In der Regel werden die zu indizierenden Klassen lediglich um passende Annotationen erweitert. Diese Annotationen legen fest, ob und in welcher Form die Klasse für kommende Volltext-Suchen indiziert werden sollen. Da in der DyWA die Klassen erst im späteren Verlauf generiert werden, hat man sich dazu entschieden, die *Programmatic API* von Hibernate Search zu verwenden. Mittels dieser API kann eine `SearchMappingFactory` erstellt werden, in der die Indizierungen konfiguriert werden. In Listing 6.2 sieht man die Basis-Konfiguration einer solchen Factory.

Listing 6.2: Auszug aus der *SearchMappingFactory*

```
final SearchMapping mapping = new SearchMapping();
mapping.analyzerDef("ngram", StandardTokenizerFactory.class)
  .filter(LowerCaseFilterFactory.class)
```

```
        .filter(NGramFilterFactory.class).param("minGramSize", "3")
          .param("maxGramSize", "3");
    final IndexedMapping dbIdentifiedMapping =
        mapping.entity(DBIdentified.class).indexed();
```

Lucene

Bei Lucene handelt es sich um eine in Java geschriebene Bibliothek für die Volltext-Suche, die in Hibernate Search verwendet wird. Es ist möglich, eine Volltext-Indizierung für Datenbankobjekte und Dokumente durchzuführen. Dabei handelt es sich nicht um eine vorgefertigte Suchmaschine. Es werden lediglich Funktionen und Klassen zur Verfügung gestellt, die von Entwicklern genutzt werden können, um eigene Suchen zu realisieren. Die Realisierung verläuft in zwei Schritten. Zuerst müssen alle Texte bzw. Dateien indiziert werden. Nach der Indizierung kann mit einer Suchanfrage auf diesen gearbeitet werden.

Indizierung

Hibernate Search ist in der Lage, jede Entität der Datenbank, die über Hibernate persistiert und geändert wurde, zu indizieren. Bei der Indizierung werden alle Werte einer Entität als String in einer Datei oder im Zwischenspeicher abgelegt. In der Entwicklungsphase sollen die generierten Indizes zunächst beim Starten des Servers neu erstellt und im Zwischenspeicher gelagert werden. In der Produktionsphase hingegen können die Indizes lokal gespeichert werden, um einen effizienteren Zugriff zu ermöglichen (siehe Listing 6.3). Zudem ist es dann nicht mehr nötig, die Indizes nach jedem Neustart des Server neu zu erstellen. Hibernate Search beobachtet jede indizierte Entität und aktualisiert diese im Hintergrund.

Listing 6.3: Index im Arbeitsspeicher lagern

```
<properties>
  <property name="hibernate.hbm2ddl.auto" value="update"/>
  <property name="hibernate.jdbc.batch_size" value="64"/>
  <property name="hibernate.default_batch_fetch_size" value="64"/>
  <property name="hibernate.dialect"
      value="de.ls5.commons.hibernate.dialects.UnlimitedPostgreSQLDialect"/>
  <property name="hibernate.generate_statistics" value="false"/>
  <property name="hibernate.cache.use_second_level_cache" value="false"/>
  <property name="hibernate.cache.use_query_cache" value="false"/>
  <property name="hibernate.show_sql" value="false"/>
  <property name="hibernate.format_sql" value="false"/>
  <property name="hibernate.use_sql_comments" value="false"/>
  <property name="hibernate.search.default.directory_provider" value="ram"/>
  <property name="hibernate.search.model_mapping"
      value="de.ls5.pg.probio.search.SearchMappingFactory"/>
</properties>
```

Bridge

Bei der Indizierung werden die Objekte auf eine Zeichenfolge „heruntergebrochen“ und in Text-Dateien gespeichert. Objekte können jedoch auch weitere Objekte beinhalten und eine große Tiefe erzeugen. Bei komplexeren Objekten ist es daher nötig,

die maximale Tiefe und die Art der Indizierung näher zu spezifizieren. Hibernate Search bietet dazu die Funktionalität der *Bridges*. *Bridges* sind Adapter, mit denen beschrieben werden kann, in welcher Form bestimmte Attribute eines Objekts gespeichert werden sollen. In den Listings 6.4 und 6.5 sieht man die Implementierung einer Bridge für die Klasse `DBType`. Diese Bridge wird verwendet, um `DBObjects` indizieren zu können und gleichzeitig die Informationen über den `DBType` zu behalten.

Listing 6.4: Beispiel einer Bridge für DBTypes

```
public class DBTypeBridge implements StringBridge {
    @Override
    public String objectToString(Object o) {
        DBType object = (DBType) o;
        return object.getName();
    }
}
```

Listing 6.5: Anwendung der Bridge

```
dbObjectMapping.property("type",
    ElementType.FIELD).field().bridge(DBTypeBridge.class);
```

6.2.3 Wahl der Technologie

Lucene bietet viele Vorteile die dazu führten, sich für diese Bibliothek zu entscheiden:

- Sie ist eine mächtige und plattformübergreifende Bibliothek, mit der eine schnelle und effiziente Suche möglich ist.
- Die API ist sehr einfach und verständlich. Entwickler können zum Beispiel neue Dateitypen hinzufügen und Interfaces entwickeln.
- Die Indizierung ist skalierbar und schnell bei gleichzeitigem, geringen RAM-Bedarf. Die Daten werden auf bis zu 30% der Ursprungsgröße verkleinert.
- Die Suche auf den Indizes kann bei Bedarf gewichtet werden

Aufgrund der geringen Komplexität und Erweiterbarkeit eignet sich Lucene für DyWA-Search. Der wichtigste Punkt ist jedoch die Erweiterbarkeit. Da noch nicht ganz klar ist, wie die Suche am Ende genau aussehen wird, ist es wichtig, schnellstmöglich auf Änderungen reagieren zu können.

6.2.4 Vorbereitung

Zunächst setzte sich die Gruppe mit der DyWA, im Speziellen mit der Datenbankstruktur und den domänenspezifischen Modellen auseinander. Hierzu wurde mit Hilfe der Criteria API[10] und Hibernate Search[11] eine kleine Suchseite implementiert, die es erlaubt, auf den Objekten und Typen der DyWA zu suchen.

6.3 Einfache Suche

Als erste Umsetzung wurde eine Testseite mit einer einfachen Suchfunktion erstellt. Auf dieser ist es möglich, nach Typen und/oder Objekten zu suchen (siehe Abbildungen 6.1 und 6.2). Die Funktionen werden durch ein Interface `SearchController` bereitgestellt. Je nachdem was gesucht wird, arbeitet der Controller mit den Typen `DBNamed`, `DBObject` oder `DBType` (siehe Abbildung 6.3).

Abbildung 6.1: Frontend der Such-Testseite

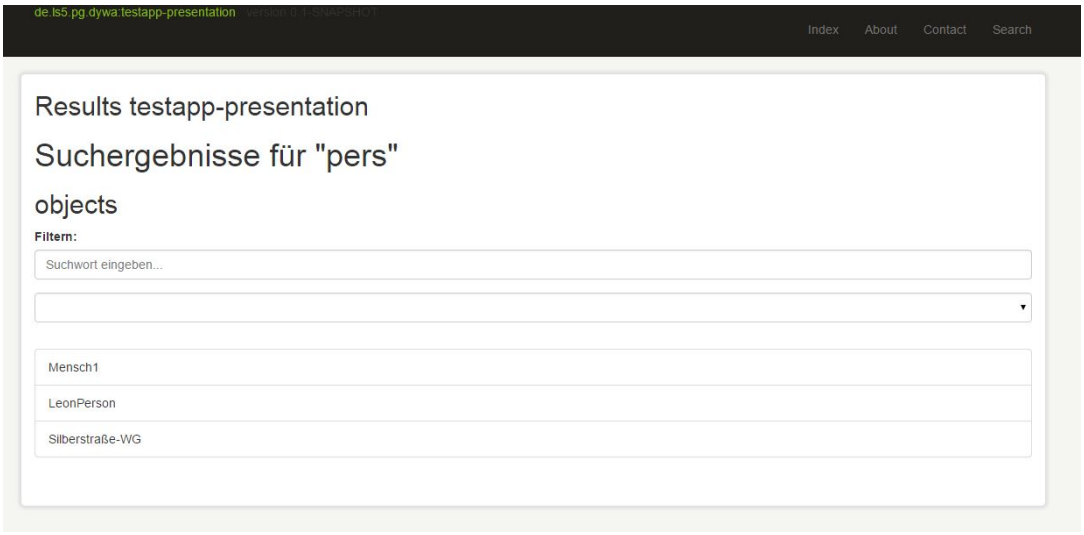

Abbildung 6.2: Frontend der Such-Testseite (Ergebnisse)

Methoden in der Implementierung des Controllers werden so umgesetzt, dass, je nach Datenbank auf der gearbeitet wird, nur die Suchanfrage geändert werden muss. Als Parameter dienten zunächst ein Suchwort und zwei boolesche Werte, die angeben, ob nach Typen oder Objekten gesucht wird.

6.3.1 Granularität

Nach der der ersten Umsetzung wurde überlegt, wie fein die Suche gestaltet werden soll. Dafür wurde zunächst die Struktur der Datenbank analysiert. Genutzt wird für die Datenbank ein Metaschema (siehe Grundlagen-Abschnitt zur DyWA). Um über

die Datenbank suchen zu können, muss mit den Typen `DBObject` und `DBType` gearbeitet werden. Es müssen die entsprechenden Controller und Klassen rausgefiltert werden, die von der DyWA erzeugt werden.

Für eine erste Suchfunktion wurden folgende Möglichkeiten festgelegt:

- Volltextsuche (`DBNamed`)
- Namen der Attribute und Oberklassen von Typen
- Werte und Namen der Attribute und Typen der Objekte

Hierfür werden im `SearchController` weitere Methoden angelegt. Es wurde auch ein weiterer Controller `SearchParameterController` erstellt, um die Parameter zu verwalten. Diese Parameter werden in einem Objekt vom Typ `SearchParameter` gespeichert.

Die Funktionen, die die Parameter verwalten, werden vom `SearchController` entkoppelt, da dieser nur mit der Datenbank arbeiten soll. Die Parameter und Ergebnisse werden zwischen den beiden Controllern ausgetauscht.

6.3.2 Architektur

In Abbildung 6.3 ist die Architektur der Suche veranschaulicht. Die folgenden Klassen haben dabei eine zentrale Rolle.

SearchController Interface, welches die grundlegenden Suchfunktionen wie *getObjectsByQuery*, *getTypesByQuery*, etc. enthält.

SearchControllerImpl Beispiel-Implementierung des `SearchControllers`

SearchParameter Kapselung der möglichen Suchparameter, wie z.B. Suchbegriff, Typ des Rückgabewertes und weiterer Restriktionen.

Bei einer Suchanfrage wird in der Klasse `Search` ein Searchparameter erzeugt und mit diesem durch den `SearchController` die Suche durchgeführt. In dem Search-Objekt wird spezifiziert, ob und welche Typen und Attribute ebenfalls gesucht werden sollen. Dies geschieht in Form einer Liste, in der die IDs, unter denen Typen und Attribute gespeichert werden, mit übergeben werden (siehe Grundlagen-Abschnitt zur DyWA). Die gefundenen Objekte und Typen werden an die Ergebnis-Seite weitergeleitet und angezeigt.

Search und Results sind dabei Klassen, die von Tapestry zur Darstellung im Frontend genutzt werden. Diese verarbeiten die übergebenen Daten wie die Suchergebnisse und zeigen diese entsprechend an.

