ulm university universität ulm

Prototyp eines JavaFX Clients für ein bestehendes Datenmodell

Philipp Streicher

Universität Ulm

Fakultät für Ingenieurwissenschaften und Informatik

Institut für Programmiermethodik und Compilerbau

Mai 2014

Bachelorarbeit im Studiengang Software-Engineering

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbstständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Quellen sind vollständig zitiert.

Datum: _____ **Unterschrift**_____

ABSTRACT

Prototyp eines JavaFX Clients für ein bestehendes Datenmodell

Von Philipp Streicher

„Software, die nicht ständig an die sich ändernde Umgebung angepasst wird, altert und ist irgendwann veraltet, d.h. sie kann nicht mehr für den ursprünglich vorgesehenen Zweck eingesetzt werden.“ (Balzert und Balzert 2009, Seite 9) Aus diesem Grund sollten Software-Projekte danach streben, mit aktuellen Frameworks zu arbeiten.

Aber welche Vorteile bringt ein neues Framework für mein Projekt? Welcher Aufwand ist mit einer Umstellung verbunden? Stehen Nutzen und Aufwand in einem gesunden Verhältnis? Diese Arbeit beschäftigt sich genau mit diesen Fragen in Bezug auf das Swing UI Framework und dessen Alternativen, unter den Umständen des Websphere Application Server und Java zu benutzen.

Das einzige Framework, das einen Mehrwert für das Projekt ePEP bringen kann, ist JavaFX. Bei der Evaluierung wird deutlich, dass JavaFX ein Framework mit Potential ist, welches jedoch noch nicht vollständig ausgeschöpft wurde. Es unterstützt das Data Binding, eine deskriptive UI und es können Webapplikationen entwickelt werden. Jedoch ist ein Framework für die Form Validierung nicht in JavaFX enthalten und das Framework hat sich in der Community bisher noch nicht etabliert.

Da die Vorteile von JavaFX überwiegen, wird hiermit ein Prototyp implementiert, um die von JavaFX benötigten Schnittstellen von Kern der Applikation zu ermitteln. Darauf aufbauend werden die Änderungen identifiziert, die einen Umstieg auf JavaFX im Projekt ePEP ermöglichen.

Es wurde deutlich, dass JavaFX die einzige Alternative zu Swing ist, die einen Mehrwert für Projekte bringt, wenn gegeben ist, dass der Websphere Application Server und Java genutzt

werden soll. Das Ergebnis stellt außerdem heraus, dass die Umstellung mit wenig Aufwand erfolgen kann, unter den Prämissen, dass bereits eine Trennung zwischen Logik und UI existiert. In ePEP existiert diese strikte Trennung in einem Viertel der Komponenten, jedoch nicht in allen. Hier muss erst ein Refactoring durchgeführt werden, damit Logik und UI voneinander getrennt werden. Weiter ist es möglich, die Swing Applikation teilweise zu migrieren, da JavaFX auch in Swing Applikationen verwendet werden kann. Dies minimiert Risiken und Kosten.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

AWT ..... *Abstract Window Toolkit*
CSS3 ..... *Cascading Style Sheets*
EDT ..... *Event Dispatch Thread*
ePEP ..... *elektronischer Produktions Einsatz Prozess*
HTML5 ..... *Hypertext Markup Language*
J2EE ..... *Java 2 Platform, Enterprise Edition*
MVVM ..... *Model-View-ViewModel*
RIAs ..... *Rich Internet Applications*
SLOC ..... *Source Lines of Code*
SWT ..... *The Standard Widget Toolkit*
WAS ..... *Websphere Application Server*
XML ..... *Extensible Markup Language*

1 EINLEITUNG

„If it works, don't fix it" – Diese Aussage findet sich in dem Buch „Refactoring: Improving the Design of Existing Code" von Martin Fowler (Fowler 1999) bereits in der Einleitung wieder. Fowler führt in die Thematik des Refactorings ein, indem er von einem scheiternden Projekt berichtet, bei welchem die Manager Anhänger dieser Aussage waren.

Warum sollte ein Framework wie Swing, welches seit 1997 in Betrieb ist und augenscheinlich gut funktioniert, ausgetauscht werden? Über die Jahre wurde Swing weiterentwickelt, getestet, dokumentiert und hat sich in der Community etabliert. Mittlerweile existieren zehntausende Codezeilen, die jedoch ersetzt werden sollen.

Oracle hat einen offiziellen Nachfolger für Swing bekannt gegeben. Das bedeutet, dass Swing nicht mehr weiterentwickelt wird und zukünftige Anforderungen mit Swing nicht mehr effizient umgesetzt werden können. Fehler, die in Swing existieren, werden möglicherweise nicht mehr behoben und keine neuen Features implementiert. Softwaresysteme, die Swing nutzen und nicht veralten sollen, werden deswegen früher oder später gezwungen sein, das UI Framework zu wechseln.

1.1 AUFGABENSTELLUNG UND ZIELSETZUNG

Ziel dieser Arbeit ist es, eine Handlungsempfehlung für die Daimler TSS GmbH zu erstellen. Es soll eine Alternative zum Swing UI Framework erarbeitet und evaluiert werden. Die Alternative soll in der Applikation ePEP (elektronischer Produktions Einsatz Prozess), die bisher das Swing UI Framework verwendet, eingesetzt werden. Eine Aufwandsabschätzung für den möglichen Austausch des UI Frameworks soll erstellt werden. In diesem Zusammenhang müssen die Schnittstellen, die es vom Kern der Applikation zu Swing gibt, ermittelt werden. Darauf aufbauend sollen die Änderungen an den Schnittstellen identifiziert werden, damit sie mit einem anderen UI Framework benutzt werden können. Ein weiteres Ziel der Arbeit ist es generelle Vorgehensweisen bzw. Hinweise zu ermitteln, die helfen Swing auf ein alternatives Framework umzustellen.

1.2 GLIEDERUNG DER ARBEIT

Der Hauptteil unterteilt sich in fünf Abschnitte. Zu Beginn wird der Leser in allgemeine Grundlagen eingeführt und mithilfe einer IST-Analyse wird der Status Quo der Anwendung ePEP ermittelt. Dieser dient als Vergleichs- und Bewertungsbasis für den SOLL-Zustand. Des Weiteren werden die Schwächen von Swing bei der IST-Analyse ermittelt. Anhand derer und der Features von alternativen Frameworks werden die Schwächen bewertet. Ziel ist es, die beste Alternative mit dem meisten Mehrwert zu wählen, um diesen in allen für ePEP relevanten Punkten zu evaluieren. Mithilfe der Ergebnisse wird ein Prototyp mit dem gewählten, alternativen Framework entwickelt, um eine Aufwandsabschätzung für eine mögliche Umstellung der ePEP UI durchführen zu können. Anhand des Prototyps wird ermittelt, wie die Schnittstellen der Logik in ePEP geändert werden müssten, um von einem neuen UI Framework genutzt werden zu können.

2 GRUNDLAGEN

Bei einem Software- und Systementwurf werden mithilfe des Pflichtenheftes eine Gesamtstruktur und eine Detailstruktur des Systems erstellt. Diese bilden eine Systemarchitektur. (Rumpe 2011, Folie 4+5) Um die Systemarchitektur festlegen zu können, müssen sich die Stakeholder[1] entscheiden, welcher Typ Client in der Anwendung verwendet werden soll. Bei ePEP wurde dies schon vor mehr als zehn Jahren bestimmt. ePEP baut auf dem Rich-Client Prinzip auf und hat mittlerweile eine Größe von ca. 860.000 SLOC[2] erlangt. Wenn bei einer Anwendung einer solchen Größe das UI Framework ausgetauscht werden soll, ist dies mit Aufwand und Kosten verbunden. Aus diesem Grund muss genau untersucht werden, welche Vor- und Nachteile sich ergeben, wenn das UI Framework ersetzt wird. Hierbei müssen Kosten und Nutzen gegenüber gestellt werden.

2.1 DEFINITIONEN

Im Folgenden werden Begriffe und Konzepte erklärt, die für die weitere Arbeit von Bedeutung sind.

2.1.1 FRAMEWORK

Ein Framework ist eine unterstützende Struktur, die dem Entwickler durch Rahmen und Einschränkungen eine bestimmte Architektur vorgibt. (ITWissen 2014, Stichwort: Framework) *Frameworks unterstützen die Entwicklung, da Komponenten besser wiederverwendet werden können.* (Riehle 2000, Seite 1) Vorgefertigte Konstrukte und Funktionen können benutzt werden, welche in Folge dessen nicht mehr selbst programmiert werden müssen. *Frameworks tendieren dazu weniger Fehler zu enthalten, da Komponenten wiederverwendet werden können.* (Riehle 2000, Seite 9)

[1] Stakeholder: *„Person, Rolle oder Organisation, die potenzielles Interesse an zukünftigem System hat...“* (Partsch 2014)

[2] Source Lines of Code (SLOC) bezeichnet die Anzahl an Codezeilen in einem Programm ohne Leerzeilen und Kommentare.

Sobald ein Framework von mehreren Benutzern eingesetzt wird, sind bestimmte Komponenten gleich. Das ermöglicht auch projektfremden Kollegen den Code einfacher zu verstehen und darüber zu diskutieren.

Der Einsatz von Frameworks muss immer mit den Stakeholdern abgestimmt werden, da die Frameworks nicht vollständig unter Kontrolle der Entwickler sind. Das bedeutet, es können Sicherheitsrisiken oder Abhängigkeiten zu den Drittanbietern entstehen.

2.1.2 Drittanbieter Bibliotheken

Drittanbieter Bibliotheken oder auch Third Party Libraries werden von Personen oder Firmen bereitgestellt. Dabei bieten sie dem Entwickler zusätzliche Komponenten, welche die Entwicklung vereinfachen sollen, jedoch nicht von dem Herausgeber der Programmiersprache bereitgestellt werden.

Je nach Projekt müssen, wie bei Frameworks auch, die Stakeholder über Sicherheitsrisiken informiert werden. Nachteilig ist zudem, dass bei dem Release einer neuen Version die Third Party Library u. U. längere Zeit benötigt, um die neuesten Features der Programmiersprache zu unterstützen. Dadurch kann eine Inkompatibilität entstehen und die neuesten Features stehen nicht zur Verfügung.

Wenn Third Party Bibliotheken genutzt werden, entsteht automatisch eine Abhängigkeit zu den Bibliotheken und damit auch zu deren Anbieter. Sollte der Support für eine Bibliothek wegfallen, müssen sich die Nutzer überlegen, ob die Bibliothek mit ihren bestehenden Fehlern und ohne neue Features weiterhin benutzt werden kann. Die Alternative ist, eine neue Third Party Bibliothek zu benutzen oder eine eigene Lösung zu programmieren. Dies kann jedoch einen größeren Aufwand mit sich bringen. Aus diesen Gründen sollten sich die Entwickler zusammen mit den Stakeholdern immer überlegen, ob eine Third Party Bibliothek in das Projekt eingebunden werden sollte.

2.1.3 JavaBeans

JavaBeans sind Java Klassen, die im herkömmlichen Sinn keine Logik implementieren dürfen. Mittlerweile gibt es jedoch keine einheitliche Definition mehr. Grundsätzlich sind JavaBeans Komponenten, welche die *„Übersichtlichkeit und Wiederverwendbarkeit“* (Kemper und

Eickler 2011, Seite 579) erhöhen sollen. JavaBeans sind in Java dafür zuständig die Eigenschaften von einem Objekt zu repräsentieren. JavaBeans können genutzt werden, um Objekte zu persistieren bzw. Datenmodelle aufzubauen. Dabei unterliegen sie Konventionen, die vorschreiben, dass die JavaBean-Klassen ihre Eigenschaften, die dazugehörigen getter und setter[3] und einen parameterlosen Konstruktor implementieren müssen.

Außerdem können sie auch Ereignisse auslösen, wenn Daten geändert werden. Das ist nützlich, sofern die JavaBeans eine View repräsentieren. Durch die Ereignisse werden die Views benachrichtigt, dass ihre Daten geändert wurden.

2.1.4 REFACTORING

Ein Refactoring beschreibt eine Technik, um existierenden Code aufzuräumen und zu säubern. Dabei wird der bestehende Code umgeschrieben, mit dem Ziel die Software verständlicher und änderbar zu implementieren, jedoch das Verhalten dabei möglichst nah am Original zu behalten. (Fowler 1999, Einleitung xvi) Durch die Technik des Refactorings soll verhindert werden, dass, bei Änderungen am Sourcecode, Fehler eingebaut werden oder die Logik verändert wird.

2.1.5 CLIENT-SERVER ARCHITEKTUR

„Die Initiative zu einer Interaktion geht vom Client aus. Er formuliert Aufträge und schickt sie an einen Dienstanbieter (Server), der eine Dienstbereitschaft für eine bestimmte Art von Dienst (Service) veröffentlicht hat und entsprechende Aufträge entgegennimmt, diese bearbeitet und die Antwort zum Client zurück schickt.“ (Geihs 1995, Seite 7) Der Vorteil von Client-Server Systemen liegt unter anderem in der hohen Anpassungsfähigkeit, wenn sich das System bspw. durch steigende Benutzeranzahl verändert. Die Architektur wird genauer erläutert, wenn die verschiedenen Clients in den nächsten Kapiteln beschrieben werden.

[3] Getter und setter: Öffentliche Methoden die den Zugriff auf eine private Klassenvariable ermöglichen.

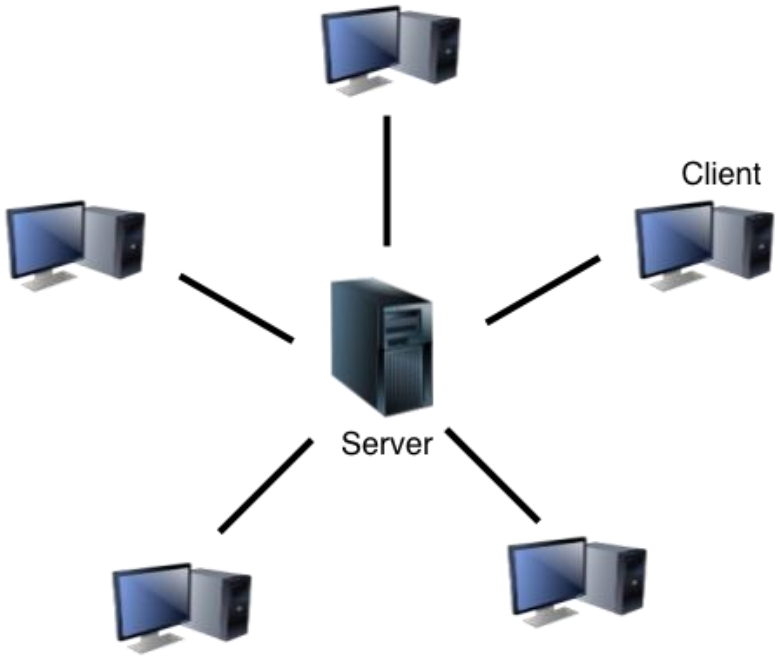

Abbildung 1: Client-Server Architektur

In der Abbildung sind fünf Clients dargestellt, die auf denselben Server zugreifen. Der Server stellt für alle Clients die gleichen Dienste zur Verfügung, wobei dies auch über Zugriffsrechte gesteuert werden kann. Dabei besteht keine direkte Verbindung der Clients untereinander. Wenn der Server einen entsprechenden Dienst anbietet, können die Clients über den Server kommunizieren.

2.2 CLIENT-TYPEN

Es gibt verschiedene Typen von Clients, die in diesem Kapitel erklärt und verglichen werden.

2.2.1 FAT-/RICH-CLIENT

Bei der Client-Server Architektur wird zwischen zwei verschiedenen Typen von Clients unterschieden: Dem Fat-/Rich-Client und dem Thin-Client. Bei Fat-Clients werden die Funktionalität und die Benutzerschnittstelle der Anwendung auf den Clients implementiert. Typische Charakteristika sind, dass *native Widgets unterstützt werden, Drag- and Drop-Funktionen möglich sind und eine Plattformunabhängigkeit gegeben ist*. (ITWissen 2014, Stichwort: Rich-Client) Außerdem ist es möglich lokale high Performance Berechnungen

durchzuführen, die dabei nicht von anderen Faktoren, wie Serverauslastung oder Netzwerkbandbreite, abhängig sind. Der Rich-Client baut auf dem Prinzip des Fat-Clients auf und *enthält eine integrierte Update-Funktionalität. Außerdem sind sie häufig Front Ends*[4] *für eine Server-Applikation und bestehen aus einem Komponentenmodell*[5]*.* (ITWissen 2014, Stichwort: Rich-Client) Beispiele für Fat-/Rich-Client Technologien sind Java mit Swing oder Windows Forms Anwendungen.

2.2.2 Thin-Client sowie Gegenüberstellung zu Fat-/Rich-Client

Der Thin-Client implementiert im Gegensatz zum Fat-/Rich-Client keinerlei Funktionalität. Alle Funktionen werden von einem Server bereitgestellt und die *Benutzerschnittstelle wird bspw. über einen Webbrowser oder Remote Desktop auf dem Client dargestellt.* (Durville, et al. 2010)

Ein Vorteil von Thin-Client Applikationen ist es, dass die Hardware für die Clients minimiert werden kann. Dadurch können Kosten gespart werden, da nur noch ein Terminal-PC bereitgestellt werden muss und kein vollständig ausgerüsteter Computer. Eine dreijährige Studie des Fraunhofer Instituts hat die Kosten untersucht, die während eines Lebens der Hardware entstehen. Dabei wurden 130 Clients betrachtet. Berücksichtigt wurden Anschaffungspreis, Software-Lizenzen, Personalaufwände. Es wurden die drei Phasen der Beschaffung, des Betriebs und der Außerbetriebnahme analysiert. Die Studie zeigt, dass Thin-Clients, *im Schnitt auf ca. 1400 € und Fat-/Rich-Clients im Schnitt auf ca. 2200 € kommen.* (UMSICHT 2011) Dieser Kostenunterschied kommt zustande, weil Thin-Clients weniger Hardware im Client benötigen, eine zentrale Wartung günstiger ist und die Rechenkapazität von den Servern optimal ausgenutzt werden kann.

Die Thin-Clients können sich dann bspw. über Remote Desktop auf den Server, der die Services bereitstellt, einloggen und die Ressourcen des Servers nutzen, um eine Applikation auszuführen.

---

[4] *Ein Gerät oder Programm, welche dem User direkten Zugriff auf weitere Gräte und Programme ermöglicht.* (Oxford University Press 2014)

[5] *Komponentenmodelle bieten eine konzeptionelle und technische Basis zur komponentenbasierten Entwicklung verteilter Anwendungen und legen Schnittstellenverträge zwischen Komponenten und der Umgebung fest.* (Hammerschall 2005)

Software Management Probleme wie Wartung und Updates werden durch Thin-Clients gelöst, da dies lediglich zentral auf dem Server durchgeführt werden muss. (McAffer, Lemieux und Aniszczyk 2010, Kapitel 1) Bei den Fat-/Rich-Clients muss ein Update und die Wartung lokal auf den Client PCs durchgeführt werden.

Ein Nachteil ist, dass es Einschränkungen bei der Visualisierung und Interaktion gibt, wenn die Anwendung über den Browser dargestellt wird. (ITWissen 2014, Stichwort: Rich-Client). Zum Beispiel ist Drag-and-Drop bei herkömmlichen Thin-Clients nicht möglich.

Ein Argument für die Fat-/Rich-Clients ist, dass diese auch offline arbeiten können. Das ist wichtig, da der Server u. U. nicht immer zur Verfügung steht und die Sicherheit bei der Kommunikation über das Netzwerk nicht sichergestellt werden kann (ITWissen 2014, Stichwort: Rich-Client)

Trotz der Vorteile der Thin-Clients für den durchschnittlichen Benutzer sind in der Praxis die Fat-/Rich-Clients häufiger vertreten. Dies ist laut der Studie des Fraunhofer Instituts vor drei Jahren darauf zurückzuführen, dass die Bekanntheit und Akzeptanz von Thin-Clients auf einem niedrigen Level lagen und dies erst noch gesteigert werden müsste. Heutzutage liegt das Potential jedoch in den Rich Internet Applications.

2.2.3 Rich Internet Application

Moderne Applikationen, die über einen Browser laufen und mit HTML5, CSS3 und JavaScript implementiert werden, gehören zu den Rich Internet Applications (RIAs). Diese Applikationen ordnen sich zwischen Rich- und Thin-Clients an. Die Applikationen werden auf dem Server gestartet und der Programmierer kann entscheiden, ob der weitere Code auf dem Client oder dem Server ausgeführt wird.

Durch den Download von Präsentations- und Businesslogik auf den Computer des Nutzers, kann bspw. auf Eingaben unmittelbar reagiert werden. (Durville, et al. 2010). Dabei können RIAs auch offline bereit stehen, lokal Daten speichern und auch mit den lokalen Ressourcen arbeiten. RIAs vereinen die Vorteile von Thin- und Rich-Clients:

- Die Wartung und das Einspielen von Updates können zentral am Server durchgeführt werden

- Es können Ressourcen von Server und Client genutzt werden
- Es gibt die Möglichkeit alle Interaktionsformen zu benutzen

Ein Nachteil von RIAs war, dass sie nur über ein Plug-In im Browser installiert werden konnten, welches eine Interaktion des Benutzers erfordert und damit potentielle Fehlerquellen birgt. Jedoch ist dieser Nachteil nicht mehr vorhanden, wenn Webframeworks wie HTML5, CSS3 und JavaScript verwendet werden. Der Grund dafür ist, dass JavaScript automatisch im Hintergrund auf den Client heruntergeladen wird.

Die Client-Logik wird in JavaScript-Files implementiert und diese werden automatisch auf den Client geladen. Allerdings kann bei großen Anwendungen durch den Download der JavaScript Files eine längere Ladezeit entstehen. Beispiele eines RIA Frameworks sind GWT, ASP.NET oder JavaFX, wobei mit JavaFX sowohl RIAs als auch Rich-Clients erstellt werden können.

In Tabelle 1 werden die Client-Typen noch einmal gegenüber gestellt.

	Fat-/Rich-Client	Thin-Client	RIA
Interaktionsmöglichkeiten	✓	✗	✓
Plattformunabhängigkeit	✓	✗	✗
Unabhängig von anderen System	✓	✗	✓✗
Kosten	✗	✓	✓✗
Software Management	✗	✓	✓
Nutzung lokaler Ressourcen	✓	✗	✓

Tabelle 1: Übersichtstabelle der verschiedenen Client-Typen

Die Tabelle zeigt noch einmal, dass die Rich Internet Applications die Vorteile von Fat-/Rich-Client und Thin-Client vereinigen. Eine RIA muss nicht immer plattformunabhängig sein.

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

Wenn die Darstellung über einen Browser realisiert wird, dann ist eine starke Abhängigkeit zum Browser gegeben. Die Interaktionsmöglichkeiten und die Nutzung lokaler Ressourcen sind vollständig gegebenen. Außerdem ist das Software Management wie bei Thin-Clients wesentlich vereinfacht.

3 STATUS QUO

Die Applikation ePEP ist eine Eigenentwicklung der Daimler TSS GmbH und nach Anforderungen der IT- und den Fachabteilungen bei Daimler Trucks implementiert. Mit ePEP werden Produktionen geplant und dokumentiert. Technisch werden Daten vom Benutzer manipuliert und aus etwa 70 Schnittstellen Informationen verarbeitet bzw. dorthin exportiert. Um den Sicherheitsstandards des Daimler Konzerns zu entsprechen, war und ist vorgegeben, dass der Websphere Application Server (WAS)[6] und Java genutzt werden müssen. Außerdem hat sich das Projektteam zum damaligen Zeitpunkt entschieden, die Client-/Server-Architektur mit J2EE[7] im Server und Swing im Client einzusetzen.

3.1 IST-ANALYSE DER UI ARCHITEKTUR

Für Swing wurde ein offizieller Nachfolger bekannt gegeben. Dies bedeutet, dass Swing nicht weiterentwickelt wird. Dadurch werden Projekte, die auf dem aktuellen Stand der Technik bleiben möchten, gezwungen auf ein anderes UI Framework umzusteigen. Zusätzlich haben sich die Anforderungen an Systeme und Frameworks verändert. Aus heutiger Sicht, bietet Swing im Vergleich zu anderen Frameworks nicht mehr die Unterstützung für die Programmierer, die andere, modernere Frameworks bieten.

3.1.1 UI AUFBAU

Der Aufbau der UI unterliegt einem festen Schema (Abbildung 2). Ziel der UI ist es eine übersichtliche Darstellung großer Datenmengen zu gewährleisten, welche dann auch bearbeitet werden können.

[6] Der WAS ist eine von IBM entwickelte Serverlaufzeitumgebung für Java Anwendungen.

[7] J2EE (Java 2 Platform, Enterprise Edition) ist *„ein modulares, komponentenbasiertes Modell, dessen Bestandteile über standardisierte Schnittstellen miteinander kommunizieren.“* (Stark 2005)

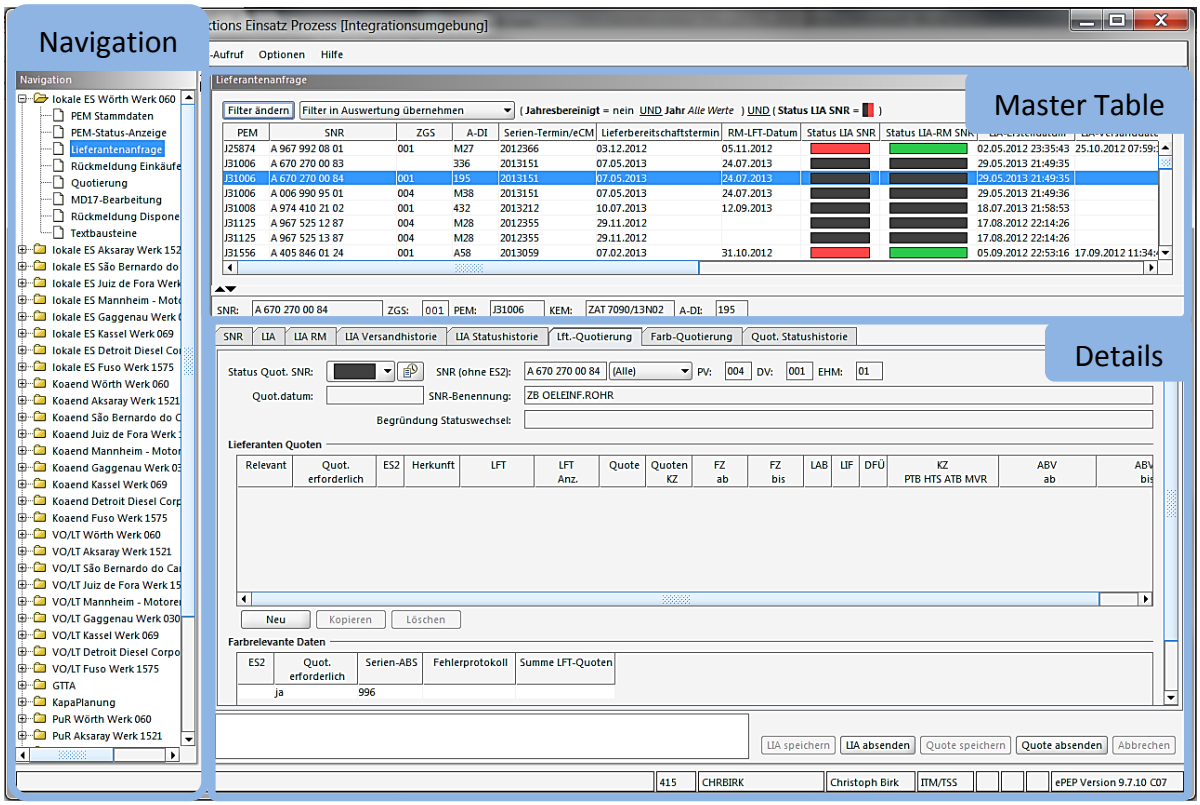

Abbildung 2: ePEP UI

Die ePEP UI ist in drei verschiedene Bereiche unterteilt: Navigation, Master Table und Details. Zusätzlich gibt es noch eine Menüleiste oben und eine User Status Leiste unten in der Anwendung. Auf der linken Seite ist die Navigation. Die Benutzer müssen zu Beginn ein Modul auswählen. Als nächstes stehen unter den Modulen, die als Ordner dargestellt werden, die Prozessgebiete für den Benutzer zur Auswahl. Dabei existieren für jedes Modul unterschiedliche Prozessgebiete. Ein User sieht nur die Module und Prozessgebiete, für die er die Zugriffsberechtigungen hat. Nach der Auswahl eines Prozessgebietes wird, auf der rechten Seite, die Master Table mit Daten befüllt. Dabei besteht die Master Table View aus einer nicht editierbaren Übersichtstabelle und Filtereinstellungen, die auf die Übersichtstabelle angewendet werden können. Dazu wird in der Tabelle nicht nur Text dargestellt, sondern auch Farbblöcke und Buttons. Jede Master Table eines Prozessgebiets entspricht einer View.

Nach Auswahl eines Datensatzes werden die Detail Tabs mit Daten befüllt. Dies bedeutet, dass die relevanten Daten aus der Datenbank geladen werden. Die Detailansichten sind immer dem Prozessgebiet angepasst und unterstehen ständigen Änderungen.

Error! Use the Home tab to apply Überschrift 1 to the text that you want to appear here.

Es gibt verschiedene Tabs, welche dieselben Daten anzeigen können, um dem Benutzer eine bessere Übersicht zu gewährleisten. Dabei ist die Konsistenz der Daten wichtig. Dies bedeutet, sobald Daten auf einem Tab geändert werden, müssen alle anderen Felder, die dieses Datenfeld ebenfalls enthalten, aktualisiert werden. Im unteren Bereich der Tabs kann der Benutzer verschiedene Operationen ausführen. Dies ist nur möglich, wenn die zugeordneten Validierungen erfolgreich ausgeführt werden und somit alle Daten korrekt sind.