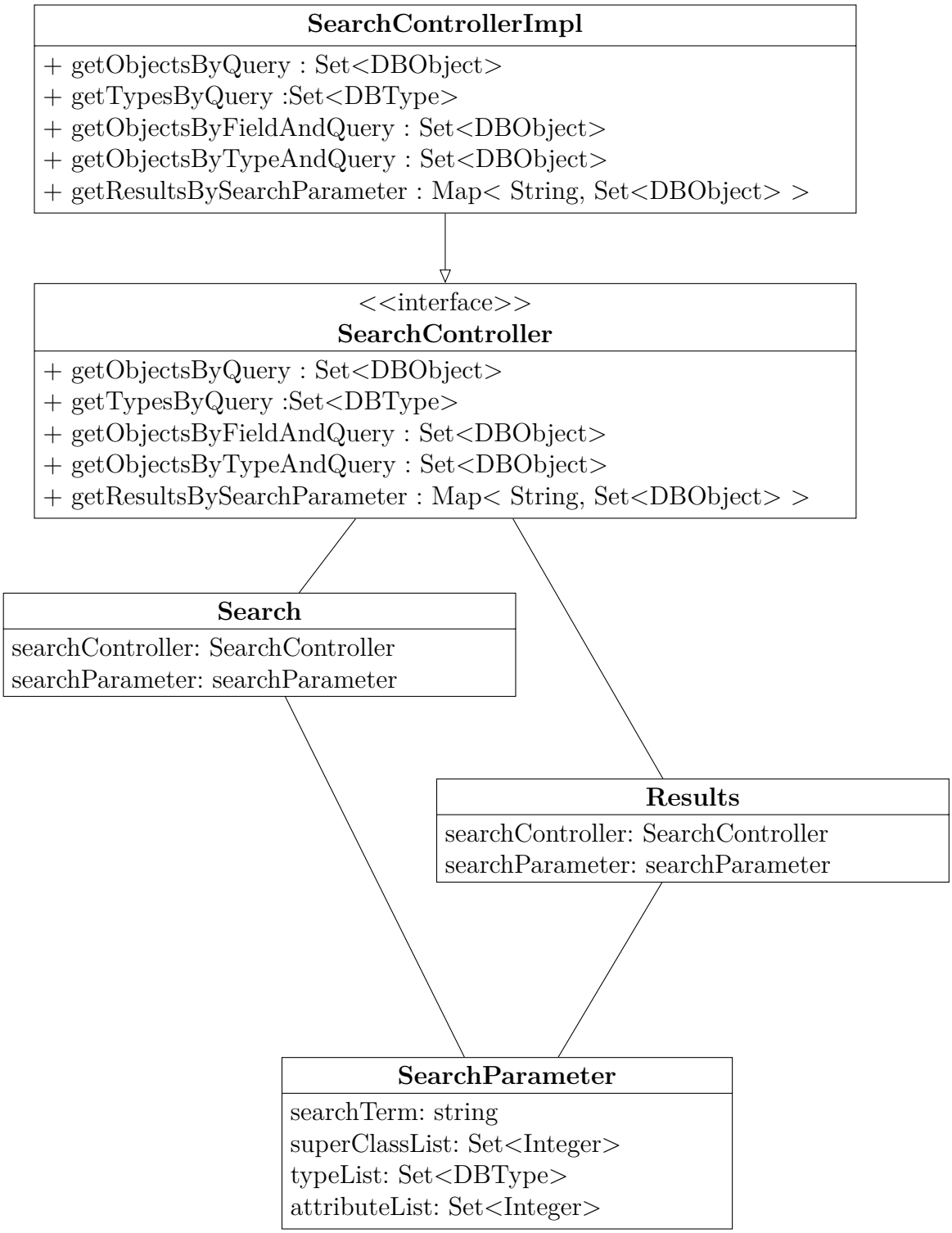

Abbildung 6.3: Architektur der Test-Umgebung

6.4 Interne Search-API

Im Laufe der ersten Phase wurde deutlich, dass die DyWA eine komplexe Grundstruktur bezüglich ihrer Modelle besitzt. Dies bedeutet, dass das Metamodell der DyWA noch keine klare Abgrenzung zu den domänenspezifischen Klassen besitzt. Im Gegensatz zum Anwender, der die tatsächlichen Typen und Objekte sieht, die vom Anwendungsentwickler erstellt wurden, ist der Entwickler, bis auf einige Ausnahmen noch dazu gezwungen, komplexe Datenbank-Abfragen zu erstellen, um die Objekte der Endanwendung zu bearbeiten oder auf diese zuzugreifen. Um dies in Zukunft zu vereinfachen, musste eine Möglichkeit geschaffen werden, die DyWA-Suche unabhängig von ihrer zugrundeliegenden Datenbank zu machen. Um die Anwendung von der Datenbank zu lösen und den Umgang mit den Modellen der domänenspezifischen Anwendung auch für den Entwickler zu vereinfachen, entwickelte man im zweiten Teil der Arbeitsphase eine API, die diese Abstraktion ermöglicht. Im Idealfall soll die API jegliche Datenbank-Abfragen in der gesamten Anwendung übernehmen. Das Ziel dieser API ist es, die internen Schichten der DyWA zu verstecken und jedem Entwickler die Möglichkeit zu geben, Objekte und Typen der DyWA sowohl als `DBObjects` oder `DBTypes` zu bekommen oder als Objekte der domänenspezifischen Anwendung wie z. B. vom Typ `Patient`. Somit kann diese Search-API als Datenbank-Abfragesprache für die Modelle der DyWA-Anwendung interpretiert werden. In Abb. 6.4 werden die angesprochenen Ebenen grafisch veranschaulicht.

6.4.1 Generierung der domänenspezifischen Klassen

Um die Search-API benutzen zu können, werden zunächst Klassen benötigt, die die Typen der Endanwendung repräsentieren. Hierzu wurde ein Generator implementiert, der über die Domänenexperten modellierten Typen iteriert und für jeden Typ eine entsprechende Klasse mit der Bezeichnung der Form `TYPNAME_` erzeugt. In dieser Klasse wird die tatsächliche *ID* und der Name des Typen zur späteren Referenzierung abgelegt. Weiterhin wird für jedes Attribut des Typs ein statisches Attribut in der Klasse für die Verwendung in der API generiert. Somit kann man nun mit Hilfe der generierten Klassen in Anlehnung an das Criteria-Metamodell die domänenspezifischen Objekte auch auf der Ebene der Entwicklung nutzen.

6.4.2 Umsetzung der Search-API

Analog zur Criteria API ist es mit der neuen Search-API möglich, dynamische Datenbank-Abfragen zu erzeugen. Ein großer Vorteil dieser API ist es, dass die API mit domänenspezifischen Typen und Objekten parametrisiert werden kann. In diesem Fall bleibt dem Entwickler zusätzlich die komplexe Datenbank-Struktur verborgen. An dieser Stelle sind die generierten domänenspezifischen Klassen zu verwenden. Somit kann man unabhängig von der unterliegenden Datenbank-Struktur entwickeln.

Intern werden die Abfragen passend übersetzt. Hierbei werden die Referenzen, die innerhalb der generierten Klassen gespeichert wurden, verwendet. Mit den jeweiligen Beziehungen zu den passenden Ergebnissen der unterliegenden Datenbank

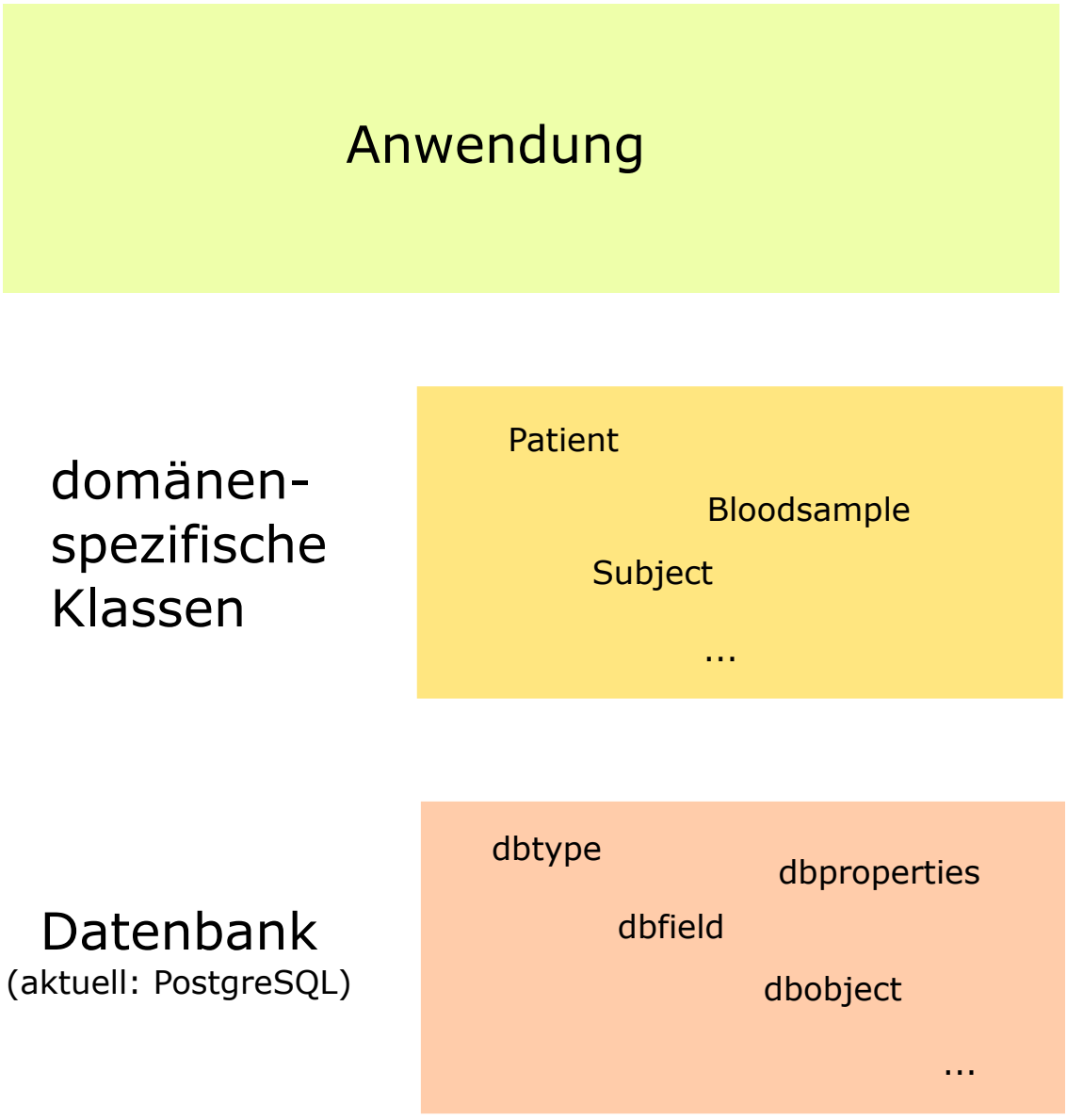

Abbildung 6.4: Ebenen der DyWA

generiert der Search-Codegenerator nun mit Hilfe der Criteria API die Datenbank-Abfragen auf dem Metaschema. Um weiterhin auf der Ebene der Domäne zu bleiben, werden die Ergebnisse der Criteria API passend zu domänenspezifischen Objekten transformiert, bevor die Ergebnisse zurückgegeben werden.

6.4.3 Verwendung der Search-API

Vor der Verwendung der Search-API basierten die Suchanfragen auf generierten `NamedQueries`. Ziel ist es aber, dass für die Queries Code generiert wird, der die domänenspezifische API nutzt. Die domänenspezifische API stellt danach die einzige Implementierung für das Ausführen von Suchen da.

Die implementierte Search-API ist angelehnt an die Criteria-API. Die Search-API wird in der Klasse `SearchControllerImpl` verwendet. Der für diese Klasse generierte Code verarbeitet die `DyWASearch`-Objekte so, dass der generierte Code die Suchanfragen über die Search-API ausführt.

Zuerst wird im `SearchBuilder` eine neue Query erstellt. Dieser Query wird als Ergebnis-Klasse der `DBType` aus dem `DyWASearch`-Objekt zugeordnet, welcher angibt, welche Art von Objekten aus der DyWA zurück geliefert werden sollen. Anschließend wird ein neuer `PathConstraintBuilder` gebaut. Dem `PathConstraintBuilder` werden dann die in den `DyWASearch`-Objekten vorliegenden Constraints hinzugefügt. Constraints sind Angaben, die die Suchergebnisse einschränken sollen, wie z. B. „wähle alle Patienten, für die gilt, dass ihre Körpergröße größer als 175 ist“. Sämtliche Constraints werden momentan mit Logisch-Und verknüpft. Ein logisches Oder ist in der API implementiert, wird aber noch nicht verwendet. Abschließend wird mit Hilfe des `PathConstraintBuilders` ein `PathConstraint`-Objekt gebaut, welches der über die API zu bauenden Query als Teil der Where-KLausel übergeben wird.

Anschließend muss noch die Map für eventuell verwendete Parameter erstellt werden. Parameter können in Constraints verwendet werden, um die Suche zur Laufzeit parametrisiert einschränken zu können. Die Map wird im generierten Code automatisch mit den ggf. vorhandenen Parametern befüllt.

Nach diesem Schritt ist die Suche bereit zur Ausführung und nutzt in ihrer Implementierung die Search-API. Dabei ist zu beachten, dass nur für die Queries Code generiert wird, welche im Search-Plugin auf den Status „aktiv“ gesetzt sind.

In Listing 6.6 ist beispielhaft der Code eingefügt, welcher eine Suchanfrage in der Klasse `SearchControllerImpl` implementiert. Der Code wurde aus Gründen der Lesbarkeit etwas vereinfacht, indem z. B. die Angabe des Packages `de.ls5.dywa.generated.entity` für `Patient` bzw. `Patient.class` weggelassen wurde.

Folglich können Entwickler mit Hilfe der neuen domänenspezifischen API elegantere Datenbank-Abfragen erzeugen, die auf einer höheren Ebene arbeiten und das Metaschema der DyWA verstecken. Zudem ist nun die Möglichkeit geschaffen, die DyWA so weit anzupassen, dass man die aktuell relationale Datenbank durch eine NoSQL-Datenbank ersetzen kann oder die DyWA mit mehreren Datenbanken parallel nutzen kann.

Listing 6.6: Vereinfachtes Codebeispiel für die Verwendung der API

```
public List<Patient> executeSearchPatientWeightMoreThan80kgQuery() {
    DyWAQuery<Patient> query = this.searchBuilder.createQuery(Patient.class);

    From<Patient> root = query.from(Patient.class);

    PathConstraintBuilder<Patient> pathConstraintBuilder =
        root.newPathConstraintBuilder();

    //Add Constraints:
    pathConstraintBuilder.addConstraints(
          pathConstraintBuilder.and(
          pathConstraintBuilder.gt(Patient_.gewicht, 80.0)));

    PathConstraint pathConstraint = pathConstraintBuilder.buildPathConstraint();
    query.where(pathConstraint);

    //parameterMap erzeugen:
    Map<String, Object> parameterMap = new HashMap<>();

    return this.searchBuilder.executeQuery(query, parameterMap);
}
```

6.5 Persistierbare Suchanfragen

Die erste Anwendung der neuen Search-API konnte mit der Entwicklung von persistierbaren Suchanfragen realisiert werden. Dem Domänenexperten ist es nun möglich, einfache und komplexe Suchanfragen selbst zu erstellen und dynamisch in modellierte Seiten einzubauen. Somit kann er direkt Datenbank-Abfragen über die domänenspezifischen Klassen erzeugen. Weiterhin können nun komplexe Abfragen vom Anwendungsentwickler modelliert werden, ohne selbst Code schreiben zu müssen.

6.5.1 Architektur der Suchanfragen

DyWASearch Erstellte Suchanfragen werden in der Datenbank als `DyWASearch`-Objekt gespeichert (Abb. 6.7). Es werden alle nötigen Informationen gespeichert die für eine Query nötig sind. Im `From` wird festgelegt auf welcher Tabelle gesucht werden soll. `DyWASearchComparison` und `DyWAOrder` realisieren WHERE-Klauseln und ein ORDER BY.

DyWASearchCriteriaBuilder Hier werden aus `DyWASearch`-Objekte Criteria-Querys erstellt, ausgeführt und die Ergebnisse zurückgegeben. Genutzt wird der CriteriaBuilder zum Beispiel für die Vorschau auf der Suchseite (Abb. 6.5).

SearchAdapter Der `SearchAdapter` (Abb. 6.6) stellt die vordefinierten Suchanfragen zur Verfügung, ohne dass dabei nach außen hin interne Klassen der API zurückgegeben werden. Dies garantiert die Plugin-Unabhängigkeit. Es werden Descriptor-Objekte zur Verfügung gestellt, die die `DyWASearch`-Objekte beschreiben. Diese Objekte können von der GUI für die Modellierung von Komponenten zur Einbettung der modellierten Suchen in die Anwendung genutzt werden.