Eine Validierung unterstützt den Benutzer und hilft ihm korrekte Daten einzugeben. Das ist w*ichtig für eine gute Software Ergonomie.* (Dahm 2006, Seite 152) Es gibt Unterschiede in den Arten von Validierungen. Einerseits gibt es Validierungen ohne Abhängigkeit von anderen Daten, wie bspw. die gültigen Zeichen eines Passworts. Andererseits gibt es Validierungen, die nicht nur auf einem einzelnen Feld, sondern auf einer Gruppe von Daten und Abhängigkeiten zwischen ihnen arbeiten. Diese Abhängigkeiten werden Business Rules genannt. Sie beschreiben die Abhängigkeiten zwischen verschiedenen Daten. Beispielsweise könnte folgende Business Rule existieren: Minimum ≤ Bestand ≤ Maximum. Da diese Validierung von mehreren Eigenschaften oder auch Objekten abhängen kann, ist die Implementierung nicht so trivial wie bei den unabhängigen Validierungen. Hier müssen immer alle Abhängigkeiten betrachtet werden und ggf. Validierungen auf allen abhängigen Feldern durchgeführt werden.

Es muss darauf geachtet werden, dass die Validierungen nicht zu viel Zeit in Anspruch nehmen, damit *die UI möglichst kurze Antwortzeiten besitzt und die Kriterien einer guten Softwareergonomie erfüllt.* (Dahm 2006, Seite 96)

Alle Operationen die längere Zeit in Anspruch nehmen, wie das Laden und Validieren der Daten, werden in einem separaten Thread ausgeführt. Zum Beispiel kann die Validierung von Daten aus einer Datenbank abhängig sein und mehrere Sekunden in Anspruch nehmen. Dabei würde die UI blockiert werden. Dabei kann der Benutzer jedoch nicht informiert werden, dass die Aktion längere Zeit in Anspruch nehmen würde. Aus Software ergonomischen Aspekten sollte das Programm aber immer *eine Rückmeldung geben.* (Dahm 2006, Seite 155)

3.1.2 ARCHITEKTUR DER UI

Insgesamt besteht die UI aus 196 einzelnen Views. Diese werden in den verschiedenen Bereichen Navigation, Master Table und den Detail Tabs dargestellt. Um die Wartbarkeit zu erhöhen, wurde für die Views eine gleichartige Architektur eingeführt: das Design, die View und der Controller. Die drei Komponenten schaffen einen einheitlichen Aufbau für eine View. Ein Nachteil ist jedoch, dass die UI Logik für jede View implementiert werden muss und nicht gekapselt sowie unabhängig ist. Durch die Dezentralisierung der Logik ist es schwer, das Verhalten in jeder View gleichartig zu implementieren. Wenn das Verhalten einer View geändert wird, müssen alle anderen Views ebenfalls angepasst werden. Dadurch kann keine Einheitlichkeit gewährleistet werden. *Diese ist jedoch wichtig für eine gute Software Ergonomie* (Dahm 2006, Seite 155).

Aus den oben erläuterten Gründen haben sich im Laufe der Zeit die Views unterschiedlich entwickelt und sind jetzt nicht mehr gleich in ihrem Verhalten. Aus diesem Grund hat der Kunde eine neue Anforderung gestellt, in der er ein einheitliches und durchgängiges UI Verhalten fordert. Um dieses zu erreichen wurde das ePEP UI Framework entwickelt. Ziel des Frameworks ist es einerseits, die Logik von der UI zu trennen und zu zentralisieren und andererseits, das Entwickeln neuer Views zu erleichtern. Durch eine zentrale Logik können alle Views dieselbe Logik verwenden und entsprechen damit den Anforderungen des Kunden.

Mithilfe von Generatoren wurde die Logik zentralisiert. Die Generatoren ermöglichen es, Klassen durch deskriptive Programmierung automatisch generieren zu lassen. Die Logik für jede UI Komponente wird im Generator implementiert. Dadurch ist es möglich die Logik im Generator zu ändern und die Modelle neu generieren zu lassen, womit alle Modelle das gleiche, neue Verhalten haben.

Durch die Implementierung von View und Logik in denselben Klassen, konnte das ePEP UI Framework nur eingeführt werden, weil Logik und UI zuerst voneinander getrennt oder neu implementiert wurden. Ein solches Vorgehen sollte nur durchgeführt werden, wenn es durch ein Refactoring unterstützt wird. Dies birgt Risiken und ist mit einem hohen Zeit- bzw. Kostenaufwand verbunden. Deswegen sind bisher nur 49 der 196 Views auf das neue Framework umgestellt.

Die UI wurde mithilfe des Presentation Model Patterns implementiert. Das Pattern bietet die Möglichkeit, Logik und UI voneinander zu trennen. *Dabei wird eine zusätzliche Schicht zwischen View und Domain Objekten implementiert, die den Status einer View repräsentiert. Dieser Status wird mit der UI synchronisiert.* (Fowler, Presentation Model 2004)

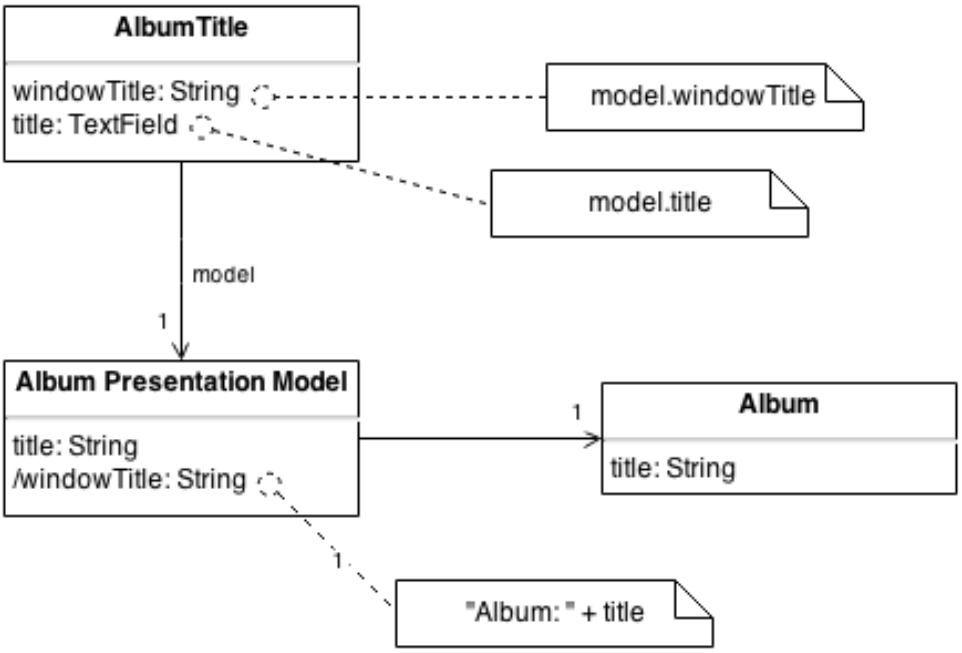

Abbildung 3: Presentation Model Pattern (Fowler, Presentation Model 2004)

Zu einer View gehört genau ein Presentation Model in dem die UI-Logik implementiert und auf die Datenfelder in den JavaBeans referenziert wird. Das Presentation Model Pattern kapselt einerseits die UI-Logik von der View, andererseits existiert durch dieses Pattern auch eine klare Abtrennung zur Datenschicht der Anwendung. Ein weiterer Vorteil des Presentation Model Patterns ist, dass der Status einer View persistiert werden kann, was in ePEP jedoch von geringer Bedeutung ist. Darüber hinaus ist interessant, dass das Pattern nicht vorgibt, wie Daten synchronisiert werden. Einerseits birgt dies potentielle Fehlerquellen, da das Pattern falsch benutzt werden könnte. Andererseits werden dem Entwickler auch mehr Freiheiten eingeräumt. Anzumerken ist, dass die Nachteile von Third Party Bibliotheken existieren (Vgl. Kapitel 2.1.2).

Um in ePEP die Daten auf dem Presentation Model an der UI anzuzeigen, wird ein Framework namens JGoodies Binding[8] verwendet. Mit dem Framework können verschiedene Datenfelder aneinander gebunden werden. Dadurch enthalten diese immer die aktuellen Werte des jeweils anderen Feldes (Vgl. Kapitel 3.2.1).

3.1.3 Schnittstellen zu Swing und abhängigen Komponenten

Die Views, die noch nicht auf das ePEP UI Framework migriert wurden, können nur von Hand mit viel Aufwand und Risiken auf ein neues UI Framework umgestellt werden. Dies sollte durch ein Refactoring unterstützt durchgeführt werden. Deswegen werden im weiteren Verlauf nur die Schnittstellen der Views betrachtet, die bereits mit dem ePEP UI Framework implementiert sind.

Durch das ePEP UI Framework existieren klar definierte Schnittstellen vom Kern zu Swing und der abhängigen Komponente JGoodies Binding.

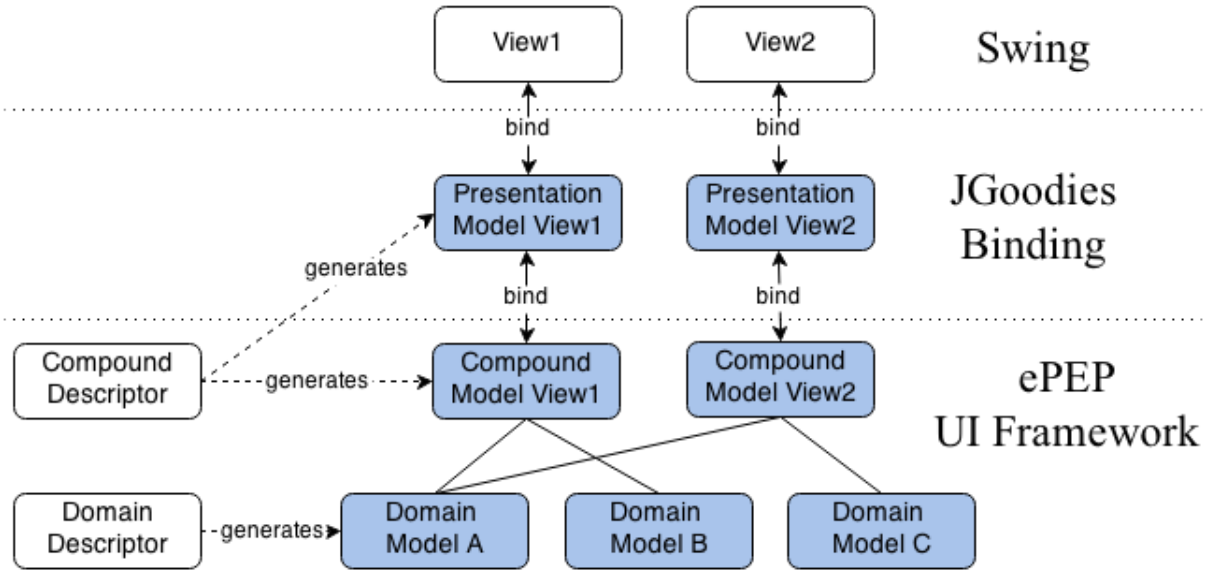

Abbildung 4: ePEP UI Framework

Durch die Domain Descriptoren des ePEP UI Frameworks werden JavaBeans generiert. Die Domain Model Klassen repräsentieren eine Entität aus dem logischen Datenmodell.

[8] http://www.jgoodies.com/freeware/libraries/binding/

Der Compound Descriptor generiert einerseits die Compound Model Klassen und andererseits das Presentation Model. Die Compound Model Klassen sind eine extra Schicht, die verschiedene Domain Models zusammenfasst, obwohl laut Presentation Model Pattern diese Aufgabe dem Presentation Model zufällt. Da aber das Presentation Model des JGoodies Binding Frameworks nur eine JavaBean verwalten kann, delegieren die Compound Models Änderungen des Presentation Models zu den verschiedenen Domain Models und umgekehrt.

Das Presentation Model repräsentiert den Status einer View. Zusätzlich, zu den einfachen gettern und settern der Datenfelder, enthält das Presentation Model ebenfalls Objekte, welche für die Validierung, visible und editable Eigenschaften der Felder zuständig sind. Eventlogik kann ebenfalls im Presentation Model implementiert sein.

Die Views werden mit Swing von Hand implementiert. Hierbei wird kein UI Builder Tool verwendet. Dadurch muss ein Programmierer viel Erfahrung haben, um eine komplexe UI aufzubauen.

3.1.4 Ermitteln der Schwächen von Swing

Die IST-Analyse der UI zeigt die Schwächen von Swing auf. Diese Schwächen werden weiter untersucht, um eine Basis zu haben, auf der die Vorteile eines neuen UI Frameworks analysiert werden können. Folgende Schwächen existieren in Swing:

- ***Data Binding***: Data Binding in Swing ist entweder durch eine Third Party Bibliothek verfügbar oder durch das von Oracle nachträglich programmierte Scene Builder[9] Tool. Das Projektteam hat sich dafür entschieden, die Bibliothek JGoodies Binding einzusetzen. Damit können nahezu alle Swing Komponenten gebunden werden. Der Nachteil bei diesem Framework ist, dass nicht alle Komponenten, wie die JProgressBar, in dem Framework umgesetzt wurden. Dadurch ist eine eigene Implementierung notwendig, um das Data Binding zu realisieren. Weiter kann das Presentation Model des Frameworks nur eine Model-Klasse verwalten. Dadurch ist eine extra Schicht notwendig, welche die Events zwischen dem Presentation Model und den benutzten Model-Klassen weiterleitet.

[9] http://docs.oracle.com/javafx/scenebuilder/1/user_guide/jsbpub-user_guide.htm

- ***Form Validierung***: Die Form Validierung wird von Swing, wenn über einfache Zeichen Validierung hinausgegangen wird, kaum unterstützt. Für den Entwickler gibt es keine Unterstützung wenn Validierungen im Hintergrund durchgeführt werden. Gleiches trifft zu, wenn Business Rules validiert werden sollen.
- ***Threads***: Das Organisieren der Threads wird in Swing weitestgehend dem Entwickler überlassen, was eine Herausforderung darstellt. Es gibt Hilfen, wie die *invokeLater() Funktion, die es ermöglicht, UI-Komponenten aus einem anderen Thread als dem* Event Dispatch Thread *heraus zu manipulieren.* (Ullenboom 2012, Kapitel 19.26.3) Mit dem Swing Worker Objekt können Methoden im Hintergrund ausgeführt werden. Das Objekt kann ein Statusupdate in den UI-Thread posten, sowie einen Rückgabewert zurückgeben, wenn die Operation beendet wurde. Ein Nachteil ist, dass der Worker ein Abbruch Kriterium selbst implementieren muss und es nicht möglich ist bspw. einen Callback zu definieren der aufgerufen wird, wenn der Swing Worker abgebrochen wurde.

3.2 SCHWACHSTELLENANALYSE VON SWING

In diesem Kapitel werden die in Kapitel 3.1.4 ermittelten Schwächen von Swing genauer analysiert. Ziel ist es, im weiteren Verlauf der Arbeit, zu untersuchen, ob ein anderes UI Framework die aufgezeigten Schwächen ausbessern könnte.

3.2.1 DATA BINDING

Eine ermittelte Schwäche von Swing ist das Data Binding. *Dies ermöglicht es eine direkte und unmittelbare Beziehung zwischen zwei Variablen herzustellen, wobei das Ändern einer Variablen auch eine Änderung der anderen Variablen zur Folge hat.* (Premkumar 2010, Kapitel 9.1.) Dadurch werden inkonsistente Daten vermieden, da, wie in Abbildung 5 dargestellt, gebundene Daten in verschiedenen Objekten automatisch und unmittelbar aktualisiert werden. Data Binding kann manuell durch verschiedene Mechanismen, wie dem Observer Pattern[10], realisiert werden. Die Arbeit eines Entwicklers wird jedoch vereinfacht, wenn eine integrierte Lösung verwendet werden kann. Genauer beschreiben das Gail und Paul Anderson indem sie sagen, dass *Data Binding weniger Code produziert, der Code nicht*

[10] Das Observer Pattern ist ein von der Gang-of-Four entwickeltes Entwurfsmuster zur Weitergabe von Änderungen an einem Objekt zu Beobachtern des besagten Objekts.

so fehleranfällig, besser zu warten und oft einfacher für den Compiler zu optimieren ist. (Anderson und Anderson 2009, Kapitel 2.6)

Es wird unterscheiden zwischen uni- und bidirektionalem Data Binding. Bei einer unidirektionalen Bindung existieren jeweils eine Quelle und ein Ziel. Alle an der Quelle vorgenommenen Änderungen werden an das Ziel weitergeleitet, jedoch nicht umgekehrt. Hingegen ist bei einer bidirektionalen Bindung die Quelle gleichzeitig Ziel und umgekehrt.

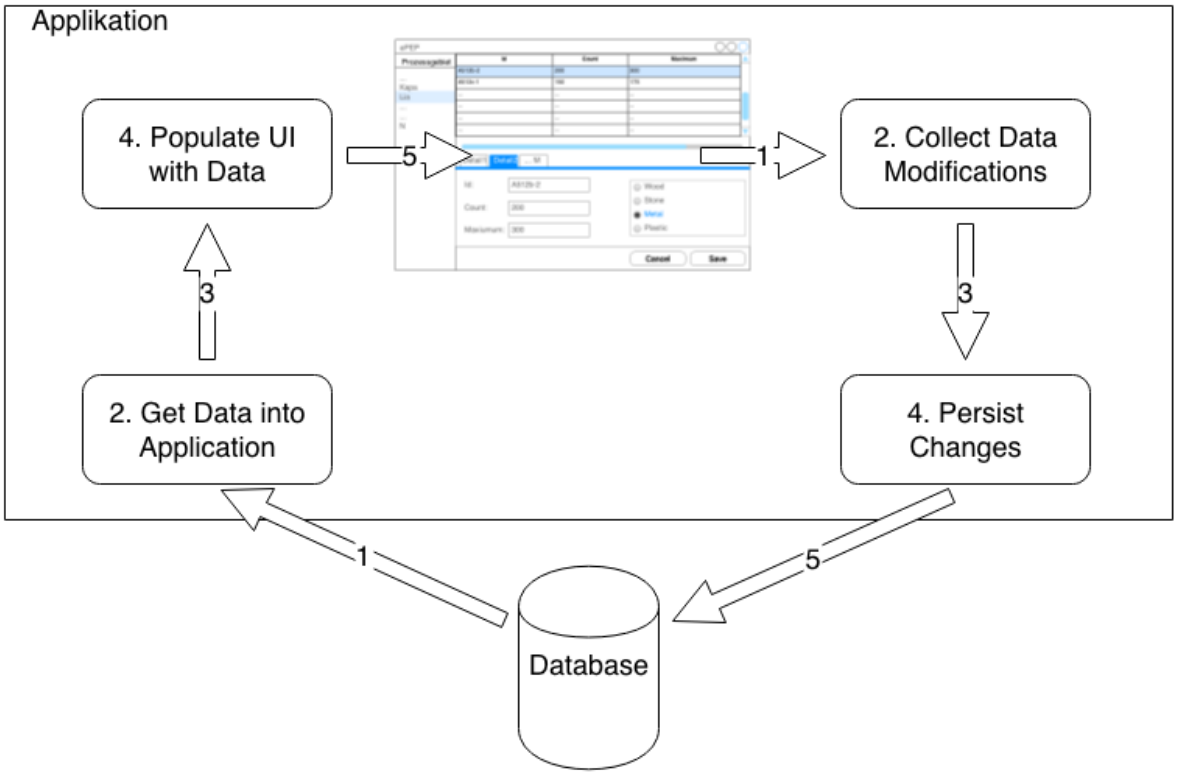

Abbildung 5: Zyklus bei einer Datenänderung mit Data Binding (Noyes 2006, Kapitel 1, Grafik 1.1.)

Der Zyklus stellt dar, was durch das bidirektionale Data Binding ausgelöst wird, sobald eine Änderung der Daten im Datenmodell oder in der Applikation auftritt:

1. Quelle stellt fest, dass eine Datenänderung vorliegt
2. Sammeln der Änderungen
3. Die Geänderten Daten werden von der Quelle an die Ziele übergeben
4. Wenn Daten sich unterscheiden werden die Änderungen übernommen, ansonsten werden die Änderungen verworfen
5. Daten persistieren, fange wieder bei 1. an

Bevor die Änderungen bei 4. übernommen werden, muss geprüft werden, ob sich die Daten tatsächlich geändert haben, da sonst eine Endlosschleife entstehen kann.

3.2.2 FORM VALIDIERUNG

Die Form Validierung wird in Swing nicht unterstützt. Form Validierung beschreibt den Vorgang festzustellen, ob Daten korrekt sind. Ist dies nicht der Fall, soll der Benutzer darauf aufmerksam gemacht werden, dass fehlerhafte Daten vorliegen. Gegebenenfalls darf der Vorgang nicht abgeschlossen werden, bis die Fehler korrigiert wurden.

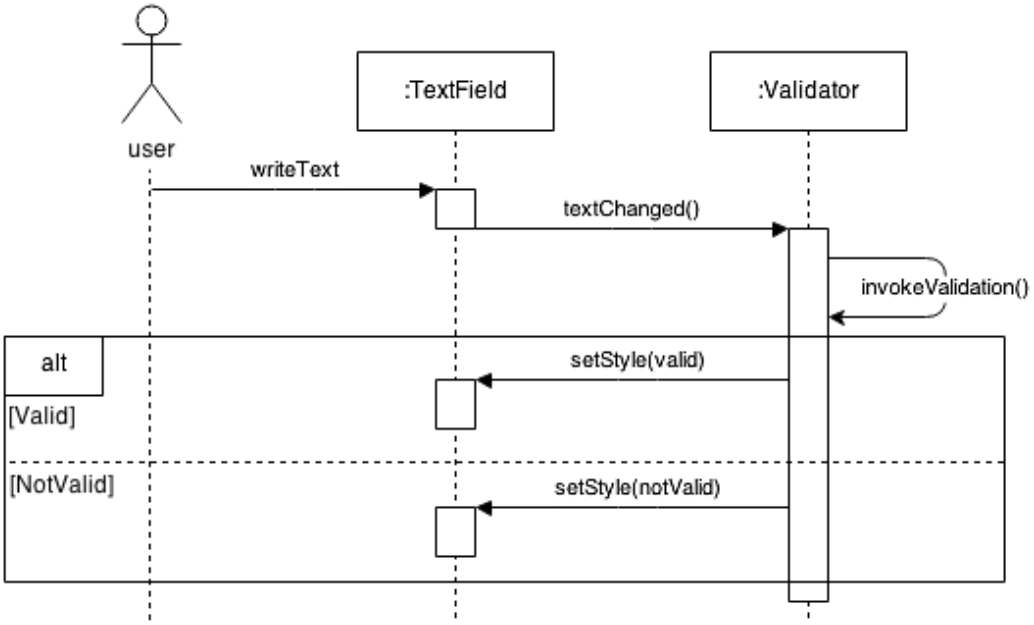

Abbildung 6: Ablauf einer Validierung

Die Abbildung stellt den Ablauf einer Validierung anhand eines Textfeldes dar. Dabei ändert der Benutzer den Text, woraufhin das Textfeld den Validator benachrichtigt, dass eine Änderung vorliegt. Dieser prüft die Änderung und setzt, je nach Ergebnis der Prüfung, den Style des Textfeldes um, damit der Benutzer eine Rückmeldung bekommt. Gegebenenfalls sollte eine Information dargestellt werden, die dem Benutzer unterstützt, die Fehler zu korrigieren.

3.2.3 THREADS (IN BEZUG AUF UI)

Die Unterstützung, die Swing für das Programmieren mit Threads liefert, ist gering. *Ein Thread ist eine unabhängige Befehlsfolge in einem Programm.* (Nagel, et al. 2010, Kapitel

2.1.) Die Definition von Threads lässt nicht vermuten, dass es beim Programmieren mit ihnen Herausforderungen gibt, mit denen auch erfahrene Programmierer Probleme haben. Das Operieren von unterschiedlichen Threads auf denselben Daten stellt eine Schwierigkeit dar. Am Beispiel von Benutzerschnittstellen lässt sich zeigen, dass es zu Inkonsistenzen kommen kann. Dabei werden Benutzersteuerelemente bzw. ihre angezeigten Daten in einem anderen Thread als dem UI Thread geändert (Abbildung 7). Dabei kann es zu einer fehlerhaften Ausführung der Software kommen.

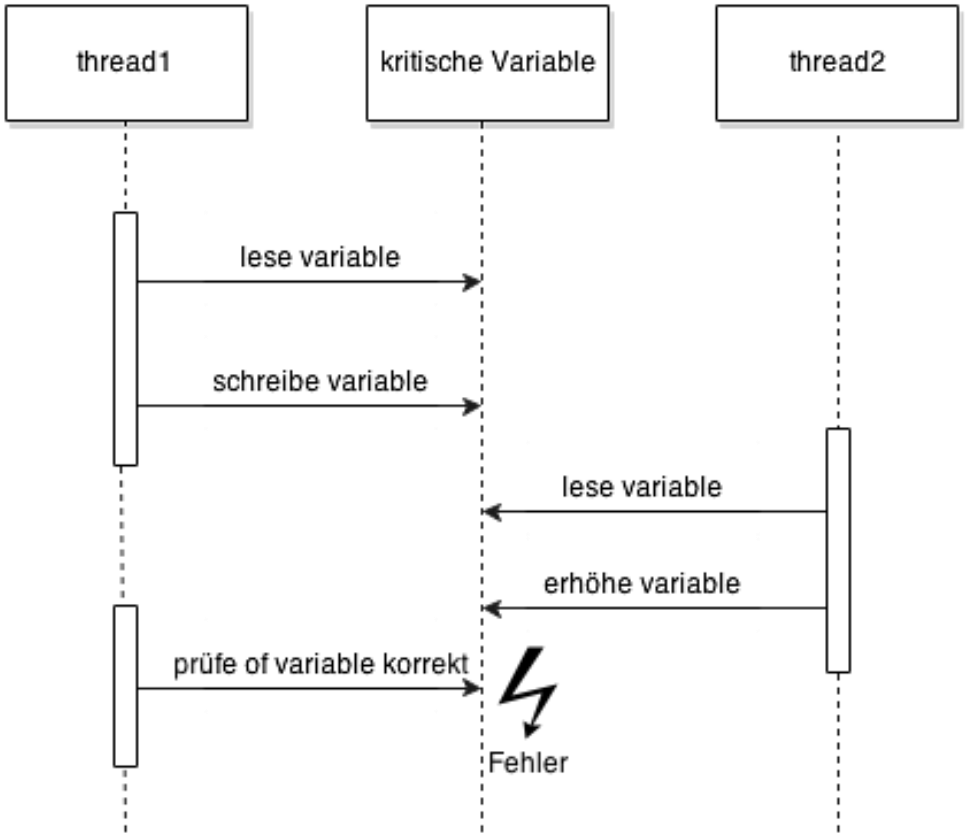

Abbildung 7: Fehlerhafter Zugriff auf eine nicht geschützte Variable in einem Thread

Das Schaubild zeigt, wie ein Thread eine Variable liest, schreibt und dann prüfen möchte, ob die Variable den korrekten Wert hat. Zwischen den Zugriffen des einen Threads erhöht ein anderer Thread die kritische Variable und damit hat die Variable nicht mehr den Wert, den der erste Thread erwartet.

Wenn der Programmierer jedoch nur einen Thread in der Anwendung benutzt, wird die Anwendung langsamer, da Befehle nicht parallel abgearbeitet werden können. Außerdem kann, während ein Befehl abgearbeitet wird, die Benutzerschnittstelle nicht neu gezeichnet

werden. Dabei entsteht das Problem, dass die UI nicht auf weitere Eingaben von dem Benutzer oder des Programms reagiert und eingefroren scheint. Dadurch ist es dem Benutzer nicht möglich weiter an seiner Aufgabe zu arbeiten, wenn der ausgeführte Befehl für den weiteren Fortschritt der Aufgabe keine Bedeutung hat.

Daraus folgt, dass auch das Programm keine Möglichkeit mehr hat, Informationen an die Benutzerschnittstelle zu schicken. Dadurch kann dem Benutzer während einer Berechnung o. ä. keine Rückmeldung über den Fortschritt der Abarbeitung gegeben werden. Wie aus den acht goldenen Regeln von Shneiderman hervorgeht, *kann das zu Verwirrung des Benutzers führen, da dieser nicht weiß, welche Funktionen das Programm grade ausführt.* (Dahm 2006, Seite 152)

Bei Threads in Bezug auf die Benutzerschnittstelle sind auch Events ein relevantes Thema. *Events bieten einen Weg, um mit dem Benutzer der Anwendung zu kommunizieren. Events können ausgelöst werden, wenn der Benutzer auf irgendeinem Weg mit dem System interagiert. Der Programmierer kann mit den gesammelten Informationen auf diese Events reagieren.* (DiMarzio 2011, Kapitel 9)

Java benutzt das Delegation Event Modell bei dem immer Quelle und Ziel existieren. *Das Objekt, auf dem das Event ausgelöst wurde, ist die Quelle und dafür verantwortlich, dass das Programm, auch Event Handler genannt, mit den nötigen Informationen versorgt wird. Der Event Handler ist das Ziel und für die Bearbeitung des Events verantwortlich.* (Walrath 2004, Seite 13-14) Die Events werden in einem Thread bearbeitet, *der dadurch nicht für andere Aufgaben, wie z. B. dem Zeichnen der UI zur Verfügung steht*. (Walrath 2004, Seite 82)

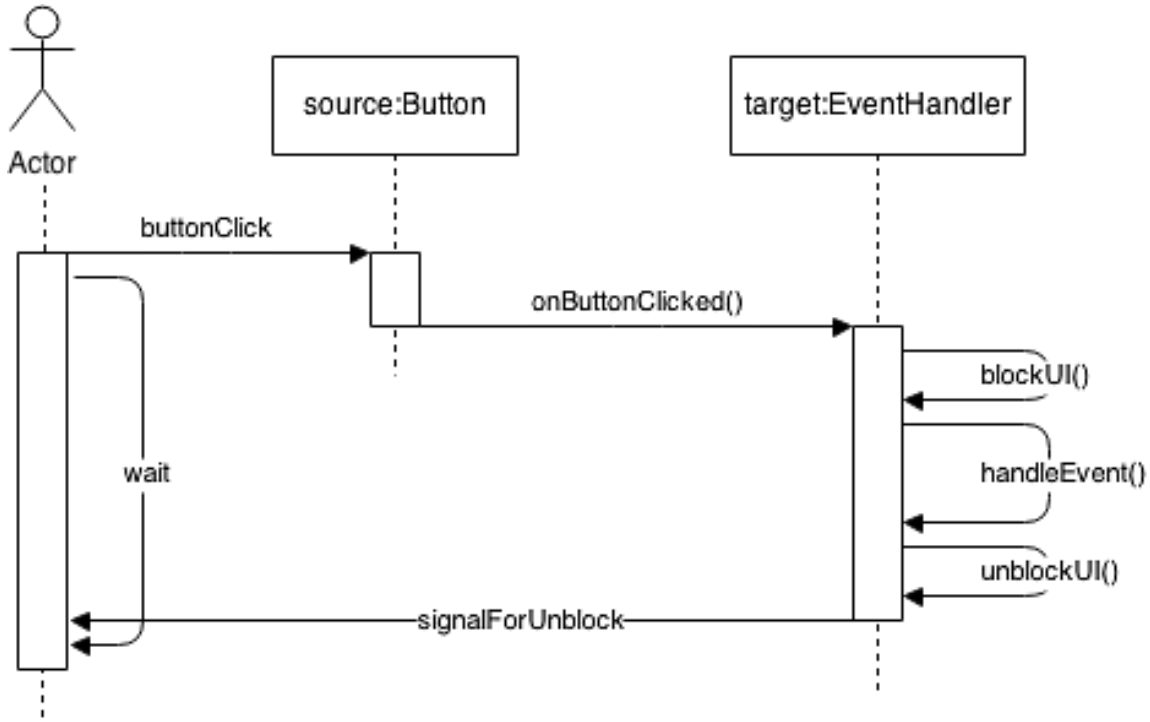

Abbildung 8: Ablauf nach Auslösung eines Events vom Benutzer

Bei dieser Abbildung löst der Benutzer ein Event aus, indem er auf einen Button klickt. Der Button ist die Eventquelle und übergibt die Weiterbearbeitung an das Ziel, den EventHandler. Dieser blockiert die UI, behandelt das Event, hört auf die UI zu blockieren und signalisiert dies dem Benutzer. Während dieser Prozedur muss der Anwender warten, bis er mit der nächsten Interaktion der Erfüllung seiner Aufgabe näher kommt. Wenn die Abarbeitung des Events zu lange dauert, entstehen die bereits genannten Nachteile.