DyWASearchCriteriaBuilder
setDyWASearch(DyWASearch dyWASearch): void executeQuery(DyWASearch query): List<DBObject> executeQuery(DyWASearch query, Integer maxResults, Integer offset): List<DBObject> appendWhere(Root<DBObject> objectRoot, DyWASearchComparison where, List<Predicate> predicates, Join<DBObject, DBProperties> dbPropertiesJoin): void appendWhere(Root<DBObject> root, List<DyWASearchComparison> where, List<Predicate> predicates, Join<DBObject, DBProperties> dbPropertiesJoin): void appendOrder(List<DyWAOrder> orderings): void getPropertyType(long fieldID): PropertyType addStringPredicate(List<Predicate> predicates, DyWASearchComparison where, Join<DBObject, DBProperties> dbPropertiesJoin): void addBooleanPredicate(List<Predicate> predicates, DyWASearchComparison where, Join<DBObject, DBProperties> dbPropertiesJoin): void addDoublePredicate(List<Predicate> predicates, DyWASearchComparison where, Join<DBObject, DBProperties> dbPropertiesJoin): void addLongPredicate(List<Predicate> predicates, DyWASearchComparison where, Join<DBObject, DBProperties> dbPropertiesJoin): void addTimestampPredicate(List<Predicate> predicates, DyWASearchComparison where, Join<DBObject, DBProperties> dbPropertiesJoin): void

Abbildung 6.5: DyWASearchCriteriaBuilder

SearchAdapter
isAdapterAvailable(): boolean getSearches(): Map<Long, SearchDescriptorImpl> getSearchesByTerm (String term): Map<Long, SearchDescriptorImpl> getSearchForId(Long id): SearchDescriptorImpl getSearchByName(String name): SearchDescriptorImpl getSearchByTerm(String term): SearchDescriptorImpl

Abbildung 6.6: SearchAdapter

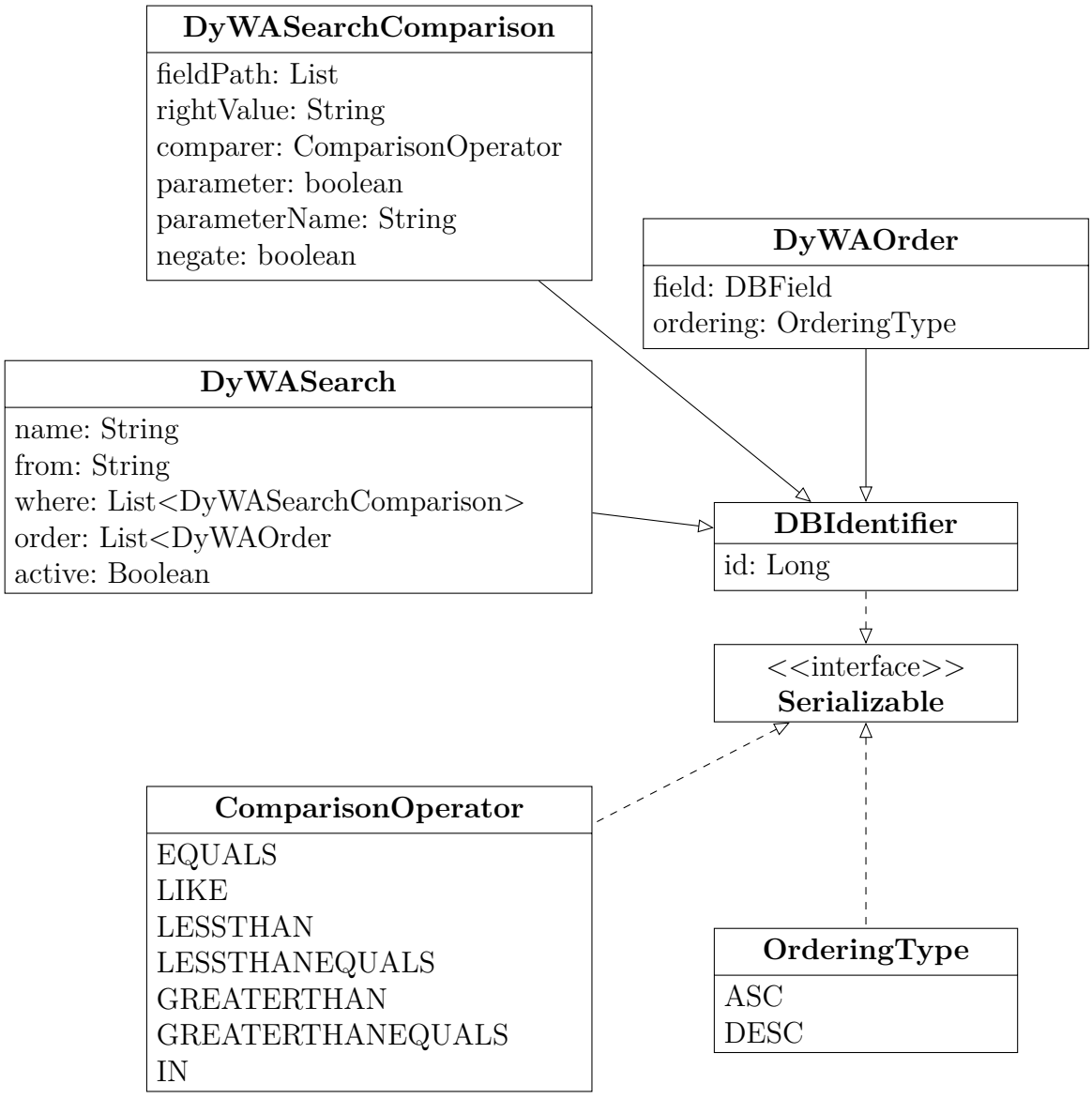

Abbildung 6.7: Vereinfachtes UML-Diagramm zu persistierbaren Suchanfragen

6.5.2 Verwendung der Suchanfagen

Um Suchen zu erstellen, klickt man innerhalb der DyWASearch-Plugins auf den Menüpunkt *Create Search*. Auf dieser Seite (Abb. 6.8) wird nun der neuen Suche ein Name gegeben und der Rückgabetyp festgelegt. Anschließend wird, basierend auf dem Rückgabetyp und den zugrundeliegenden Typen, eine oder mehrere Bedingungen definiert (Abb. 6.9). Unterhalb dieser Textfelder befindet sich eine Liste bereits vorhandener Bedingungen (Abb. 6.10).

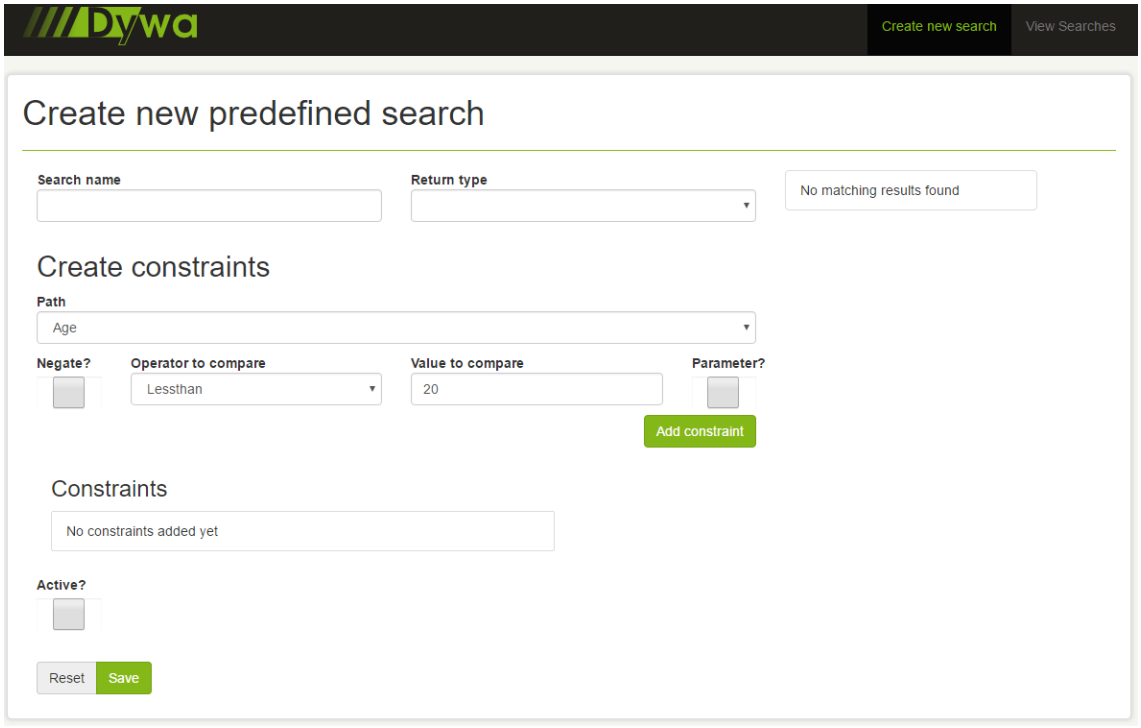

Abbildung 6.8: Suche erstellen

Es gibt derzeit zwei verschiedene Arten von Bedingungen:

statische Bedingung Der Wert des Vergleichs wird zur Modellierungszeit festgelegt.

dynamische Bedingung Der Wert des Vergleichs wird nicht zur Modellierungszeit gesetzt. Stattdessen muss ein Name für diesen Vergleich vergeben werden, der hinterher bei der Verwendung der Suche als Platzhalter für diesen Wert dient. Somit können Such-Anfragen dynamisch zur Laufzeit parametrisiert werden.

Auf der rechten Seite werden die Suchergebnisse (bzgl. der aktuellen Datenbank) als Vorschau angezeigt.

Bereits angelegte Suchen werden in einer Liste unter *View Searches* angezeigt (Abb. 6.11). Mit einem Klick auf ein Listenelement kann man diese ausgewählte Suche bearbeiten.

Mit dieser neuen Funktionalität können sich Domänenexperten nun individuelle Inhalte in der Anwendung anzeigen lassen. Mit Hilfe der anderen Erweiterungen dieser

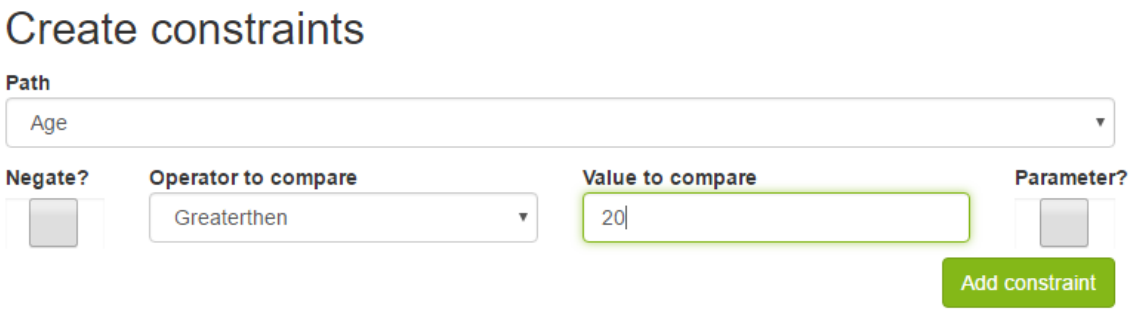

Abbildung 6.9: Bedingung hinzufügen

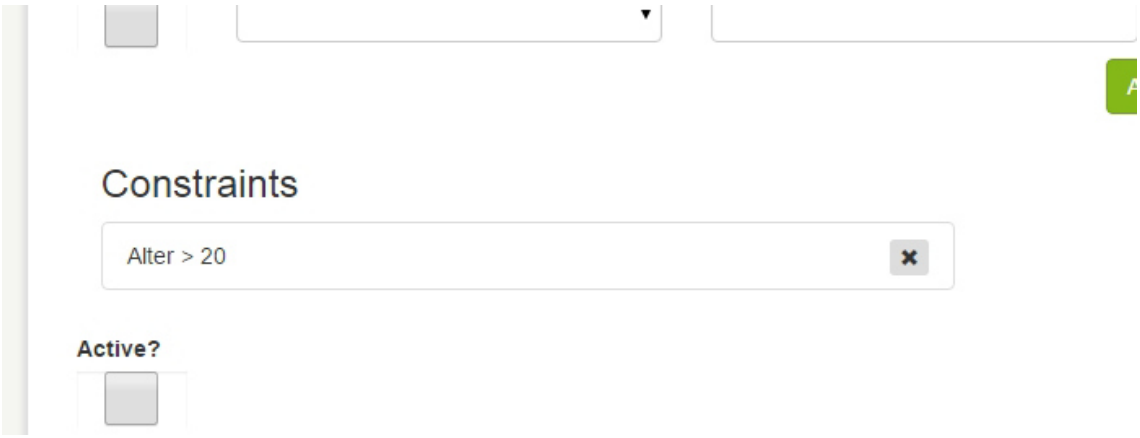

Abbildung 6.10: Liste der hinzugefügten Bedingungen

Projektgruppe sind z.B. verschiedene Listenansichten erstellbar, die abhängig von den Rechten des aktuellen Nutzers bestimmte Inhalte versteckt. Diese Filterung kann in Zukunft mit der Erstellung von spezifischen Suchanfragen und den damit verbundenen Prozessen und Absicherungen realisiert werden.