4 BEWERTUNG ALTERNATIVER UI FRAMEWORKS

In diesem Kapitel werden alternative Frameworks evaluiert. Nach der Evaluierung der Alternativen wird die mit dem meisten Mehrwert detailliert in Bezug auf die Schwächen von Swing untersucht.

4.1 VORSTELLUNG UND WAHL ALTERNATIVER FRAMEWORKS

Nach der Vorstellung von alternativen Frameworks zu Swing werden diese in Bezug auf ihren Mehrwert miteinander verglichen. Dabei sind die in Kapitel 3.1 beschriebenen Vorgaben, Java sowie den WAS zu nutzen, einzuhalten.

4.1.1 SWT

Mit dem von Eclipse entwickelten Standard Widget Toolkit (SWT) können Anwendungen programmiert werden, die *auf verschiedenen Betriebssystemen ausgeführt werden können und die UI dabei auf den nativen Benutzerschnittstellen der Betriebssysteme aufsetzen.* (Northover und Wilson 2004, Kapitel 1) SWT ist der Hauptkonkurrent von Swing und damit auch ein etabliertes Framework auf dem Markt der Softwareentwicklung. Für SWT existieren, wie bei Swing auch, Third Party Bibliotheken, die Data Binding zur Verfügung stellen. Mit der Zeit hat sich eine große Community aufgebaut, die SWT benutzt und dem Entwickler Unterstützung bietet.

4.1.2 AWT

AWT wurde in der ersten Version von Java 1995 ausgeliefert und ist der Vorgänger von Swing. Das Abstract Window Toolkit wird, wie SWT, für die plattformunabhängige Entwicklung eingesetzt. Auch AWT *greift auf die nativen Schnittstellen des Betriebssystems zu. AWT bietet wenige Komponenten, da jedes AWT-Widget*[11] *auf ein Widget der Betriebssysteme abgebildet wird.* (Ullenboom 2012, Seite 914) *Außerdem werden Features wie Tool-Tips und Icons auf Widgets nicht unterstützt.* (Ullenboom 2012, Kapitel 19.1.4)

[11] Ein Widget ist ein Benutzersteuerelement auf einer Oberfläche.

4.1.3 JAVAFX

JavaFX wird von Oracle offiziell als Nachfolger von Swing gehandelt. (JavaFX Frequently Asked Questions) Mit JavaFX können sowohl RIAs als auch Rich-Client Applikationen erstellt werden. Dadurch ist es möglich JavaFX Applikationen auch im Browser und auf mobilen Geräten auszuführen und dabei weiterhin in Java zu programmieren.

JavaFX bietet Unterstützung für das Data Binding, jedoch nicht für Validierungen. Es ist möglich Swing-Anwendungen mit JavaFX zu kombinieren. Dabei können JavaFX Komponenten in existierenden Swing Applikationen verwendet werden. Es ist möglich die UI deklarativ[12] mit FXML zu programmieren und UI Styles mit CSS3 zu implementieren. FXML ist eine auf XML basierende Sprache.

4.1.4 BEWERTUNG DER ALTERNATIVEN

Eine Gegenüberstellung der Features von Swing und alternativen Frameworks wird in Tabelle 2 gezeigt. Dabei werden acht verschiedene Kriterien betrachtet, die in Bezug auf ePEP eine Rolle spielen. Als Vergleichsbasis dienen die Features der UI Frameworks. D.h. Third Party Bibliotheken finden keine Berücksichtigung, da so wenige Abhängigkeiten wie möglich zu Third Party Bibliotheken existieren sollen. Hier hätten Swing, SWT und auch AWT allerdings einen Vorteil zu JavaFX.

[12] Eine deklarative Oberfläche ist eine Oberfläche deren Komponenten beschrieben und nicht direkt implementiert werden. Dadurch ist die Oberfläche unabhängig von der Programmiersprache.

	Swing	SWT	AWT	JavaFX
Verfügbare Komponenten	✓	✓	✗	✗
Schnelligkeit	✓	✓	✓	✓
Deklarative UI	✗	✗	✗	✓
Data Binding integriert	✗	✗	✗	✓
Rich-Clients und RIAs	✗	✗	✗	✓
Daten Validierung	✗	✗	✗	✗
Threadsicherheit	✗	✗	✓	✗
Native Ausführung	✗	✓	✗	✗

Tabelle 2: Gegenüberstellung Swing und Alternativen

Bei der Betrachtung der Tabelle 2 fällt auf, dass keine der Alternativen alle Anforderungen des Projekts ePEP erfüllt. Dies ist ein generelles Problem der Softwareentwicklung. Durch die vielen unterschiedlichen Anforderungen einer Software, ist es kaum möglich ein Framework zu finden, welches den Entwickler bei der Umsetzung aller Anforderungen unterstützt. Umso wichtiger ist es die Entscheidung für ein Framework fundiert zu treffen.

Swing hat AWT abgelöst, d.h. AWT ist veraltet. AWT bietet weniger Komponenten, welche zudem nur mit Aufwand benutzerdefiniert gestaltet werden können. Die Logik und Daten werden nicht von einander getrennt. (Ullenboom 2012, Kapitel 19.1.4) Einen Vorteil bietet AWT jedoch: Threadsicherheit von Komponenten. Das bedeutet, dass *„mehrere Threads zur gleichen Zeit Methoden der AWT Komponenten aufrufen können.“* (Ullenboom 2012, Kapitel 19.1.4)

Ein Vorteil von SWT ist, dass die *Anwendungen mit nativem Quellcode ausgeführt werden und sich so der Ausführungsumgebung anpassen.* (Hatton 2005, Seite 1) Dies jedoch stellt für ePEP kein relevantes Auswahlkriterium dar, da alle Clients auf der gleichen Umgebung arbeiten. SWT bietet im Gegensatz zu Swing also keine Features, die einen Umstieg der Technologie in ePEP rechtfertigen würden.

JavaFX bietet mit einem integrierten Data Binding und dem deklarativen Ansatz bei der UI Programmierung, Argumente, die für die Umstellung von Swing auf JavaFX sprechen. JavaFX ist außerdem ein Framework mit dem sowohl Rich-Clients als auch RIAs entwickelt werden können. Jedoch stehen in Swing mehr Komponenten zur Verfügung. Da JavaFX aber ständig weiterentwickelt wird, kann davon ausgegangen werden, dass fehlende Features in Zukunft implementiert werden. Zusätzlich können Swing-Anwendungen mit JavaFX kombiniert werden, was eine schrittweise Umstellung von ePEP unterstützen würde. Dadurch könnte ePEP auch nach und nach an Webapplikation herangeführt werden. Das sollte ein langfristiges Ziel für das Projekt sein, um die Vorteile der RIAs nutzen zu können. Die Möglichkeit, außer Java auch HTML und JavaScript zu benutzen, ist ebenfalls attraktiv.

Bei der Schnelligkeit der Frameworks besteht kein Unterschied, der für das Projekt ePEP relevant wäre. Ebenfalls wird die komplexe Validierung der Daten von keinem der Frameworks unterstützt.

Das Ergebnis der Evaluierung ist, dass JavaFX das einzige Framework ist, welches bei einer Umstellung einen Mehrwert für das Projekt ePEP bringen kann. Natürlich wurden hier nur Frameworks betrachtet die den gegebenen Kriterien, Java und den WAS zu benutzen, entsprechen. Die Nachteile von AWT überwiegen, sodass der Vorteil der Threadsicherheit der Komponenten nicht überzeugend ist. SWT bietet lediglich den Vorteil der nativen Ausführung, der jedoch bei ePEP nicht relevant ist.

Wegen der genannten Kriterien wird JavaFX als Alternative zu Swing im weiteren Verlauf der Arbeit detailliert untersucht.

4.2 Evaluierung der Schwächen von Swing bei JavaFX

Für den Vergleich von Swing und JavaFX werden die in Kapitel 3.1.4 herausgearbeiteten Schwächen verwendet.

4.2.1 DATA BINDING

Durch die Integration des Data Bindings in JavaFX, fallen die Nachteile eines Third Party Frameworks weg. Die Umsetzung wird über sogenannte Properties realisiert. Properties sind Objekte, die uni- und bidirektional an andere Properties und auch Observable Values gebunden werden können.

Dabei ist nur eine unidirektionale Bindung in der Richtung von Observable Value zu der Property erlaubt. Dies bedeutet, dass Änderungen an der Property, die an ein Observable Value gebunden ist, ungültig sind und eine Runtime Exception ausgelöst wird. (Weaver, et al. 2012, Kapitel 3)

Die Property Objekte sind gleichzeitig Wrapper Klassen für den Wert, der durch das Data Binding geändert wird. Auf diesen Wert kann über getter und setter zugegriffen werden. JavaFX liefert vorgefertigte Property Klassen, wie die StringProperty-Klasse, die alle Primitiven und den Objekt Datentyp in Java implementieren. Die JavaFX Controls besitzen verschiedene Properties die ebenfalls gebunden werden können (Abbildung 9).

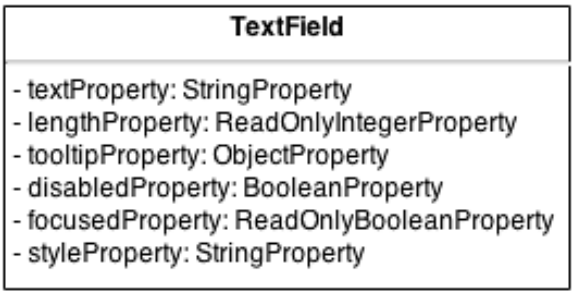

Abbildung 9: Beispiele für TextField Properties

Wichtig ist, dass die Werte, z. B. von einem Textfeld, nur über die Properties geändert werden können. Dadurch synchronisieren sich alle Objekte, die eine Bindung zu der Property haben, automatisch.

Die JavaFX Properties unterstützen auch das sogenannte Lazy Binding. Das bedeutet, *dass eine Property nach einer Wertänderung ein Event auslöst. Das Event benachrichtigt gebundene Properties. Diese aktualisieren ihren Wert jedoch nicht sofort, sondern markieren ihn als ungültig. Der Wert wird aktualisiert, sobald er abfragt wird. Dadurch ist es möglich, dass die Quelle den Wert beliebig oft ändert, jedoch nur ein einziges Mal im Ziel aktualisiert*

wird. (Weaver, et al. 2012, Kapitel 3) Dieses Verhalten ist wünschenswert, wenn eine große Zahl von Abhängigkeiten existiert. Durch viele Abhängigkeiten kann es längere Zeit in Anspruch nehmen bis alle Daten aktualisiert sind, obwohl sie u.U. gar nicht abgefragt oder angezeigt wurden. Des Weiteren reduziert das Lazy Binding auch den Datenverkehr, da bei mehreren Änderungen der Daten, die abhängigen Daten nur einmal aktualisiert werden.

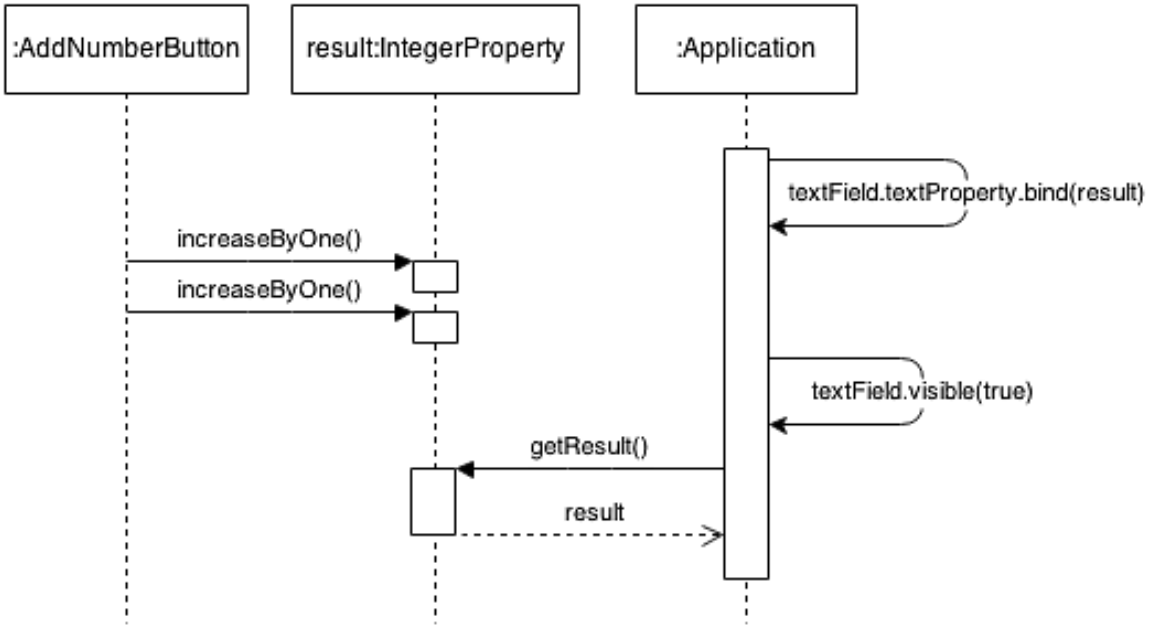

Abbildung 10: Lazy Data Binding

In der Grafik wird ein Beispiel für die Lazy Binding Verhaltensweise dargestellt. Die Applikation bindet die IntegerProperty an ein Textfeld, das momentan jedoch nicht sichtbar ist. Ein Button, der von dem Benutzer betätigt werden kann, erhöht die IntegerProperty. Nach diesem Schritt würde die Application ohne Lazy Data Binding den Wert aktualisieren. Der Wert wird jedoch erst aktualisiert, nachdem das TextField sichtbar gemacht wurde.

Ein weiteres, wichtiges Konstrukt im Zusammenhang mit Data Binding sind die JavaBeans, welche bereits in Kapitel 2.1.3 eingeführt wurden. JavaFX bringt die JavaFXBean Konvention mit sich. Diese spezifiziert, wie die JavaBeans implementiert sein müssen, um JavaFX konform zu sein. Nach außen hin unterscheiden sich die Schnittstellen geringfügig. Es existiert lediglich eine weitere Methode die eine Property als Rückgabetyp hat. Das ist wichtig, da die neuen JavaFXBeans damit auch kompatibel zu den alten JavaBeans sind.

Somit funktionieren Anwendungen, die auf alten JavaBeans arbeiten, ebenfalls mit den JavaFXBeans (Abbildung 11).

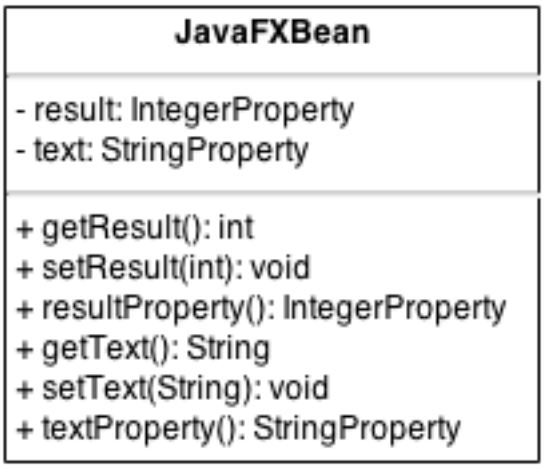

Abbildung 11: Klassendiagramm JavaFXBean

In den JavaFXBean-Klassen werden primitive Datentypen und Objekte durch die Property-Klassen ersetzt, z. B. wird ein String zu einer StringProperty. Diese Properties bilden eine Wrapper-Klasse für primitive Datentypen sowie Objekte und sind, wie zuvor beschrieben, für das Data Binding zuständig.

Das für ePEP genutzte Presentation Model Pattern wird durch das Model-View-ViewModel (MVVM) Pattern ersetzt. Das Pattern wurde ursprünglich für WPF[13] und Silverlight[14] Anwendungen entwickelt, um die Vorteile dieser beiden Frameworks zu nutzen. Dabei baut das MVVM Pattern auf dem Presentation Model Pattern auf (Vgl. Kapitel 3.1.2). Es existieren, wie bei dem Presentation Model Pattern, die View und das Model. Jedoch wird das Presentation Model durch das ViewModel ersetzt. Der einzige Unterschied ist, dass *die Kommunikation von View und ViewModel über Data Binding und Commands[15] stattfindet.* (Brumfield, et al. 2011, Kapitel 5)

[13] http://msdn.microsoft.com/de-de/library/aa970268(v=vs.110).aspx

[14] http://www.microsoft.com/silverlight/

[15] Commands werden auf der UI repräsentiert und können von einem Benutzer ausgelöst werden. (Brumfield, et al. 2011, Kapitel 5)

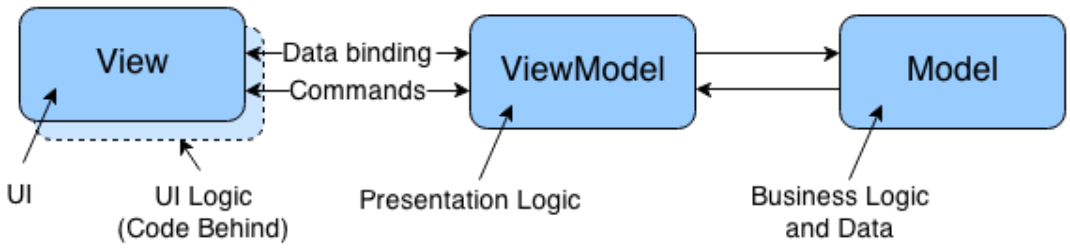

Abbildung 12: MVVM-Pattern vgl. (Brumfield, et al. 2011, Kapitel 5)

Die Grafik zeigt das MVVM Pattern, wie es für WPF und Silverlight entwickelt wurde. Die View kapselt die UI und die UI Logic, wie z.B. Zeichenoperationen. Das ViewModel beinhaltet die Logik für die Anzeige und den Status einer View. Aus dem Model kommen die Daten und die Business Logik. Die Kommunikation zwischen View und ViewModel findet über Commands sowie über Data Binding statt. Nicht nur Daten werden von der View an das ViewModel übertragen, sondern auch Commands. Ein Beispiel für ein Command ist ein Button-Klick, den der Benutzer durchführt. Die Reaktion des Programms darauf wird im ViewModel implementiert. Wie das ViewModel mit dem Model kommuniziert, bleibt dem Entwickler überlassen.

Mit JavaFX kann dieses Pattern nicht direkt verwendet werden, da bspw. das ViewModel die Komponenten der View implementiert, was für das MVVM Pattern nicht vorgesehen ist. Genauer wird das bei der Vorstellung des Prototyps erläutert (Vgl. Kapitel 5.3).

4.2.2 Form Validierung

Sowohl Swing, als auch JavaFX, unterstützt die Programmierer bei der einfachen Feld Validierung. Eine Validierung, bspw. mit der Einschränkung von festen Werten oder Formaten, ist bei beiden Frameworks einfach umzusetzen. Ein Beispiel dazu ist im Prototyp implementiert und wird näher in Kapitel 5.4 erläutert. Bei den Business Rules findet der Programmierer in keinem der beiden Frameworks angemessene Unterstützung. Dies birgt Verbesserungspotential.

4.2.3 Threads in Bezug auf UI

Da Java die gemeinsame Basis von Swing sowie JavaFX ist, gibt es Gemeinsamkeiten beider Frameworks im Bereich Threading:

- UI-Änderungen dürfen nur in einem Thread durchgeführt werden – in Swing im Event Dispatch Thread (EDT) und in JavaFX im Application Thread
- Durch ein bereitgestelltes Konstrukt existiert die Möglichkeit in einem Thread, welcher nicht der UI-Thread ist, Daten an der UI zu ändern

Unterschiede gibt es beim Aufbau der Komponenten. *Mit JavaFX können Komponenten, die nicht in einer Scene[16] eingebunden sind, in anderen Threads geändert werden. Das heißt, Komponenten die große Datenmengen benötigen, können im Hintergrund aufgebaut werden. Nachdem die Komponente vollständig geladen wurde, kann sie an der UI angezeigt werden.* (Oracle 2013) Dadurch können Multicore-Prozessoren ausgenutzt werden und der Benutzer sieht nicht wie eine Komponente auf der UI aufgebaut wird.

Des Weiteren gibt es in JavaFX einen Render Thread *wodurch ermöglicht wird, Frame N darzustellen während Frame N+1 bereits bearbeitet wird. Dies bietet vor allem bei den Multi-Core Prozessoren einen Vorteil.* (Oracle 2013) Durch die Parallelität besteht die Möglichkeit, Befehle zu bearbeiten, die in Zukunft gebraucht werden. Dadurch kann eine schnellere Bearbeitung ermöglicht werden, was bei zahlreichen Animationen hilfreich sein kann.

JavaFX bietet die Möglichkeit zusätzlich zum normalen Threading in Java einen sogenannten Service auszuführen (Abbildung 13: Services in JavaFX). *Dieser wird in einem eigenen Thread ausgeführt und wurde dafür entwickelt, dem Anwender das Managen von Multithread-Code zu erleichtern, der mit der Benutzerschnittstelle kommuniziert. Deswegen darf ein Service auch nur aus dem Application Thread aufgerufen werden. Der Entwickler kann den Status des Services überwachen oder den Service abbrechen.* (Oracle 2014) Der JavaFX Service bietet den Vorteil, dass dieser abgebrochen werden kann, ohne selbst eine Implementierung bereitstellen zu müssen. Bei einem Abbruch oder einer Unterbrechung werden Callbacks aufgerufen, in denen der Entwickler auf die Aktion reagieren kann. Es ist möglich den Service zu parametrisieren und sich als Listener auf Events zu registrieren, die nach Beendigung des Services ausgelöst werden.

[16] Eine Scene enthält grafische Benutzersteuerelemente und wird in den Scene Graph von JavaFX eingebunden, welcher *der Startpunkt für die Konstruktion von JavaFX Anwendungen ist.* (Oracle 2013)

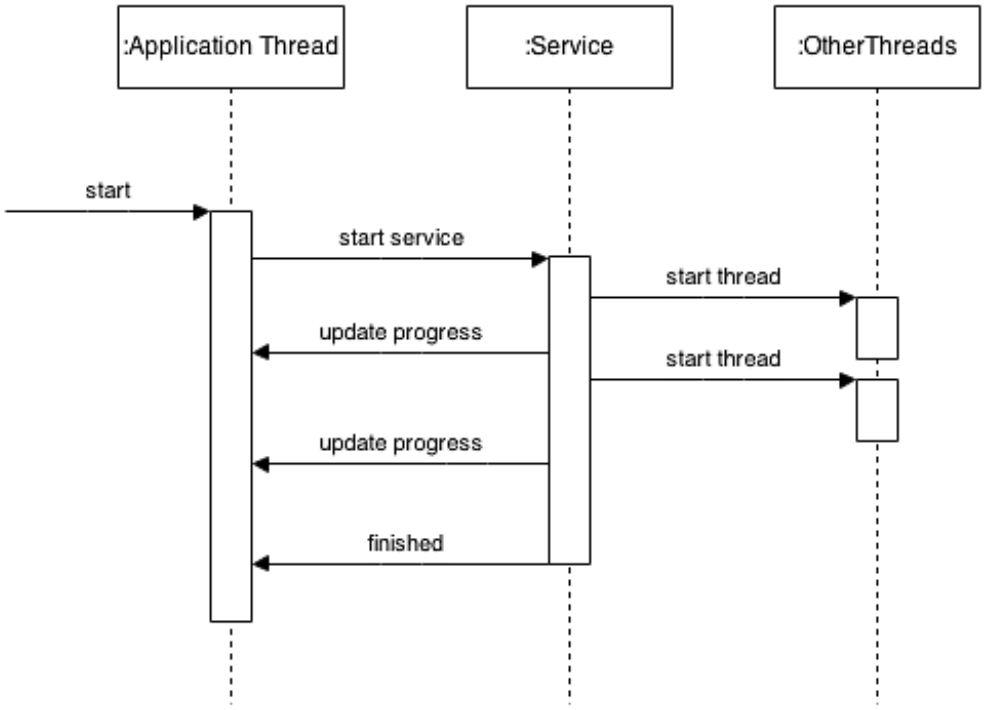

Abbildung 13: Services in JavaFX

Die Abbildung zeigt, wie der Ablauf eines aufgerufenen Services sein kann. Der Service wird aus dem Application Thread aufgerufen und in einem eigenen Thread ausgeführt. Aus dem Service heraus können weitere Threads gestartet und dem Application Thread Aktualisierungen über den Fortschritt mitgeteilt werden. Am Ende des Services benachrichtigt dieser den Application Thread.

Ein weiterer Vorteil, den JavaFX im Threading mit sich bringt, ist, dass eine Exception ausgelöst wird, sobald im Code eine UI-Komponente aus einem anderen Thread, als dem Application Thread, heraus ändert. Dieses Verhalten erleichtert die Fehlerfindung, welche sich im Zusammenhang mit Threading teilweise schwierig gestaltet. Außerdem zwingt dieses Verhalten den Programmierer zu einer sauberen Implementierung von Threads und beugt damit Fehlern vor.

5 IMPLEMENTIERUNG EINES PROTOTYPS

Ziel des Prototyps ist es zu untersuchen, ob die in Kapitel 3.1.4 ermittelten Schwächen von Swing auch in JavaFX vorhanden sind. Des Weiteren soll eine ähnliche Architektur in ePEP aufgebaut werden, um zu ermitteln, wie ePEP Schnittstellen geändert werden müssen, um JavaFX nutzen zu können.

5.1 VORGEHENSWEISE

Das Data Binding in JavaFX wird durch die Implementierung einer Form des MVVM Patterns getestet. Ebenso wird die Unterstützung für Multithreading geprüft, indem Daten im Hintergrund geladen werden und dem Benutzer signalisiert wird, dass die UI momentan nicht bearbeitet werden kann. Nachdem die Daten geladen wurden, werden diese auf der Benutzeroberfläche angezeigt. Die Validierungen wurden über eine Third Party Bibliothek im Prototyp implementiert.

Darüber hinaus werden mithilfe des Prototyps die Schnittstellen von JavaFX zur Logik ermittelt. Die Best-Practices von JavaFX werden implementiert, um JavaFX Features bestmöglich nutzen zu können. Ziel ist es aufzuzeigen, wie die Schnittstellen des ePEP UI Frameworks verändert werden müssten, um mit JavaFX kompatibel zu sein.

Während der Implementierung wird nicht im Detail auf die Architekturen von der ePEP-Anwendung geachtet. Es werden lediglich vereinfachte, wesentliche Strukturen nachgebildet.

5.2 VORSTELLUNG DER UI

Die UI im Prototyp ist an die UI von ePEP-Anwendung angelehnt. Dabei wurden erneut nur die wesentliche UI-Architektur, mit Navigation, Master Table und Details nachgebaut (Vgl. Kapitel 3.1.1).

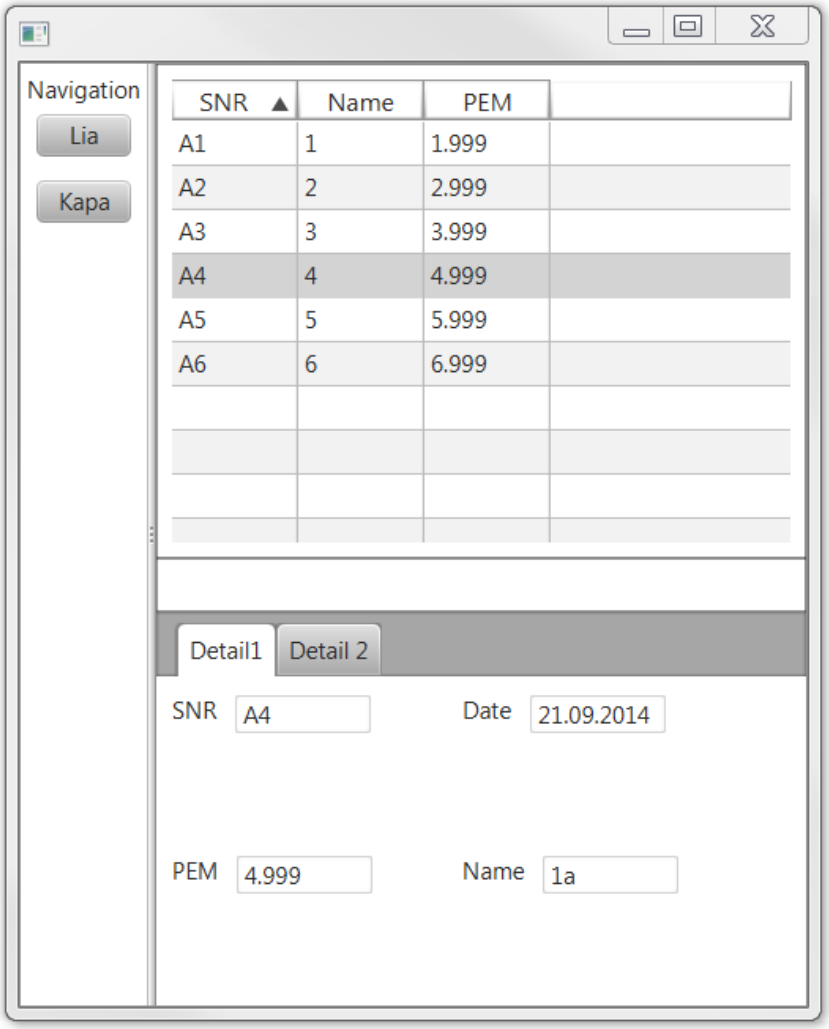

Abbildung 14: Screenshot Prototyp

Wie der Screenshot des Prototyps zeigt, kann der Benutzer in der Navigation zwischen zwei Prozessgebieten wählen, bei denen verschiedene Datensätze in die Master Table geladen werden. Nach Auswahl eines Datensatzes werden die Detail Tabs mit Daten befüllt.