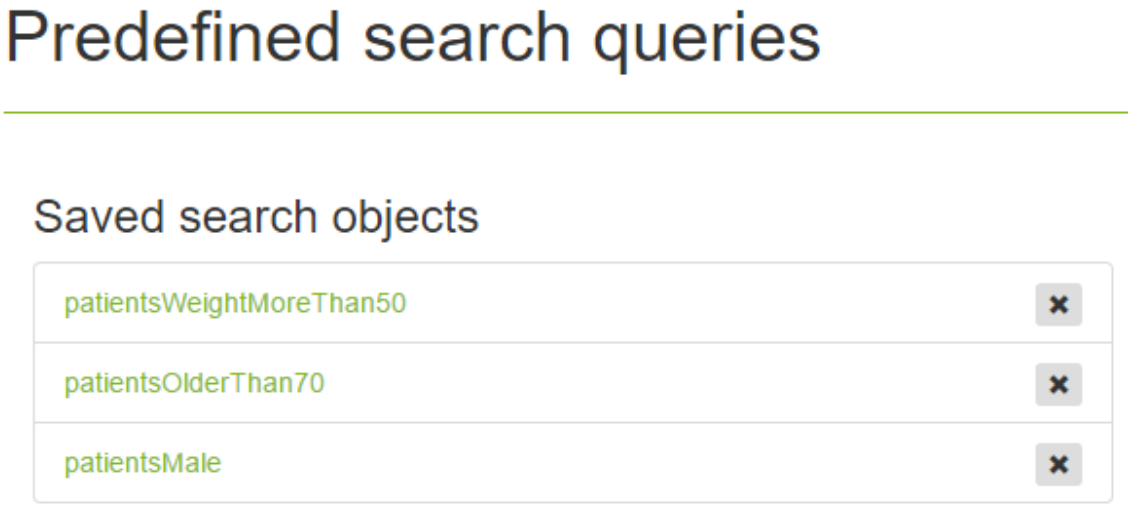

Abbildung 6.11: Liste persistierter Suchen

6.6 Zusammenfassung

Mit dem neuen DyWA-Search Plugin ist es nun möglich, Datenbankanfragen auf domänenspezifischen Klassen zu erstellen. Diese Anfragen müssen mit der neuen Search-API implementiert werden. Diese nutzen analog zur bekannten Criteria-API Objekte als Parameter. Aber im Gegensatz zur Criteria-API werden hier die domänenspezifischen Klassen verwendet, die vom DyWA-Search Plugin generiert werden.

Eine erste Implementierung dieser API wurde mit den persistierbaren Suchanfragen durchgeführt. Diese persistierbaren Suchanfragen ermöglichen dem Anwendungsentwickler einfache bis komplexe Abfragen über die GUI zu erstellen und zu nutzen. So kann der Anwendungsentwickler beispielsweise eine Suche erstellen, die ihm Patienten mit einer bestimmten Blutgruppe als Suchergebnis liefert. Diese Suche kann anschließend auf einer beliebigen Seite verwendet werden. Dort können die Ergebnisse als Liste ausgegeben und modifiziert werden.

6.7 Ausblick

In diesem Abschnitt sollen mögliche Erweiterungen des Plugins skizziert werden.

6.7.1 Vorschau der Suchergebnisse

Während der Erstellung einer Suche werden bereits Suchergebnisse als Vorschau angezeigt. Die Datenbank-Abfragen, die die Objekte für die Vorschau holen, verwenden derzeit noch direkt die Criteria API. Um eine völlige Unabhängigkeit von der Datenbank zu haben, ist es daher nötig, diese Abfragen ebenfalls mit der Search-API zu implementieren.

6.7.2 Volltextsuche

Damit die Datenbank in Zukunft effizienter verwenden werden kann, sollte die Verwendung von `Hibernate Search` in Erwägung gezogen werden. Mit Hilfe dieser Volltextsuche können Inhalte indiziert werden und v.a. Suchen über Textdateien und PDFs ermöglicht werden. `Hibernate Search` wurde bereits in der Vorbereitungsphase von der DyWA Search Gruppe verwendet. Die Ergebnisse wurden jedoch nicht direkt in das Plugin übernommen.

6.7.3 Erweiterung der Such-Optionen

Die Möglichkeiten der Such-Optionen sind bislang teilweise noch stark eingeschränkt und können in Zukunft erweitert werden.

Konditionalisierung der Constraints

Die Constraints einer Suche lassen sich derzeit nicht als OR-Constraint hinzufügen. Hinzugefügte Constraints werden mit einem AND verknüpft. Anfragen wie $\mathsf{Alter} > 40\ \mathsf{OR}\ \mathsf{Alter} < 50$ lassen sich also für die persistierbaren Suchen noch nicht realisieren. Es ist auch noch nicht möglich, über Tabellen einen JOIN zu vollziehen. Die Entität `DyWASearch` müsste hinsichtlich mehrerer FROM-Elemente erweitert werden. Im `DyWASearchCriteriaBuilder` müssten auch die Methoden zum Erstellen einer Query angepasst werden. Basierend auf solchen Änderungen wären zudem leichte Anpassungen im Codegenerator nötig.

Erweiterung der GUI

Derzeit werden nur grundlegende Funktionen bei der Erstellung von Constraints unterstützt. So kann man in der GUI beispielsweise noch keine Constraints hinzufügen, die Werte mit Objekten oder Listen vergleichen können. Diese Funktionalitäten werden aber bereits von der internen Search-API unterstützt, so dass diese Funktionalitäten lediglich im Frontend implementiert werden müssen. Wenn die Möglichkeit von OR-Verknüpfungen und JOINs implementiert wird, muss auch dementsprechend die GUI erweitert werden.

Kapitel 7

Sicherheit

In diesem Kapitel geht es um die Arbeit der Untergruppe „Security“ der PG ProBio. Die Gruppe betrachtet vor allem Aspekte bezüglich der Sicherheit, darunter fällt vor allem die Zugriffskontrolle einzelner Nutzer, bei der verschiedene Benutzer unterschiedliche Rechte haben und nur für sie bestimmten Daten einsehen dürfen.

7.1 Ausgangssituation

Authentifikation und Autorisierung sind die grundlegenden Funktionen einer funktionierenden Zugriffskontrolle. Die Implementierung dieser Funktionen ist für eine Anwendung im biomedizinischen Umfeld unabdingbar, da sensible Daten, beispielsweise Patientendaten, vor unbefugten Zugriffen geschützt werden müssen. Zu Beginn der Projektgruppenarbeit bieten weder die DyWA noch das jABC Möglichkeiten zur Modellierung und Einbeziehung von Security-Aspekten während der Domänenmodellierung bzw. der Prozessmodellierung.

Die Untergruppe „Security“ der PG nimmt sich diesen Herausforderungen an. Dazu wurden im ersten Schritt die Anforderungen an die Security-Aspekte analysiert.

7.2 Anforderungsanalyse

Aus der oben genannten Analyse der Ausgangssituation ergibt sich für uns die Aufgabe, sämtliche für Sicherheit relevanten Funktionen zu implementieren. Ziel ist es, dass wir für die spätere Anwendung die Schutzziele *Vertraulichkeit* und *Integrität* sicherstellen können. Dafür muss der Zugriff auf Daten und Prozesse durch geeignete Mechanismen kontrolliert werden können. Unter *Zugriffskontrolle* versteht man den Zugriff auf bestimmte Ressourcen mit vorangegangener Überwachung und Steuerung, so dass unberechtigte Zugriffe verhindert werden können. Die zu implementierende Zugriffskontrolle soll es ermöglichen, dass der Zugriff auf bestimmte Daten und Prozesse geschützt werden kann. Dazu muss ein geeignetes Rechtemodell (siehe Abschnitt 7.3) implementiert werden. Später soll die Modellierung der Zugriffskon-

trolle unter Nutzung der DyWA bzw. des jABC möglich sein, also in den Prozess der Domänenmodellierung und der Formalisierung von Prozessen eingebunden sein.

Das Sicherheitsziel Integrität wird in den Information Technology Security Evaluation Criteria, kurz ITSEC, als „Schutz vor unbefugter Veränderung von Informationen“ [13] definiert. Über die Verhinderung von unautorisierten Modifikationen hinaus ist das Sicherheitsziel der Vertraulichkeit zu erfüllen. Dabei ist Vertraulichkeit als „Schutz vor unbefugter Preisgabe von Informationen“ [13] definiert.

Im Bereich der Biomedizinanwendungen sind für uns aus der Sicht der Sicherheit vor allem die Daten bzgl. Integrität und Vertraulichkeit zu schützen. Aus diesem Grund soll die spätere Webanwendung dazu in der Lage sein, den jeweiligen Nutzer zu authentifizieren. Auf Grundlage der festgestellten Identität des Anwenders wird die Autorisierung bzgl. des Zugriffs auf Daten und Prozesse abschließend bestimmt. Da die spätere Anwendung im Bereich der Biomedizin eingesetzt werden soll, ist die Zugriffskontrolle auf die zum Großteil sensiblen biomedizinischen Daten (z. B. Patientendaten) unabdingbar.

Ein weiteres wichtiges Ziel der Informationssicherheit ist das Erfüllen des Schutzziels *Verfügbarkeit*. Verfügbarkeit ist nach [13] definiert als „Schutz vor unbefugter Vorenthaltung von Informationen oder Betriebsmitteln“. Die Erfüllung dieses Schutzziels ist für uns zunächst nicht von Relevanz, da die Verfügbarkeit der späteren Anwendung nicht alleine abhängig ist von unserer Arbeit und zu viele externe Faktoren, wie die zugrunde liegende Datenbank oder der verwendete Server, eine Sicherstellung des Ziels Verfügbarkeit für uns nicht ermöglichen würden.

Um die Schutzziele *Vertraulichkeit* und *Integrität* sicher zu stellen, wird ein bereits vorhandenes Sicherheitsframework für unsere Arbeit genutzt werden. Mögliche Frameworks für diesen Zweck werden in Abschnitt 7.4 vorgestellt und evaluiert. Durch die Verwendung eines bereits bestehenden Frameworks für die Implementierung unserer Ziele soll eine möglichst fehlerfreie und mächtige Erfüllung der Anforderungen erreicht werden.

7.3 Rechtemodelle

Bei den u. g. Rechtemodellen handelt es sich um die bekanntesten Arten, Dateien oder Bereiche vor unberechtigten Zugriffen zu schützen. Die Besonderheiten und Eigenheiten werden im Detail in den jeweiligen Abschnitten beschrieben und ihre Vor- und Nachteile erläutert. Ziel ist es, eine dynamische Zugriffskontrolle zu implementieren, die es ermöglicht, Daten ohne vorherige Festlegung vor unberechtigten Zugriffen zu schützen. Als Beispiel wäre ein Experiment im Umfeld der Biomedizin zu nennen, bei dem der Ersteller alleinigen Zugriff erhält und diesen anderen nicht erst explizit verbieten muss.

7.3.1 Rollenbasierte Zugriffskontrolle

Die rollenbasierte Zugriffskontrolle, abgekürzt auch RBAC für „Role-Based Access Control“, ist ein Verfahren zur Steuerung und Kontrolle auf Dateien oder Dienste.

Das RBAC-Modell wurde 1992 von David F. Ferraiolo and D. Richard Kuhn in [5] vorgestellt.

Der Ansatz der rollenbasierten Zugriffskontrolle nutzt das Konzept der Rolle, um den Zugriff von Benutzern auf bestimmte Dienste zu erlauben. Dabei abstrahiert das Modell das Konzept der Rolle von Arbeitsprozessen. Grundgedanke war, dass jeder Benutzer einer oder mehrerer Rollen zugeordnet werden kann. Dabei können die Rollen mit den Arbeitsprozessen weitgehend übereinstimmen. Gibt es den Posten des Systemadministrators und des Netzwerkadministrators, könnte einem Administrator eine der beiden Rollen oder beide zugewiesen werden. Beide Rollen könnten dem Nutzer unterschiedliche Rechte zuordnen. Die Rolle wird somit vom System als Basis für die Zugriffskontrolle verwendet. Anders ausgedrückt, bestimmt die Rolle die Rechte des Benutzers.

Vorteil der Zuordnung über Rollen ist, dass neuen Nutzern direkt durch Zuweisung einer bestehen Rolle, Zugriffsrechte zugeteilt werden können. So kann einem neuen Benutzer, der z. B. Arzt ist, die Rolle „Arzt“ zugeordnet werden und der neue Benutzer erhält darüber direkt die der Rolle „Arzt“ zugeordneten Zugriffsrechte und -möglichkeiten. RBAC ist ein relativ intuitives Modell zur Zugriffskontrolle, da die Rollen aus den Arbeitsprozessen abstrahiert werden können. Die Zuordnung aller Ärzte zur Rolle „Arzt“ ist offensichtlich. Darüber hinaus können die Rollen natürlich auch noch feingliedriger erstellt werden. So wäre z. B. die Zuordnung eines neuen Arztes zu den Rollen „Oberarzt“, „Notarzt“ und „Kinderarzt“ möglich.

Die Nachteile am RBAC-Modell sind, dass die Zuordnung spezieller Zugriffsrechte an einzelne Benutzer nicht einfach erfolgen kann und das die Komplexität auf Grund einer großen Menge an Rollen steigen kann. Die Zuweisung der Rechte über Rollen macht es notwendig, das Nutzern, welche spezielle Zugriffsrechte erhalten sollen, diese nur über die explizite Anpassung der Rollen erhalten können. Dies würde aber zur Folge haben, dass alle Nutzer mit dieser Rolle ebenfalls dieses Zugriffsrecht erhalten würden. Da dies nicht gewünscht ist, müsste für den Nutzer eine spezielle Rolle angelegt werden, welche dieses exklusive Zugriffsrecht dem Nutzer zuweisen würde. Eine dynamische und nutzerspezifische Zuweisung von Zugriffsrechten ist daher nicht möglich. An diesem Fall lässt sich bereits die Komplexität ansatzweise erkennen. Eine feingranulare Zuteilung von Rechten auf Grundlage von Rollenzugehörigkeiten kann eine enorme Anzahl an Rollen nach sich ziehen. Den Überblick über die einem Nutzer zugeordneten Zugriffsrechte zu behalten ist schwer, da die Rollen des Nutzers bezüglich der Zugriffsrechte betrachtet werden müssen. Eine kontextbezogene Rechteverteilung ist mit dem RABC-Modell nicht möglich, wodurch es für unsere Vorstellung bezüglich der Umsetzung uninteressant ist.