5.3 VORSTELLUNG ARCHITEKTUR

Ein Vorteil von JavaFX ist, dass die Anwendungsoberfläche und der Java Code strikt voneinander getrennt sind, da in JavaFX Oberflächen deklarativ programmiert werden. Das bedeutet, dass die Benutzerschnittstelle nicht mit der Logik im Java File, sondern ausgelagert von der Logik, in einem FXML Dokument, programmiert wird. *Da FXML direkt auf Java abbildet, kann der Benutzer sehr leicht erkennen, welche Elemente und Attribute erlaubt sind. Dadurch ist eine schnelle Implementierung gewährleistet.* (Fedortsova 2014)

```
1  <AnchorPane fx:controller="view.NavigationController">
2    <children>
3      <Button fx:id="kapaBtn" layoutX="10.0" layoutY="56.0"
4  prefWidth="60.0" onAction="#onKapaBtnClicked" text="Kapa"/>
5    </children>
6  </AnchorPane>
```

Der abgebildete Codeausschnitt ist ein Auszug aus der Navigation.fxml Datei. In der ersten Zeile wird der Container mit dem dazugehörigen Controller definiert. Das fx:controller Attribut enthält den relativen Pfad zu der Controller-Klasse. Zwischen der zweiten und der fünften Zeile werden alle Elemente definiert, die zu dem Container gehören. In der dritten und vierten Zeile wird ein Button definiert, welcher den Namen des Java Objekts in dem fx:id Attribut enthält. Zusätzlich erhält der Button das onAction Attribut, welches den Namen der Methode enthält, die bei dem onAction Event ausgeführt wird. In der FXML-Datei werden auch Attribute wie Breite und Höhe definiert.

Folgender Auszug stellt den Quellcode des dazugehörigen Controllers dar.

```
1  public class NavigationController implements Initializable{
2      @FXML
3      Button kapaBtn;
4
5      @Override
6      public void initialize(URL location, ResourceBundle resources){}
7
8      public void onKapaBtnClicked(ActionEvent e){}
9  }
```

In der ersten Zeile wird die Klasse deklariert. Spezifisch für JavaFX ist hierbei, dass die Klasse das javafx.fxml.Initializable Interface implementieren muss. Objekte, die eine FXML-Komponente repräsentieren, müssen eine Annotation enthalten, siehe Zeile zwei. Der Name des Objekts kapaBtn findet sich in der obigen FXML-Datei wieder, genau wie der Methodenname onKapaBtnClicked, welche bei einem Event vom Button ausgelöst wird.

Durch die Trennung von Logik und UI ist es bspw. möglich, Controller Klassen generieren zu lassen und damit auch Daten automatisch zu binden. Die Beziehung zwischen Controller und View ist in Abbildung 15 dargestellt.

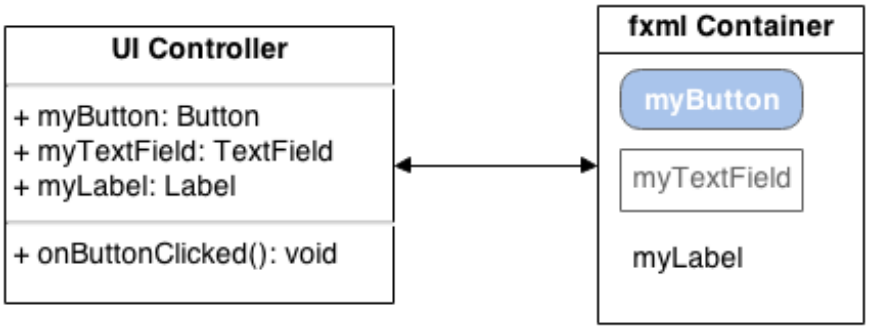

Abbildung 15: Beziehung zwischen Java UI Controller und UI-Komponenten

Die Abbildung zeigt, dass die Zuordnung von Controller und FXML Container immer eins zu eins ist. Der UI Container kann dann beliebig viele UI-Komponenten enthalten, wobei diese über Dependency Injection[17] an die Objekte in der Controller-Klasse gebunden werden. Dadurch können Events, Eigenschaften wie enabled/disabled usw. direkt auf das Objekt abgebildet werden. Die Kommunikation zwischen UI-Komponenten und Eigenschaften in der Controller-Klasse findet hier automatisch über das mitgelieferte Data Binding statt.

In JavaFX kann die UI mit einem Tool namens Scene Builder per Drag and Drop zusammengebaut werden. Dadurch hat der Entwickler sofortiges Feedback über das Aussehen der UI. Weiterhin ist es mit dem Scene Builder möglich, Werte der Eigenschaften zu setzen und Events mit Methoden des Controllers zu verbinden, ohne Sourcecode von Hand zu programmieren. Dies reduziert den Aufwand für die UI Programmierung, die Fehlerquellen sowie den Code, der in der Logik des Controllers implementiert ist.

Das MVVM Pattern ersetzt das in ePEP genutzte Presentation Model Pattern. Da für JavaFX nicht das Original MVVM Pattern verwendet werden kann, benutzt der Prototyp eine Abwandlung.

[17] *Dependency Injection ist eine Technik, um eine lose Kopplung bei Quellcode zu erreichen.* (Seemann 2012, Seite 5)

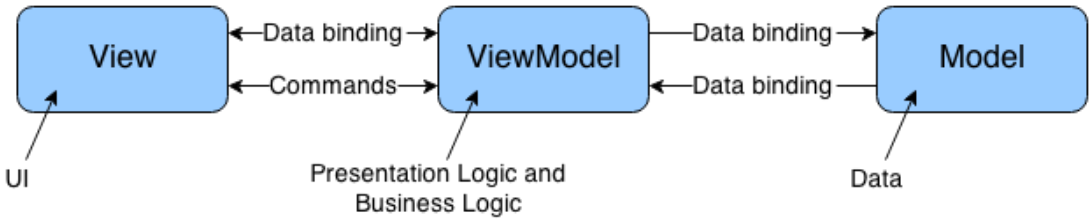

Abbildung 16: MVVM Pattern für JavaFX, angelehnt an (Brumfield, et al. 2011, Kapitel 5)

Die Controller-Klasse besitzt sowohl Objekte, welche die UI-Komponenten der View repräsentieren, als auch Referenzen auf die JavaBean-Klassen. JavaFX übernimmt die Bindung der UI-Objekte des Controllers an die UI-Komponenten der FXML Datei. Durch das Data Binding Framework von JavaFX werden die Komponenten im Controller an die JavaBeans gebunden. Bei dem Binding zwischen View und ViewModel kommt ein bidirektionales Binding zum Einsatz, sodass keine Inkonsistenzen auftreten. Die Business Logik wird, anders als im Original, im ViewModel implementiert.

Abbildung 17 zeigt, wie das MVVM Pattern in Verbindung mit dem Prototyp implementiert ist. Das ePEP UI Framework kommt nicht zum Einsatz, da dies den Prototyp unnötig komplizieren würde. Stattdessen repräsentieren JavaBeans die Schnittstellen zum ePEP UI Framework.

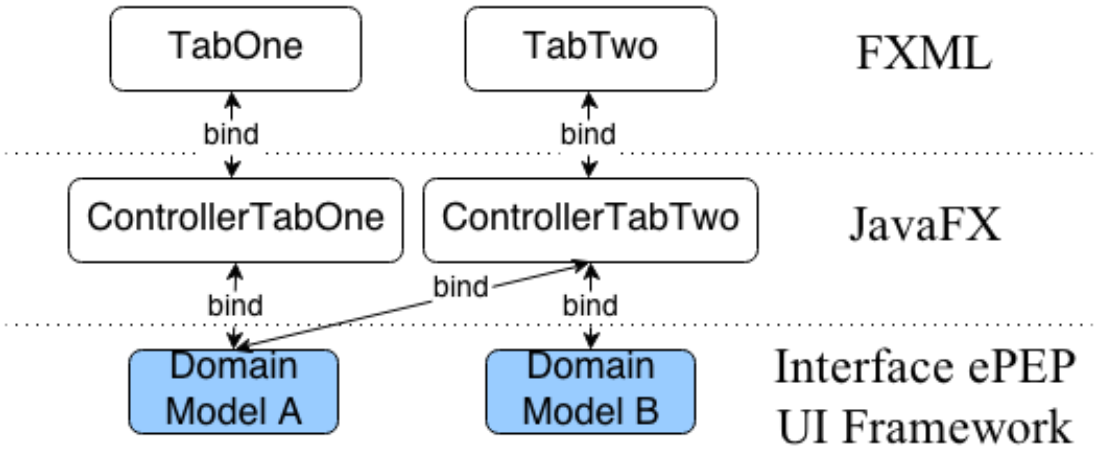

Abbildung 17: Prototyp UI Framework

Die Abbildung zeigt das MVVM Pattern am Beispiel der zwei Detail Tabs. TabOne und TabTwo sind die FXML Dateien, welche die View repräsentieren. Der Controller repräsentiert das ViewModel. Für das Data Binding zwischen View und ViewModel müssen in der View lediglich die Namen der Java Objekte und des Controllers angegeben werden. Die DomainModels A und B sind die Models. Der Controller bindet das Model an seine UI-Komponenten.

Hierbei ergeben sich die ersten Vorteile, da das Data Binding zwischen View und ViewModel bzw. zwischen ViewModel und Model, durch JavaFX und nicht mehr über das externe JGoodies Binding Framework implementiert werden muss. Dadurch minimiert sich der Code und wird weniger komplex.

Diese Struktur findet sich in allen Views wieder. Die Parallelen zur aktuellen Struktur von ePEP erleichtern einen Umstieg auf die neue Technologie, da viele vorhandene Strukturen wiederverwendet werden können.

5.4 VALIDIERUNGEN

Im Prototyp ist eine Form von Validierung implementiert, die nicht auf mehreren Threads arbeitet. Es kommt ein Framework aus dem Internet[18] zum Einsatz. Durch dieses Framework kann eine Validierung durchgeführt werden. Je nach Ergebnis werden CSS3-Styles der Komponenten gesetzt, die dem Benutzer eine visuelle Rückmeldung über das Ergebnis der Validierung geben.

Durch das Data Binding und die Properties bietet JavaFX eine Schnittstelle, die geeignet ist, um eine Validierung durchzuführen. Die Validatoren können ausgeführt werden, sobald sich die TextProperty eines Textfeldes oder die FocusProperty eines Buttons, geändert hat. Dabei behält der Nutzer die Freiheit die Fehleranzeige nach seinen Wünschen zu gestalten. Allerdings erhält der Benutzer keine Unterstützung bei der Ausführung von Validierungen in verschiedenen Threads.

[18] http://dev-fjord.blogspot.de/2012/11/javafx-field-validation.html

```
1  ITypeValidator validator = new ITypeValidator(){
2      @Override
3      public State validate(Object typeToValidate){
4          // Whatever validation code is required
5          if (isValid(typeToValidate)){
6              return State.VALID;
7          } else {
8              return State.ERROR;
9          }
10     }
11 };
12
13 MyValidator idValidator = new MyValidator
14 (textField.textProperty(),validator, State.VALID);
15
16 idValidator.addStyleTargets(errorLabel, textField);
17
18 ChangeListener<State> listener = new ChangeListener<State>() {
19     @Override
20     public void changed(ObservableValue observable, State oldValue,
21 State newValue) {
22         switch (newValue) {
23             case ERROR:
24                 //Errortext setzen
25             break;
26         }
27     }
28 }
29
30 idValidator.stateProperty().addListener(listener);
```

Es muss das ITypeValidator Interface implementiert werden (Zeile 1 bis 11). Diese Implementierung führt die Validierung durch und gibt den Status der Validierung zurück. Das MyValidator Objekt wird erzeugt und bekommt drei Parameter: Eine Property bei dessen Änderung er benachrichtigt wird, den ITypeValidator der die Validierung durchführt und einen initialen Zustand der Validierung (Zeile 13 und 14). Der Konstruktor der MyValidator-Klasse bindet einen Listener an die übergebene Property und führt in diesem Listener den mitgegebenen ITypeValidator aus. Nach dessen Ausführung setzt die setState-Methode das Ergebnis der Validierung, wobei alle StyleTargets den zugehörigen CSS3-Style in dieser Methode annehmen. Über die addStyleTarget-Methode werden Komponenten hinzugefügt (Zeile 16). Um dem Benutzer eine Rückmeldung zu geben, ändern die Komponenten nach einer Validierung ihren Style. Ein Listener, der ausgeführt wird, sobald sich der Status einer

Validierung geändert hat, wird implementiert (Zeile 18 bis 29). Hier kann auf das Ergebnis der Validierung reagiert werden und es kann z. B. ein Fehlertext von einem Errorlabel gesetzt werden. Am Schluss wird der Listener an den Status des Validators gebunden (Zeile 31).

5.5 JAVAFX SERVICE

Der JavaFX Service bietet dem Entwickler eine bessere Unterstützung beim Managen von Multithread-Code. Der Service wird in einem separaten Thread ausgeführt und kann voll kontrolliert werden (Vgl. Kapitel 4.2.3). In dem Prototyp lädt ein Service die Daten, die in der Master Table und in den Details angezeigt werden. Dabei soll das Laden aus einer Datenbank simuliert werden, um eine realistische Ladezeit, wie in der Anwendung ePEP, zu testen. Durch die Ausführung des Services in einem eigenen Thread wird der UI Thread nicht blockiert. Der Hauptthread soll die UI für den Benutzer sperren und diesem einen Ladevorgang anzeigen. Dies signalisiert dem Benutzer, dass eine weitere Bearbeitung der Daten momentan nicht möglich ist. Die einzelnen Detail Tabs werden parallel geladen. Zu beachten ist, dass sich der Ladevorgang erst beendet, nachdem die Daten für alle Tabs geladen sind. Dabei wird das in Kapitel 4.2.3 vorgestellte Service Konstrukt von JavaFX verwendet.