7.3.2 Benutzerbestimmbare Zugriffskontrolle

Benutzerbestimmbare Zugriffskontrolle beschreibt ein Sicherheitskonzept, die die Zugriffsberechtigungen anhand der Identität des Nutzers überprüft. Bekannt ist die benutzerbestimmbare Zugriffskontrolle unter dem Kürzel DAC. Dieses steht für **D**iscretionary **A**ccess **C**ontrol. Zugriffsrechte werden immer pro Benutzer festgelegt. Das DAC-Konzept sieht eine Rechteweitergabe durch den Besitzer an andere

Benutzer vor. Eine Abschwächung des Konzeptes wird durch die Verwendung von Benutzerrolle und -gruppen erreicht, siehe dazu Abschnitt 7.3.1. Die Festlegung der Zugriffsrechte pro Benutzer wird bei einer großen Anzahl an Nutzern schnell unübersichtlich und erfordert durch den großen Verwaltungsaufwand eine Menge Zeit. Bei der Autorisierung kann mit negativen und positiven Regeln gearbeitet werden, welche die Komplexität weiter erhöhen. Außerdem ist die Weitergabe von sensiblen Informationen durch einen autorisierten Nutzer möglich, da keine Einschränkung erfolgt, wie diese genutzt werden dürfen bzw. können [20]. Aus den gleichen Gründen wie in Abschnitt 7.3.1 ist die benutzerbestimmbare Zugriffskontrolle für unsere Vorstellung bezüglich der Umsetzung unser Sicherheitsanforderungen nicht von Interesse.

7.3.3 Verbindliche Zugriffskontrolle

Regelbasierte Zugriffskontrolle beschreibt eine Zugriffskontrollstrategie, die die Entscheidung über eine Zugriffsberechtigung nicht nur auf Grundlage des Nutzers und der Ressource, auf die zugegriffen werden soll, trifft, sondern als Basis ebenfalls noch zusätzliche Regeln betrachtet [14]. Bekannt ist die regelbasierte Zugriffskontrolle unter der Abkürzung MAC. MAC steht hierbei für **M**andatory **A**ccess **C**ontrol. Die zusätzlichen Regeln können umfassen, dass zusätzlich eine Zuweisung von Labels oder eine Zuordnung zu Kategorien erfolgt. Die Labels oder die Kategorien werden in der Entscheidung auf Zugriffsberechtigung ebenfalls regelbasiert betrachtet und in die Zuteilung oder Ablehnung der Zugriffsanfrage mit einbezogen.

Im Wesentlichen existieren zwei Arten von MAC-Konzepten. Einerseits sind dies die Multi-Level-Sicherheitssysteme. Diese bilden verschiedene Schutzstufen ab. Dabei wird jede Ressource und jeder Nutzer einer Schutzstufe zugeordnet. Der Nutzer erhält nur dann Zugriff, wenn seine Sicherheitsstufe mindestens so hoch ist wie die Sicherheitsstufe der Ressource, auf welche der Nutzer zugreifen möchte. Im anderen Fall handelt es sich bei dem umgesetzten MAC-Konzept um ein sogenanntes multilaterales Sicherheitsmodell. Diese ermöglichen nicht nur eine vertikale Gliederung anhand von Schutzstufen, sondern ermöglichen auch eine Zuordnung zu sogenannten Verbänden. Ein Verband umfasst dabei mehrere Schutzstufen.

Die regelbasierte Zugriffskontrolle würde unseren Anforderungen an eine Zugriffskontrolle nicht gerecht werden können. Regelbasierte Zugriffskontrolle ist dann sinnvoll, wenn ein Zugriff auf Grundlage von Schutzstufen festgelegt werden könnte. Regelbasierte Zugriffskontrolle wäre nicht einfach genug vom Anwender der späteren Software handhabbar. Außerdem wäre die Unterteilung in Schutzstufen nicht passend und entweder nur zu grob oder zu fein möglich.

7.3.4 Attributbasierte Zugriffskontrolle

Das Ziel, eine dynamische Zugriffskontrolle zu implementieren, die der späteren Anwendung ermöglicht, Zugriffsrechte unabhängig von den dynamischen Sicherheitsbereichen festzulegen und Daten, welche noch nicht im System sind, zu schützen, sollte mit Hilfe der attributbasierten Zugriffskontrolle [28], auch **A**ttribute-**B**ased **A**ccess

Control (kurz: ABAC), gelingen, da dieses Sicherheitsmodell nicht auf den simplen und statischen Methoden basiert wie z. B. das rollenbasierte Zugriffsmodell. Um das Verständnis für ein attributbasiertes Zugriffskontrollmodell zu erhöhen, erfolgt eine kurze Definition von Attributtypen:

- **Subject:** Das Subjekt ist die Entität, welche auf eine Ressource zugreift. Mit dem Subjekt erhält man zusätzliche Informationen bzgl. der Identität und andere Merkmale, die z. B. die Rolle des Subjektes charakterisieren und definieren.

- **Resource:** Eine Ressource wird von einem Subjekt benutzt und verwendet. Innerhalb der Ressource können sich Informationen befinden, welche den Zugriff regeln.

- **Enviroment:** Die Umgebung kann Informationen in Form von Attributen enthalten, welche kontextbezogene Entscheidungen bzgl. der Zugriffskontrolle erlaubt. So ist es denkbar, das Daten intern für jeden frei zugänglich sind und externe keinen Zugriff auf diese bekommen.

Bei der attributbasierten Zugriffskontrolle werden alle drei Aspekte beachtet, wodurch o. g. Zugriffskontrollmodelle in Einklang gebracht werden. Dadurch sind Zugriffsregeln semantisch reicher und können ausdrucksstärker gestaltet werden. Außerdem erlaubt die Kombination dieser drei Attribute eine feingliedrige Gestaltung.

7.3.5 Entscheidung

Die Untergruppe „Security“ hat sich dafür entschieden, dass das zu implementierende Rechtemodell ein attributbasierten Ansatz widerspiegeln soll. Dieses Modell ermöglicht die Einbettung der Zugriffskontrolle am besten in den Prozess der Domänenmodellierung und der Formalisierung von Prozessen, wie sie mit Hilfe der DyWA und dem jABC später durch die Biomediziner durchgeführt werden soll.

Ein „Attribute-based Access Control“-Ansatz ermöglicht die Ausrichtung und Festlegung von Zugriffskontrolle über den Gedanken, *was* geschützt werden muss, statt Gedanken an die Frage aufzuwenden, *wem* Zugriff gestattet werden muss. Mit der Implementierung eines solchen Rechtemodells soll es den Biomedizinern, also den späteren Anwendern, ermöglicht werden, bei der Modellierung der Domäne und der anschließenden Formalisierung von Prozessen direkt den Zugriff auf neu eingeführte Daten und Datentypen einfach, intuitiv und direkt nebenbei korrekt sicherzustellen. Bei der Verwendung eines rollenbasierten Rechtemodells müssten Gedanken an die Zugriffskontrolle separat zum Prozess der Domänenmodellierung und der Prozessformalisierung erfolgen. Die separaten Arbeitsschritte zur korrekten Festlegung dieser Art der Zugriffskontrolle erscheint uns schwieriger und fehleranfälliger gegenüber dem Weg der Einbindung der Zugriffskontrolle, den wir vorhaben zu implementieren. Eine direkte Sicherstellung der korrekten Zugriffskontrolle im Prozess der Domänenmodellierung und der Formalisierung von Prozessen ist daher das Ziel unserer Sicherheitsbemühungen.

7.4 Mögliche Frameworks

Im Folgenden werden verschiedene Frameworks zur Implementierung von Anforderungen bzw. Aspekten bezüglich Sicherheit betrachtet. Die u. g. Frameworks wurden von uns zur Umsetzung der Security-Aspekte näher evaluiert. Die nachfolgenden Abschnitte enthalten dabei mehr Informationen über die einzelnen Frameworks und deren Eigenschaften sowie Vor- und Nachteile. Unsere Entscheidung zum Einsatz von Apache Shiro wird dann später in Abschnitt 7.4.5 näher erläutert.

7.4.1 Apache Shiro

Apache Shiro [24] ist ein mächtiges, einfach einzusetzendes Sicherheits-Framework, welches auf Java basiert. Die wichtigsten Funktionen des Frameworks sind

- Authentifizierung,
- Autorisierung,
- Verschlüsselung und
- Session Management.

Um Shiro effektiv einzusetzen, bietet es eine einfache API. Durch diese API lassen sich schnell kleine, aber auch sehr große Web-Applikationen schützen.

High-Level-Übersicht

Auf dem höchsten konzeptionellen Level hat die Shiro-Architektur drei primäre Konzepte. Die drei Begriffe sind: Subject, Security Manager und Realms.

- **Subject:** Das Subject ist die sicherheitsspezifische Sichtweise auf den aktuellen existierenden Benutzer. Mit dem Wort Benutzer wird sofort von einem menschlichen Benutzer ausgegangen. Dabei kann es sich bei einem Subject auch um ein 3rd-Party-Service handeln, wie z. B. Daemon Accounts, Cron Jobs oder ähnliches. Alle Instanzen von Subject sind an den Security Manager gebunden. Ohne einen Security Manager gibt es auch keine Subjects. Wird mit einem Subject interagiert, werden diese Interaktionen zu Subject-spezifischen Interaktionen mit dem SecurityManager.
- **Security Manager:** Der Security Manager ist das Herzstück von Shiro und agiert als eine Art Mantel-Objekt, dass alle internen Sicherheitskomponenten über einen Objekt-Graphen steuert. Nachdem der Security Manager konfiguriert wurde, verlagert sich der Fokus auf die Subject-API, mit deren Hilfe die Autorisierungsanfragen bearbeitet werden.
- **Realms:** Realms agieren als eine Art Brücke oder Verbindung zwischen Shiro und den eigentlichen Anwendungsdaten. Werden sicherheitsrelevante Daten benötigt, um z. B. die Authentifizierung (Login) oder die Autorisierung (Zugriffskontrolle) durchzuführen, wird auf ein oder mehrere konfigurierte Realms

zurückgegriffen. In diesem Sinne ist ein Realm ein sicherheitsbezogenes Datenzugriffsobjekt. Es fasst dabei alle wichtigen Verbindungsdaten zusammen und ermöglicht einen Zugriff auf die Daten, die Shiro benötigt. Es muss zwingend ein Realm für die Authentifizierung und Autorisierung konfiguriert werden. Allerdings ist es möglich, mehrere zu konfigurieren, wenn verschiedene Arten von Datenquellen (z. B. Datenbanken und statische Dateien) zur Verfügung stehen.

Rechteüberprüfungen finden mit Hilfe von Permissions statt. Diese Berechtigungen können individuell festgelegt werden und obliegen der Definition des Domänenexperten. Feste Vorgaben und implizite Rollenvorgaben entfallen. Die Ressourcen können explizit für die Überprüfung der Berechtigungen miteinbezogen werden. Apache Shiro unterstützt von sich aus *new-RBAC* als Rechtemodell, aber ermöglicht durch seine Offenheit die Anpassungen für *ABAC*, so dass ein dynamisches, kontextabhängiges Zugriffskontrollsystem implementiert werden kann.

7.4.2 Spring Security

Das Spring Security Framework [22] fokussiert sich vor allem auf die Unterstützung von Authentifizierung und Autorisierung und bietet darüber hinaus Schutz gegen Attacken, die für Webapplikationen in Frage kommen. Die Konfiguration erfolgt über XML-Dateien, welche schnell groß und unübersichtlich werden. Bei späteren Wartungen wäre dies ein Kritikpunkt. Außerdem scheint Spring Security auf Grund seiner flexiblen Konfigurierbarkeit kompliziert bei der Anwendung zu sein. Ein weiterer Nachteil ist die favorisierte Nutzung des rollenbasierten Zugriffsmusters, welches unserer Entscheidung aus Abschnitt 7.3.5 widerspricht. Ein Vorteil ist die Integration von Spring Security als externe Komponente in das Projekt, sodass Aktualisierungen unabhängig vom Programmcode durchgeführt werden könnten.
Spring Security unterstützt Apache Tapestry, auf dem die DyWA aufbaut, beim Anmeldeprozess nicht direkt, sondern nur über Drittanwendungen. Die Verwendung von Drittanwendungen über das Framework hinaus stellt keinen erkennbaren Vorteil dar. Außerdem müsste dann zusätzliche verwendete Lizenzen untersucht werden. Spring Security selbst steht unter der Apache Lizenz, welche uns bei der Realisierung unseres Projektes keine Probleme bereiten würde.

7.4.3 Java SE Security

Die Java Security Architektur [21] bietet eine Vielzahl von verschiedenen Services an, wie eine große Anzahl an APIs und die Implementierung häufig verwendeter Sicherheitsalgorithmen, -mechanismen und -protokollen, welches die Programmierung und Absicherung von Anwendungen ermöglicht. Die APIs decken einen großen Bereich der Sicherheit ab und ermöglichen unter anderem die Nutzung von Kryptografie, Authentifizierung, sicherer Kommunikation und Zugriffskontrolle. Die Java SE Security Architektur basiert auf folgenden drei Grundprinzipien:

- **Implementation independence:** Anwendungen müssen Sicherheit nicht eigenständig implementieren, sondern können auf Sicherheitsfunktionen der Ja-

va Plattform per Schnittstellen zugreifen. Sicherheitsfunktionen werden über Provider zur Verfügung gestellt.

- **Implementation interoperability:** Anwendungen sind nicht an Provider gebunden, welche wiederum nicht an Anwendungen gebunden sind. Dieses ermöglicht einen freien Austausch.

- **Algorithm extensibility:** Durch die Implementierung eines angepassten Providers ist es möglich, nicht implementiere Algorithmen und Funktionen für die Anwendung bereitzustellen.

Provider spezifizieren die Services, die implementiert sind und erlauben den Zugriff darauf. Bei der Nutzung von mehreren Providern ist eine Priorisierung notwendig, damit eine Anfrage an eine sicherheitsrelevante Funktion an die Provider weitergeleitet werden kann.
Gegen die Verwendung von Java SE Security spricht das fehlende Session Management, welches uns im Gegensatz dazu z. B. Apache Shiro bietet. Da unsere spätere Anwendung es den Nutzern ermöglichen muss, sich einzuloggen, ist Session Management für unsere Ziele unabdingbar. Der, im Falle des Verwendens von Java SE Security, zusätzliche anfallende Aufwand für die gesonderte Implementierung des Session Managements spricht gegen die Verwendung von Java SE Security.

7.4.4 AspectJ

Zusätzlich haben wir für unsere Anforderungen bezüglich der Security noch AspectJ [16] mit in die Überlegungen zur Auswahl eines Frameworks einbezogen. Dabei handelt es sich bei AspectJ nicht direkt um ein Framework, sondern um eine aspektorientierte Erweiterung von Java. Dadurch wäre eine Implementierung unserer Ziele auf Grundlage einer aspektorientierten Programmierung möglich. Aspektorientierte Programmierung, kurz AOP, ist ein Programmierparadigma für objektorientierte Programmierung. Mit Hilfe von AOP können querschneidene Belange (engl. Cross-Cutting Concerns), die sogenannten Aspekte, zentral formuliert werden. Aspekte können so unabhängig von der eigentlichen Anwendungslogik implementiert und zu einem späteren Zeitpunkt in den Programmcode gewoben werden. Aspekte umfassen nicht-funktionale Anforderungen an Software, wie z. B. Sicherheitsaspekte, die bei herkömmlicher Programmierung in vielen Stellen im Code einzeln realisiert werden müssten. Ein gutes Beispiel für einen solchen Aspekt ist die immer wiederkehrende Abfrage der Form „darf der aktuelle Nutzer auf diese Daten zugreifen?“.

Die folgenden Vor- und Nachteile zur aspektorientierten Programmierung sind [1] entnommen. Aspektorientierte Programmierung würde die Vorteile der Modularisierung mit sich bringen. AOP würde es ermöglichen, querschneidene Belange besser definieren und planen zu können. Zudem könnten die querschneidenen Belange konsistenter implementiert werden. Demgegenüber stehen einige Nachteile, die AOP mit sich bringen würde. Zuerst wäre ein Testen und Verifizieren des Systems mit Aspekten um einiges schwieriger als ohne den Einsatz von Aspekten. Außerdem können querschneidene Belange Auswirkungen auf die regulären Belange haben. Außerdem

ist die richtige Wahl der Aspekte nicht trivial. Die Wiederverwendbarkeit der Aspekte ist allerdings stark von der richtigen Wahl abhängig.

Auf Grund der zuvor genannten Nachteile haben wir uns gegen den konkreten Einsatz von AspectJ entschieden. Die Technologie der aspektorientierten Programmierung findet über die Nutzung der Java EE Funktionalitäten trotzdem Eingang in unsere Arbeit.

7.4.5 Entscheidung

Im Verlaufe der Evaluation der möglichen Frameworks viel unsere Wahl auf die Verwendung von Apache Shiro. Die Entscheidung für Shiro fiel auf Grund der Tatsache, dass Shiro uns eine gute Grundlage bietet, auch hinsichtlich der gewünschten Implementierung des gewünschten Rechtemodells, siehe dazu auch Abschnitt 7.3.5. Zudem ist Shiro für unsere Zwecke am flexibelsten in unser Projekt einzufügen und scheint unsere Bedürfnisse und Anforderungen am besten abzudecken bzw. zu erfüllen.

Die aspektorientierte Programmierung mit AspectJ ist auf Grund der wenigen Erfahrung der Teilnehmer und auf Grund der oben angesprochenen Nachteile für die Verwendung ausgeschieden. Gegen die Verwendung von Java SE Security sprach für uns, dass der Implementierungsaufwand im Vergleich zu Shiro deutlich höher wäre, da z. B. das Session-Management komplett von uns implementiert werden müsste, auch wenn dieses uns flexibler machen würde. Ähnliche Gründe sprachen gegen den Einsatz von Spring Security. Spring Security scheint erheblichen Aufwand für die Wartung der XML-Konfigurationsdateien mit sich zu bringen – diesen möchten wir im Vorhinein umgehen.

7.5 Kombination von Shiro und DyWA

In diesem Abschnitt wird die Verbindung bzw. Zusammenführung der DyWA mit dem von uns gewählten Sicherheits-Framework (Shiro) erklärt. Damit enthält der Abschnitt 7.5 eine Übersicht über die ersten Implementierungsarbeiten der Untergruppe „Security“.

7.5.1 DyWA-Realm

Um Shiro effektiv einzusetzen, muss ein DyWA-Realm implementiert werden. Dabei wird auf die generierten Controller-Klassen aus der DyWA zugegriffen. Vor allem muss sichergestellt sein, dass es einen standardisierten User-Typen gibt. Dieser Typ, bei uns Subject genannt, sollte mindestens die drei folgenden Variablen besitzen:

- **Name:** Damit dem Benutzer ein Name zugeordnet werden kann.
- **E-Mail:** Analog zu **Name**.
- **Passwort:** Dient zur Authentifizierung.

Bei der Authentifizierung wird auf die Daten, die in der DyWA eingetragen sind zugegriffen. Bei der Implementierung eines eigenes Realms müssen zwei Methoden überschreiben werden, nämlich:

- protected AuthenticationInfo doGetAuthenticationInfo (AuthenticationToken authenticationToken): Mit dieser Methode sollen die DyWA-Benutzer authentifiziert werden.
- protected AuthorizationInfo doGetAuthorizationInfo (PrincipalCollection principals): Mit dieser Methode werden die Zugriffsrechte der Benutzer festgelegt.

Die doGetAuthenticationInfo-Methode erhält als Parameter ein AuthenticationToken, indem alle relevanten Logininformationen enthalten sind und führt den eigentlichen Loginprozess aus. Der Loginprozess sucht sich aus der benutzen Informationsquelle, bei uns eine Datenbank, den im Token übergebenen Benutzer und überprüft die angegebenen Informationen auf Richtigkeit. Anschließend wird eine AuthenticationInfo erstellt, die alle Permissions des Users enthält.

7.5.2 Permissions

Um Shiro im Zusammenhang mit der DyWA zu nutzen, möchten wir das Konzept der Permission so verändern, dass die Permissions Beziehungen zwischen den Objekten überprüfen (ABAC Rechtemodell). Dabei werden die Permissions in zwei Kategorien unterteilt.

- CheckPermission: Die CheckPermissions überprüfen die eigentlichen Berechtigungen des eingeloggten Benutzers. Dabei werden die Objekte, die im Zusammenhang stehen, überprüft.
- RequestPermission: Die RequestPermission wird erstellt, wenn ein Recht geprüft werden soll. So ein Recht könnte z. B. das Ändern eines Datenbestandes sein.

Dabei muss allen Subjekten alle CheckPermissions zugeordnet werden. Da die Permissions in einer willkürlichen Reihenfolge ausgeführt werden und es nicht feststellbar ist, wann welche Permission benötigt wird, muss jedes Subjekt jederzeit über alle CheckPermissions verfügen.

Um Missverständnissen vorzubeugen: dies bedeutet auf keinen Fall, dass alle Benutzer alle Rechte haben werden.

Beispiel

Ein Benutzer will ein Experiment um einen Datensatz erweitern. Dabei gilt z. B. nur das Recht, dass der Ersteller des Experiments dieses auch verändern darf. Soll jetzt ein weiterer Datensatz hinzugefügt werden, wird geprüft ob der aktuelle Benutzer dem Ersteller des Experiments entspricht. Es wird also eine RequestPermission erstellt. Diese hat als Parameter das aktuell ausgewählte Experiment. Findet eine Rechteüberprüfung statt, so wird die *implies*-Methode von allen Permissions des

Subjects aufgerufen. Hier ist besonders wichtig, das vorher alle Permissions zu dem Subject hinzugefügt wurden. Trifft eine passende CheckPermission auf die dazugehörige RequestPermission, so findet eine Überprüfung der Beziehung statt, wie in Listing 7.1 zu sehen ist.

Listing 7.1: CheckPermissionExperiment

```
public class ExperimentCheckDelete implements Permission {
  @Override
  public boolean implies(Permission permission) {
      // Wrong Permission
      if(!(permission instanceof ExperimentRequestDelete)){
        return  false;
      }
      else {
        // Casting to get the Experiment Object
        ExperimentRequestDelete request = (ExperimentRequestDelete) permission;
        Subject subject = SecurityUtils.getSubject();
        Experiment experiment = request.getExperiment();
        // user, which is currently logged in
        User currentUser = (User) subject.getPrincipal();
        if(experiment.getcreator().equals(currentUser)){
            return true;
        }
      }

      return false;
  }
}
```

7.5.3 Security-Prozesse

Um Prozesse der DyWA absichern zu können, hat sich die PG dazu entschlossen, die Absicherung selbst über spezielle Prozesse zu realisieren. Diese speziellen Prozesse sind die sogenannten Security-Prozesse. Sie werden wie andere Prozesse im jABC modelliert und können durch ihre Variabilität dem Domänenexperten vielfältige Rechteüberprüfungen gestatten. Die speziellen Security-Prozesse können im jABC vielfältig im Rahmen der Möglichkeiten des jABC's modelliert werden. Die Security-Modellierung anhand von Prozessen ermöglicht es, dass Prozesse durch Prozesse selbst abgesichert werden.

Modellierung im jABC

Security-Prozesse müssen gesondert im jABC modelliert werden. Das Vorgehen hierbei ändert sich nicht im Vergleich zur Erstellung eines normalen Prozesses. Einige Besonderheiten gelten allerdings für die Modellierung von Security-Prozessen im jABC. Security-Prozesse müssen im (konfigurierbaren) Package *de.ls5.dywa.security.process* abgelegt werden. Weiterhin muss bei der Modellierung von Security-Prozessen darauf geachtet werden, dass diese im Falle der Verweigerung auf *denied* bzw. im Falle der positiv befundenen Sicherheitsüberprüfung auf *granted* enden.

Wie in Abbildung 7.1 unter Variablen zu sehen ist, kann innerhalb des jABC's auf den aktuell eingeloggten User zugegriffen werden, hier "currentUser (Subject)", indem dieser injiziert wird. Dadurch werden komplexe Regeln innerhalb eines Kontextes ermöglicht, welche im Zusammenhang mit dem Benutzer stehen können, es

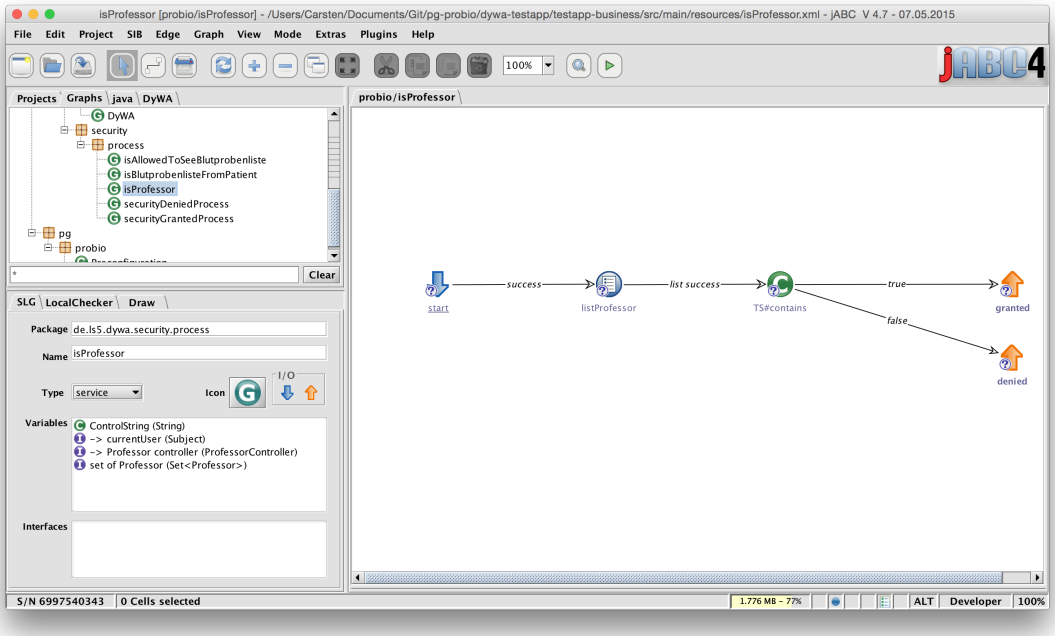

Abbildung 7.1: Beispiel eines Security Prozess im jABC

aber nicht müssen. Einfache Abfragen, wie in der Abbildung die Überprüfung, ob der gerade eingeloggte Nutzer ein Professor ist, werden ermöglicht. In diesem Beispiel wird die Liste aller Professoren vom zuständigen Controller angefordert. Anschließend erfolgt ein Überprüfung, ob der eingeloggte Nutzer in der Liste enthalten ist. Abhängig vom Ausgang der Überprüfung wird der Zugriff gewährt bzw. blockiert.

Einbindung in der DyWA

Die Security Gruppe bietet zwei unterschiedliche Formen der Einbeziehung von Security-Prozessen in die modellierte Anwendung. Die erste Form ermöglicht es, Prozesse mit Sicherheitsüberprüfungen auszustatten. Abbildung 7.2 zeigt dabei die Auswahlmöglichkeit der einzelnen Prozesse. Über den “Secure“-Button gelangt man zu der Eingabemaske, wie in Abbildung 7.3 zu sehen, welche das Hinzufügen und Entfernen von Securityprozessen erlaubt. Die verwendeten Prozesse bestimmt der Domänenexperte dabei selbst. Die zweite Möglichkeit besteht darin, analog zur Prozessabsicherung, einzelne Komponenten der GUI abzusichern. Abbildung 7.4 und 7.5 visualisieren das Vorgehen. Durch die feingliedrige Auswahl und Absicherung einzelner Komponenten ist es möglich, diese benutzerspezifisch abzusichern. Ein weiterer Vorteil ist die Verwendung der gleichen, generierten Seite, welche abhängig vom aktuellen Benutzer unterschiedliche Funktionen anbietet. So sind z. B. sensible Daten vor unberechtigten Zugriffen geschützt bzw. nur autorisierte Benutzer sind in der Lage, spezielle Funktionen zu sehen und zu benutzen. Durch die Absicherung der Komponenten ist eine abstufende Vorgehensweise möglich, da Komponenten

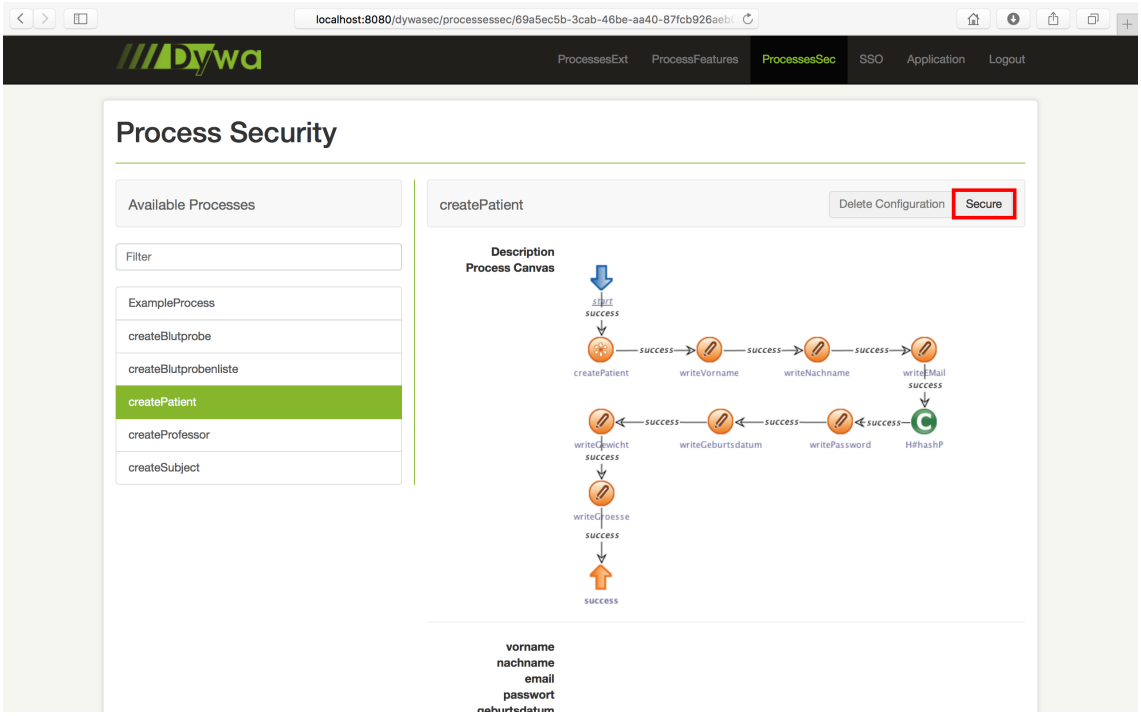

Abbildung 7.2: Absicherungsfunktion in der DyWA

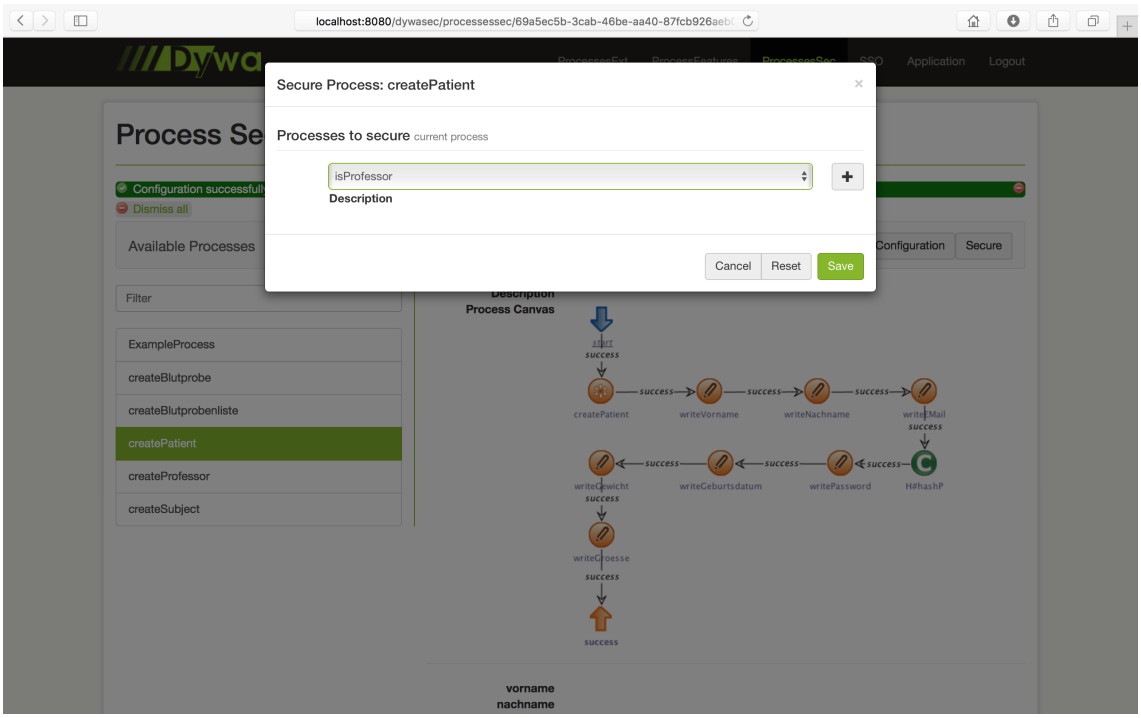

Abbildung 7.3: Auswahl der einzelnen Securityprozesse

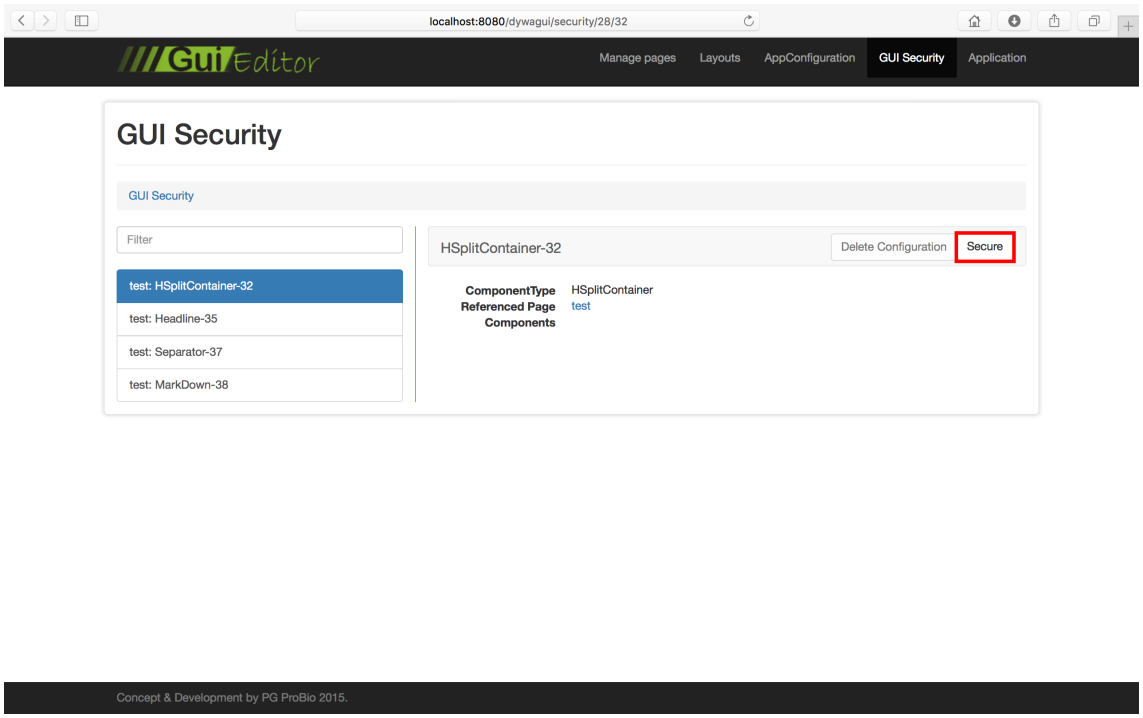

Abbildung 7.4: Auswahl der zu sichernden Komponente in der DyWA

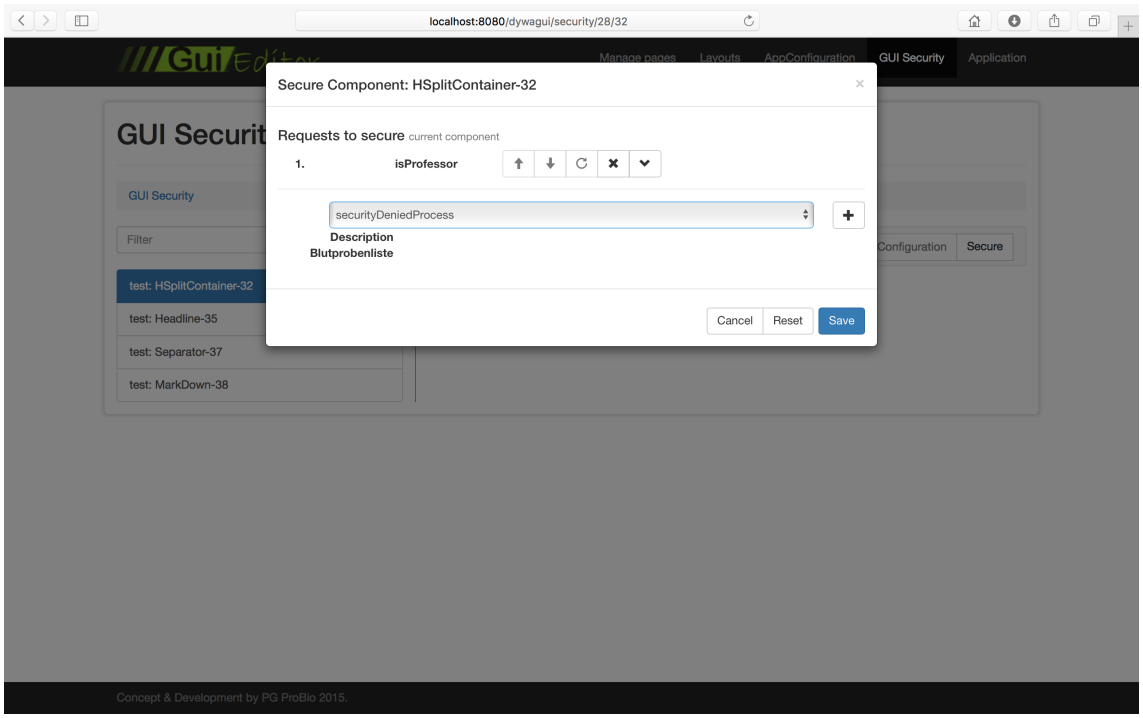

Abbildung 7.5: Auswahl der einzelnen Securityprozesse

wiederum Komponenten enthalten können. Diese können mit spezialisierten Sicherheitsprozessen geschützt werden.

7.5.4 Page Protection Filter

Die Absicherung von Webseiten in einem dynamischen Umfeld lässt sich mit Hilfe von Tapestry und des ComponentRequestFilter leicht lösen, indem jede Seite, die öffentlich zugänglich sein soll, eine Klassenannotation der Form @PublicPage erhält. Jede andere Seite ist nicht öffentlich zugänglich und führt für einen Besucher zur Login-Seite. Die Implementierung des ComponentRequestFilter von Tapestry fungiert dabei als Zwischenhändler, durch den jeder Aufruf einer Webseite geleitet wird. Die PageProtectionFilter-Klasse übernimmt diese Aufgabe und überprüft bei jeder Anfrage, ob die angefragte Seite die o. g. Annotation enthält. Wenn dieses nicht der Fall ist, so folgt die Authentifikation des Besuchers. Das Ergebnis dieser Überprüfung entscheidet, ob die angeforderte Webseite ausgeliefert wird oder der Zugriff verweigert wird. Da die Absicherung der GUI bis jetzt auf einzelne Komponenten beschränkt ist, wäre es zukünftig denkbar, dass ganze Seiten mit Hilfe von Securityprozessen abgesichert werden können. Bei dieser Vorgehensweise müssen die entsprechenden Überprüfungen in die *PageProtectionFilter*-Klasse hinein generiert werden. Eine mögliche Lösung wäre die Generierung und Verwendung von weiteren Klassenannotationen, die eine individuelle Rechteüberprüfung implizieren.

7.5.5 Verbindung

Die Einstellungen bzgl. der Securityprozesse und damit der Sicherheitsüberprüfungen werden in der Datenbank gespeichert. Dazu besteht u. a. eine Klasse ComponentSecurity, die die grundlegenden Informationen enthält und mit der abzusichernden GUI-Komponente in Beziehung steht. Vor der Codegenerierung wird ein Mapping der betroffenen Komponenten und den Absicherungen erstellt. Für jede Komponente werden die Überprüfungen anschließend mit Hilfe einer eigenen Abfrage generiert. Die Abfrage erfolgt, indem auf dem aktuellen Subject die Funktion “isPermitted“ aufgerufen wird. Diese liefert entsprechend der Modellierung einen Wahrheitswert zurück. Sollte dieser Wahrheitswert false sein, so werden die notwendigen Befugnisse nicht eingeräumt. Ein Zugriff auf die abgesicherte GUI-Komponente kann somit nicht erfolgen. Prozesse an sich werden abgesichert, indem für die Securityprozesse Check- und Requestklassen generiert werden. Jeder abgesicherte Prozess erhält bei der Generierung der Applikation einen Securityhandler mit der größtmöglichen Ausführungspriorität, so dass die Rechteüberprüfung vor der Prozessausführung erfolgt.

7.5.6 Switch User Funktion

Für einen Administrator ist es unerlässlich, Aktionen im Namen eines anderen Users auszuführen, falls dieser z. B. im Urlaub ist oder die Aktion nicht selbst durchführen kann. Um diese Funktion zu gewährleisten, bietet die Security Gruppe ein Feature an, dass es einem ermöglicht, jede beliebige Identität, welche im System vorhanden

ist bzw. alle DyWA-Typen, die von Subject erben und so berechtigt sind sich im System anzumelden, zu übernehmen. Alle nachfolgenden Aktionen werden im Namen desjenigen ausgeführt, dessen Identität übernommen worden ist. Das Ganze funktioniert natürlich ohne die Offenlegung von fremden Passwörtern, da diese geheim und privat bleiben müssen. Der Zugriff auf die notwendige Oberfläche ist dabei auf eine geringe Anzahl von Personen zu beschränken (DyWA-Administratoren), damit keine unautorisierten Aktionen durchgeführt werden können.

7.6 Zusammenfassung

Die Security Gruppe hat verschiedene Rechtemodelle evaluiert und sich für die attributbasierte Zugriffskontrolle, kurz *ABAC*, entschieden. Diese ermöglicht eine dynamische, kontextabhängige Zugriffskontrolle, bei der ressourcenbasierte Informationen, die Umgebung und das Subjekt, welches Zugriff auf die Informationen etc. haben möchte, im Vordergrund stehen.

Zur Umsetzung der *ABAC*-Zugriffskontrolle für das Apache Shiro Framework gewählt, da es eine gute Grundlage für unsere ausgehende Entwicklung bereithält und Anpassungen bzgl. unseres Rechtemodells erlaubt.

Die Modellierung von Berechtigungen erfolgt mit Hilfe von Securityprozessen im jABC. Diese weisen eine vordefinierte Syntax auf und ermöglichen eine Vielzahl von unterschiedlichen Möglichkeiten zur Regeldefinition. Dadurch werden reichhaltige, anwendungsspezifische Sicherheitsregeln ermöglicht. Der Domänenexperte hat damit die freie Wahl, welche Vorgänge und Informationen wie geschützt werden müssen.

Des Weiteren wurde die Grundlage gelegt mit Hilfe eines Filters ganze Seiten gegenüber unauthentifizierten Benutzern abzusichern und eine administrative Schnittstelle geschaffen, die es ermöglicht Benutzern bei der Ausführung von Prozessen zu helfen. Die Authentifizierung und Autorisierung von Nutzern stellt das zentrale Element dar, da diese Funktionen die Grundlage für die von uns implementierten Sicherheitsfunktionen bilden.

7.7 Ausblick

Zukünftige Entwicklungen können den in Abschnitt 7.5.4 genannten Filter dahingehend erweitert, so dass dieser die Absicherung von ganzen Seiten mit Hilfe von Securityprozessen erlaubt. Eine weitere Möglichkeit wäre die Generierung von standardisierten Securityprozessen, welche die Überprüfung von z. B. Rollen ermöglicht. Prozesse und GUI-Komponenten verbinden ausgewählte Securityprozesse per Konjunktion. Eine Erweiterung zur disjunktiven Verbindung oder einer Auswahl beider Möglichkeiten wäre denkbar. Die Absicherung der domänenspezifischen Objekte in einer generierten Anwendung erfolgt bislang lediglich über das Ein- bzw. Ausblenden von GUI-Komponenten zur Bearbeitung dieser Objekte. Zusätzlich können Änderungen an Objekten auch noch direkt über die Verbindung von Zugriffskontrollen mit den Möglichkeiten der DyWA als eine Art Bean Validation realisiert werden. Des weiteren wäre die Einbindung von Kryptographie in die Anwendung möglich.

Dadurch könnte der Domänenexperten dem Anwender Möglichkeiten zur Verfügung stellen, die es ihm ermöglichen, verschiedene Dateien bzw. Texte zu verschlüsseln.

Kapitel 8

Lizenzen

In diesem Kapitel wird eine Übersicht über die von der PG verwendeten, externen Software-Pakete und die Lizenzen dieser Software-Pakete gegeben.

8.1 Apache Software License 2.0

Die in der folgenden Aufzählung angegebene Software steht unter der Apache Software License 2.0 und wird von der PG verwendet:

- greenbird xml-formatter
  (`https://github.com/greenbird/xml-formatter-core`)
- Tapestry:
  - tapestry-core, tapestry-upload und tapestry-yuicompressor
    (`http://tapestry.apache.org/about.html`)
  - tapestry-bootstrap
    (`https://github.com/trsvax/tapestry-bootstrap`)
  - tapestry-jquery
    (`https://github.com/got5/tapestry5-jquery`)
  - tapestry-security (Tynamo)
    (`http://www.tynamo.org/tapestry-security+guide/`)
- Bootstrap Markdown
  (`http://www.codingdrama.com/bootstrap-markdown/`)
- Apache Shiro
  (`http://shiro.apache.org/license.html`)
- Apache POI
  (`https://poi.apache.org/legal.html`)
- Apache Commons Email
  (`https://commons.apache.org/proper/commons-email/index.html`)

- markdown4j
  (https://www.versioneye.com/java/org.commonjava.googlecode.markdown4j:markdown4j/2.2-cj-1.0)
- toopay bootstrap-markdown Editor
  (https://github.com/toopay/bootstrap-markdown)

8.2 MIT License

Folgende verwendete Software steht unter der MIT License:

- Bootstrap
  (http://getbootstrap.com/getting-started/#license-faqs)
- Bootstrap Submenu
  (https://github.com/vsn4ik/bootstrap-submenu/blob/master/LICENSE)
- Chart.js
  (http://www.chartjs.org/docs/#notes-license)

8.3 LGPL 2.1

Die in der nachfolgenden Aufzählung genannte Software steht unter der LGPL 2.1-Lizenz. LGPL steht dabei für die GNU Lesser General Public License.

- REngine und Rserve
  (https://github.com/s-u/REngine)
- Hibernate EntityManager
  (http://hibernate.org/community/license/)

8.4 BSD License

Die im Folgenden aufgezählte Software steht unter der BSD License:

- Postgres JDBC Driver
  (https://jdbc.postgresql.org/about/license.html)

Folgende Software steht unter einer Simplified BSD License:

- tapestry-breadcrumb
  (https://github.com/argoyle/tapestry-breadcrumbs)

Kapitel 9

Zusammenfassung

Das Ziel der PG war es, Werkzeuge zu erstellen, welche das Umsetzen einer Webapplikation ermöglicht, die auf die Bedürfnisse der späteren Anwender abgestimmt sind. Dazu wurde vor allem auf die Erweiterung der DyWA und des jABC geachtet, da diese Tools den Ausgangspunkt für unser Projekt gelegt haben. Die Untergruppe zum Thema GUI-Modellierung entwickelte eine Anwendung mit deren Hilfe die Gestaltung einer Webanwendungsoberfläche ohne Programmierkenntnisse möglich ist. Sie bieten zahlreiche Tools zur Modellierung an, darunter befinden sich neben einfachen Markdown-Komponenten zur Textverarbeitung, Komponenten zur Einbindung von Bildern auch aufwändigere Komponenten, die es einem ermöglichen, Diagramme zu generieren, laufende Prozesse zu sehen und fortsetzen zu können, sowie Eingabeformulare, um Daten in der Anwendung abzulegen. Das Search Plugin ermöglicht dem Anwender die Erstellung von Suchanfragen, die später in die grafische Oberfläche der Webanwendung eingebunden werden können. Dort zeigen sie in tabellarischer Form die Ergebnisse der Suche an. Sicherheit wird durch die Modellierung von Securityprozessen gewährleistet, welche von der dafür zuständigen Untergruppe eingeführt wurden. Die Authentifizierung und Autorisierung sind die Hauptmerkmale, um die sich während der einjährigen Projektphase von der Security Gruppe gekümmert wurden. Prozesse, einzelne GUI-Komponenten sowie ganze Seiten können nun mit extra modellierten Sicherheitsprozessen abgesichert werden. Mit den Erweiterungen der Logic Gruppe können vor allem erweiterte Semantiken auf SIBs in jABC-Prozessen angewendet werden, die die Verwendung von Timern, die Unterbrechung der Prozesse für Benutzerinteraktionen oder den Nachrichtenaustausch zwischen Prozessen innerhalb von Webanwendungen ermöglichen. Dazu können unterbrochene Prozessinstanzen über ein Plugin in einer Weboberfläche verwaltet werden.

Insgesamt ist es der PG gelungen, die o. g. Tools DyWA und jABC um viele Funktionen zu erweitern. Außerdem gelingt das Zusammenspiel der einzelnen Untergruppen und der von ihnen angebotenen Funktionen, so dass die Modellierung und Generierung einer domänenspezifischen Anwendung möglich ist. Dadurch wurde das aus DyWA und jABC bestehende Framework so erweitert, dass ein System entstanden ist, mit dem andere Systeme durch Domänenexperten gebaut werden können.

Kapitel 10

Ausblick

Die PG hat in ihrer Zeit zahlreiche neue Features implementiert und die vorhandenen Werkzeuge DyWA und jABC erweitert. Trotz allen Anstrengungen haben einige Funktionalitäten nicht den Weg in die jeweiligen Erweiterungen gefunden. Darunter fallen bei der GUI Gruppe die Erweiterung der Suchfunktionalität mit Suchparametern, die zusätzliche Verwendung von Prozessergebnissen in vorhandenen Komponenten, das Anbieten von mehreren Sprachen zwecks Internationalisierung und die Wiederverwendung von Komponenten. Bei der Search Gruppe steht vor allem die Implementierung einer Volltextsuche auf der Agenda und die Erweiterung der Suchoptionen. Eine Erweiterung der Sicherheitsfunktionen um Kryptographiemöglichkeiten wäre für die Security Gruppe ein anzustrebendes Ziel. Des weiteren wäre die disjunktive Verbindung von Securityprozessen während der Modellierung und eine sich verändernde Ausgabe der GUI, abhängig von den Rechten des Benutzers, wichtige Ziele, die die Funktionalität der gestalteten Anwendung weiter erhöhen würde. Bei den Prozessen sind insbesondere Erweiterungen an der Verbindung mit der GUI-Modellierung für eine benutzerfreundliche Nutzung von Wizards sowie die Bereitstellung einer Batchausführung für Prozesse vorzunehmen.

Abbildungsverzeichnis

Programmcode

Literatur

[1] Josef Adersberger. *Aspektorientierte Softwareentwicklung: Eine Einführung mit Schwerpunkt Aspektorientierte Programmierung*. http://www11.informatik.uni-erlangen.de/Lehre/WS0708/HS-AOSD/materialien/vortraege/adersberger-aop-tutorial.pdf. [Online; Zugriff 30. März 2015].

[2] Mike Bostock. *Overview*. URL: http://d3js.org (besucht am 30. 09. 2015).

[3] Chart.js. *Dokumentation*. URL: http://www.chartjs.org/docs/ (besucht am 30. 09. 2015).

[4] Henri Chen. *ZK - Ajax without JavaScript*. Apress, 2008, S. 125. ISBN: 978-1-59059-901-3.

[5] DF Ferrailio, DR Kuhn und R Chandramouli. „Role based access control". In: *15th National Computer Security Conference*. 1992.

[6] Markus Frohme. „Agile Domänenmodellierung für prozessgesteuerte Webanwendungen (Bachelorarbeit)". TU Dortmund, 2013.

[7] Michael Y. Galperin und Guy R. Cochrane. „The 2011 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection". In: *Nucleic acids research* 39.Database issue (2011), S. D1–6. ISSN: 1362-4962. DOI: 10.1093/nar/gkq1243.

[8] GenBank Statistics. *GenBank Statistics (ca. 2008)*. URL: http://www.ncbi.nlm.nih.gov/genbank/genbankstats-2008/ (besucht am 06. 04. 2015).

[9] K. Zeppenfeld H. El. Moussaoui. *Informatik im Fokus: AJAX - Geschichte, Technologie, Zukunft*. 1. Aufl. Berlin, Heidelberg: Springer, 2008. ISBN: 978-3-540-73112-2.

[10] Hibernate.org. *Criteria API - Konzepte*. URL: http://docs.oracle.com/middleware/1213/toplink/concepts/queries.htm#OTLCG94396 (besucht am 15. 10. 2015).

[11] Hibernate.org. *Dokumentation*. URL: http://hibernate.org/search/documentation/getting-started/ (besucht am 07. 04. 2015).

[12] Torstein Honsi und Grethe Hjetland. *Dokumentation*. URL: http://www.highcharts.com/docs (besucht am 30. 09. 2015).

[13] Bundesamt für Sicherheit in der Informationstechnik. *IT-Sicherheitskriterien und Evaluierung nach ITSEC*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/ITSicherheitskriterien/itsec-dt_pdf.pdf;jsessionid=69C5C21E38F3F9428C4A831CE10316FD.2_cid286?__blob=publicationFile. [Online; Zugriff 30. März 2015].

[14] Yixin Jiang u. a. „Security analysis of mandatory access control model“. In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Bd. 6. 2004, 5013–5018 vol.6. DOI: 10.1109/ICSMC.2004.1400987.

[15] Sven Jörges. *Construction and Evolution of Code Generators - A Model-Driven and Service-Oriented Approach*. Bd. 7747. Lecture Notes in Computer Science. Springer, 2013.

[16] Gregor Kiczales u. a. „An overview of AspectJ“. In: *ECOOP 2001 - Object-Oriented Programming*. Springer, 2001, S. 327–354.

[17] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. 1. Aufl. Addison-Wesley Professional, 2008. ISBN: 0321553454, 9780321553454.

[18] Johannes Neubauer. *Higher-Order Process Engineering: The Technical Background*. https://eldorado.tu-dortmund.de/bitstream/2003/33102/1/hope-tr.pdf. Apr. 2014.

[19] Johannes Neubauer. „Higher-Order Process Engineering“. Diss. TU Dortmund, 2014.

[20] R. S. Sandhu und P. Samarati. „Access control: principle and practice“. In: *IEEE Communications Magazine* 32.9 (1994), S. 40–48. ISSN: 0163-6804. DOI: 10.1109/35.312842.

[21] Java SE Security. *Dokumentation*. URL: http://docs.oracle.com/javase/8/docs/technotes/guides/security/index.html (besucht am 07. 04. 2015).

[22] Spring Security. *Dokumentation*. URL: http://docs.spring.io/spring-security/site/docs/4.0.0.RELEASE/reference/htmlsingle/ (besucht am 07. 04. 2015).

[23] Howard Lewis Ship. *Apache Tapestry Documentation*. 2016. (Besucht am 26. 03. 2016).

[24] Apache Shiro. *Dokumentation*. URL: http://shiro.apache.org/documentation.html (besucht am 07. 04. 2015).

[25] Bernhard Steffen, Henning Bordihn und Oliver Rüthing. *Introduction to Code Generation for Model-Driven Software Development (MDSD)*. Vorlesungsfolien Virtualisierung und Compilation, Wintersemester 2013/14, TU Dortmund. 2013.

[26] Bernhard Steffen u. a. „Model-driven Development with the jABC“. In: *Proceedings of the 2Nd International Haifa Verification Conference on Hardware and Software, Verification and Testing*. HVC'06. Berlin, Heidelberg: Springer-Verlag, 2007, S. 92–108. ISBN: 978-3-540-70888-9.

[27] Manfred Steyer Vildan Softic. *AngularJS: Moderne Webanwendungen und Single Page Applications mit JavaScript*. Beijing; Köln [u.a.]: O'Reilly, 2015.

[28] Eric Yuan und Jin Tong. „Attributed based access control (ABAC) for web services". In: *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE. 2005.