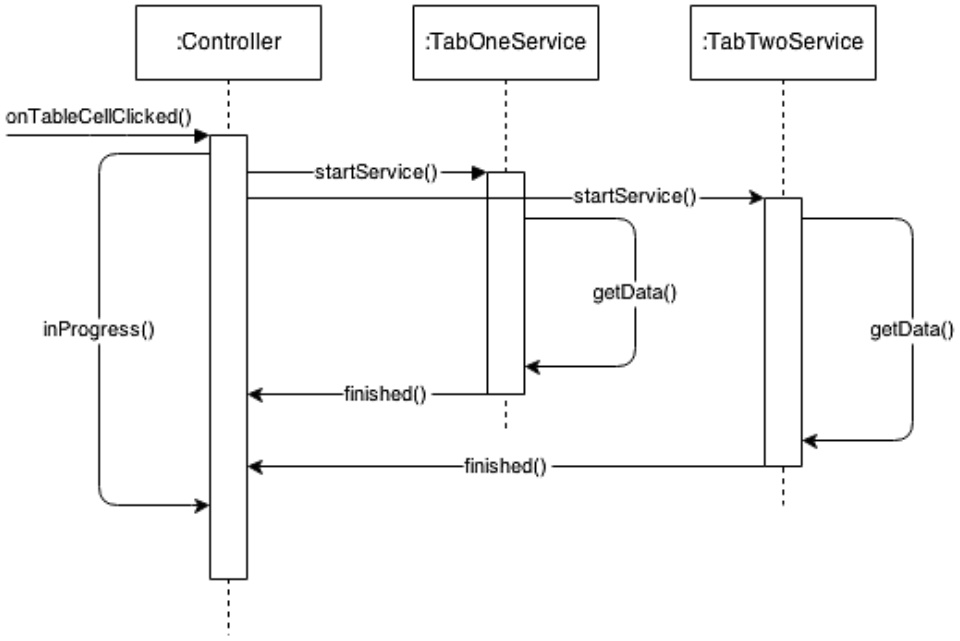

Abbildung 18: Sequenzdiagramm zum Ablauf eines Service Aufrufes

Das Diagramm beschreibt den Ablauf, um zwei Detail Tabs mit Daten zu befüllen. Nachdem ein Benutzer auf eine Zelle der Tabelle klickt und somit einen Datensatz auswählt, startet der Ablauf. Hierbei startet der Controller zwei unabhängige Services, die den Controller benachrichtigen, sobald sie fertig sind. Die Services sind für die Datengenerierung zuständig. Die generierten Daten werden dann an der Oberfläche angezeigt. Währenddessen setzt der Controller den Zustand „inProgress“ im Model auf true, sodass der Benutzer eine Rückmeldung bekommt.

Ein Weg die Daten anzuzeigen ist, dass die Services die ViewModels direkt manipulieren. Jedoch muss darauf geachtet werden, dass dies mit dem JavaFX Befehl Platform.runLater() ausgeführt wird. Dadurch werden die Änderungen der Daten nur in dem Application Thread durchgeführt. Wenn der Entwickler dies nicht berücksichtigt, wird eine Exception ausgelöst, da Änderungen an der UI nur im Application Thread durchgeführt werden dürfen (Vgl. Kapitel 4.2.3).

Der andere Weg ist, dass die Services die manipulierten Daten an den Controller zurückgeben und dieser die Aktualisierung der UI durchführt. Dadurch würde der Controller jedoch eine weitere Aufgabe bekommen. *Laut dem Single Responseability Pattern sollte eine Klasse aber nur eine Verantwortlichkeit besitzen.* (Martin 2009, Seite 138)

Im entwickelten Prototyp kommt beim Laden der Detail Tabs die erste Methode und beim Laden der Master Table die zweite Methode zum Einsatz. Vorteil der zweiten Methode ist, dass die Änderungen der UI auf Klassen konzentriert sind, die im Hauptthread laufen. Das hilft Fehler zu vermeiden und zu finden, da Daten nicht threadübergreifend geändert werden. Des Weiteren muss der Entwickler nicht berücksichtigen, dass Änderungen der UI nur im Hauptthread gemacht werden dürfen.

5.6 DATA BINDING

Am Beispiel der Master Table wird gezeigt, wie die beim MVVM Pattern beteiligten Klassen aufgebaut sind. Darüber hinaus ist abgebildet, wie diese einen Service benutzen, der die anzuzeigenden Daten im Hintergrund ändern soll (Abbildung 19).

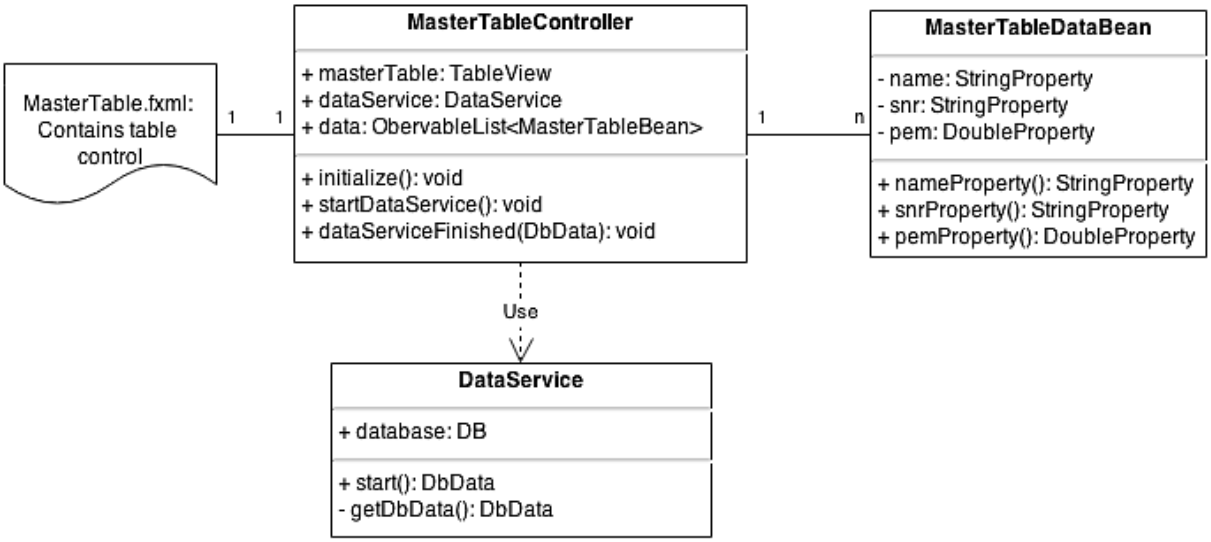

Abbildung 19: Klassendiagramm Master Table

Das Klassendiagramm stellt die Struktur dar, die gebraucht wird, um die Master Table mit Daten zu füllen. Die in der MasterTable.fxml Datei beschriebenen Komponenten werden durch JavaFX mit den UI-Komponenten in der MasterTableController-Klasse verbunden.

Der Controller besitzt außerdem eine ObservableList der Daten, die in der Tabelle angezeigt werden sollen und bindet diese Liste an die Tabellen Komponente der UI. Durch die ObservableList von JavaFX ist kein extra JavaBean notwendig, der den Inhalt der Tabelle repräsentiert.

Außerdem besitzt der Controller einen Service, der in einem separaten Thread ausgeführt wird, sowie einen Datenbank Zugriff simuliert und Daten aus der Datenbank zurückgibt. Aufgabe des Controllers ist auch, die vom Service zurückgegebenen Daten in die Liste zu konvertieren, die an der UI angezeigt wird. Die Komponenten werden beim Initialisieren des Controllers aneinander gebunden.

Da u. a. das Data Binding von JavaFX genutzt werden soll, müssen die Schnittstellen des ePEP UI Frameworks geändert werden. Diese bauen auf dem JGoodies Binding Framework auf, welches nicht mehr genutzt werden soll.

5.7 IDENTIFIKATION DER ÄNDERUNGEN

Die Views, die noch nicht auf dem neuen ePEP UI Framework basieren, können nur mit sehr hohem Aufwand umgestellt werden. Zuerst müssten sie auf das ePEP UI Framework migriert werden (Vgl. Kapitel 3.1.2). Momentan wäre dies mit zu hohen Kosten verbunden, könnte aber ein langfristiges Ziel für das Projekt ePEP sein. Deswegen wird im Folgenden nur betrachtet, wie Komponenten, die bereits mit dem ePEP UI Framework implementiert sind, umgestellt werden können.

Die UI und ihre Logik besteht aus den Schichten Swing, JGoodies Binding und dem ePEP UI Framework (Vgl. Kapitel 3.1.2). Bei einem Wechsel der UI von Swing mit dem Presentation Model Pattern auf JavaFX mit dem MVVM Pattern, ergeben sich Änderungen in der Architektur. FXML ersetzt die Swing UI. Die Controller fassen die Presentation Models von dem JGoodies Binding Framework und die Compound Models zusammen. Die Controller übernehmen die Rolle des ViewModels in dem MVVP Pattern (Vgl. Kapitel 5.3). Das ePEP UI Framework soll weiterhin genutzt werden. Dadurch kann die UI Logik, wie die Validierung, weiterhin verwendet werden. Daraus ergeben sich folgende Änderungen für die Deskriptoren:

1. Der Domain Model Descriptor muss die Domain Models so generieren, dass sie den JavaFXBean Konventionen entsprechen. Dazu müssen die Properties implementiert werden, die zu den Eigenschaften gehören. Weiterhin müssen die getter und setter so angepasst werden, dass die Properties genutzt werden.
2. Der Compound Descriptor muss statt der Compound Models und der Presentation Models nun nur noch die Controller generieren. Er verbindet seine UI-Komponenten mit den dazugehörigen Domain Model Properties.
3. Die Views müssen in JavaFX nachprogrammiert werden. Danach muss der Entwickler den zugehörigen Controller an den UI Container binden.

Dadurch entstehen Änderungen in der Architektur (Abbildung 20).

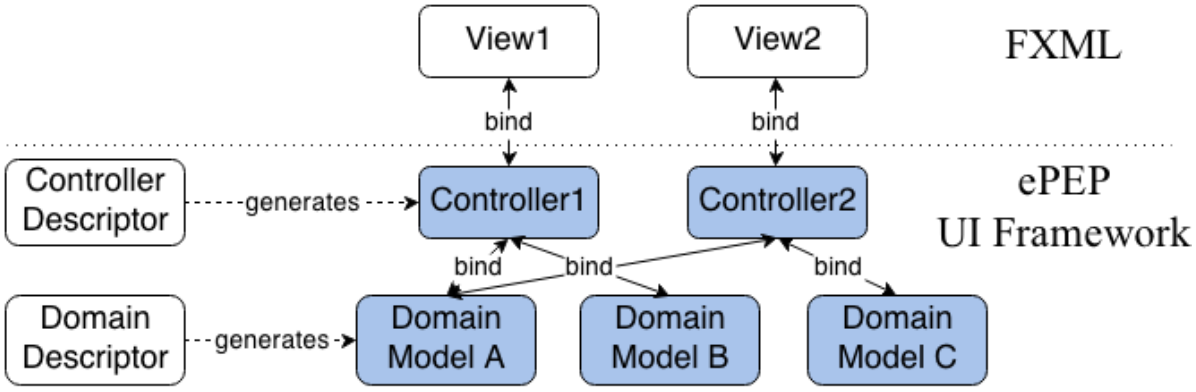

Abbildung 20: ePEP UI Framework mit JavaFX

Durch die neue Architektur entfällt die Schicht der Compound Descriptoren. Der Controller Descriptor generiert die Controller. In dem vorherigen Framework hat der Compound Descriptor Änderungen an dem Presentation Model an die richtigen Domain Models weitergeleitet. Dies übernimmt jetzt der Controller selbst, da es durch das Data Binding nicht mehr nötig ist, Änderungen explizit weiterzuleiten, nachdem die Daten gebunden sind. Dadurch wird der generierte Code minimiert und das MVVM Pattern so verwendet, wie es in Kapitel 5.3 beschrieben ist.

5.8 WEB-APPLIKATION

Mithilfe des von JavaFX mitgelieferten Deployment Tools können Applikationen deployed werden. Das Tool benutzt Ant[19], um ein executeable JAR-File und ein JNLP-File zu erstellen. Dadurch ist es ohne Quellcodeänderungen möglich die Applikation als Stand-Alone, als Web Start[20] oder im Browser eingebettet, auszuführen.

Der Vorteil ist, dass Java Entwickler nun auch Webapplikationen mit reinem Java entwickeln können und keine andere Programmiersprache lernen müssen. Ein Nachteil ist, dass Java auf dem Client installiert sein muss. Das ist für ePEP wenig hinderlich, da Java auf allen Clients installiert ist.

---

[19] Ant ist ein von Apache entwickeltes Tool, welches aus Quellcode ausführbare Programme baut. http://ant.apache.org/

[20] *Web Start ermöglicht unabhängige Java Applikationen mit einem klick über das Netzwerk zu deployen.* (Oracle)

6 SCHLUSSFOLGERUNG

In diesem Kapitel werden die beiden Frameworks Swing und JavaFX zusammenfassend bewertet. Es werden die in der Arbeit erläuterten Vor- und Nachteile noch einmal aufgelistet. Darüber hinaus wird eine Handlungsempfehlung für das Projekt ePEP erstellt und die Risiken einer Umstellung beschrieben.

6.1 BEWERTUNG SWING / JAVA FX

Swing war jahrelang, zusammen mit SWT, *der* Standard für Fat-/Rich-Client Applikationen, die mit Java entwickelt wurden. Swing ist ein Framework, das mittlerweile in vielen Anwendungen etabliert ist. Es existiert eine große Community, gute Dokumentation und zahlreiche Beispiele. Es gibt viele Third Party Bibliotheken, die zusätzliche Funktionalität für Swing bereitstellen. Diese bringen jedoch gleichzeitig eine Abhängigkeit mit sich, die nicht immer von Vorteil ist (Vgl. Kapitel 2.1.2). Bei Swing existiert kaum, bzw. gar keine Unterstützung für das Data Binding, Threading und für die Form Validierung (Vgl. Kapitel 3.1.4).

JavaFX wurde von Oracle als offizieller Nachfolger von Swing benannt. In dem neuen Release von der JSE8, am 18. März 2014, ist auch die neue Version von JavaFX enthalten. Somit wird JavaFX nun zusammen mit der JSE als Standard UI Framework ausgeliefert. Das bestärkt das Verlangen von Oracle, JavaFX auf dem Markt zu etablieren.

Es bildet sich langsam eine Community, welche die Entwicklung von JavaFX Applikationen unterstützt. Auf der Homepage von Oracle stehen mittlerweile zahlreiche Bibliotheken zur Verfügung, die zusätzliche Funktionalitäten für JavaFX bieten (Oracle). Diese sind jedoch für die Benutzung oft noch nicht ausgereift oder befinden sich wie die JideFX Bibliothek (Qiao 2013) in Beta Testphasen. Auch ist es für die Entwickler schwieriger, Beispiele und Hilfe zu finden, wenn Probleme bei der Entwicklung entstehen oder ein Entwickler JavaFX erlernt.

Mit JavaFX ist es möglich RIAs zu entwickeln, die Potential haben, indem sie Vorteile von Fat-/Rich- und Thin-Clients vereinen (Vgl. Kapitel 2.2.3). Durch das Data binding werden inkonsistente Daten vermieden und zugleich der Sourcecode minimiert (Vgl. Kapitel 3.2.1).

Durch die deskriptive UI Programmierung und die Möglichkeit CSS3 einzusetzen, können UI und Logik leichter getrennt und zentralisiert werden. Zusätzlich erleichtert der Scene Builder die Entwicklung.

Die Form Validierung wird nur für einfache Fälle unterstützt. Durch das Data Binding existieren jedoch Schnittstellen, die es erleichtern, die Form Validierung umzusetzen.

6.2 HANDLUNGSEMPFEHLUNG

Durch die saubere Kapselung in den Views, die mit dem ePEP UI Framework umgesetzt wurden, ist es möglich, die UI mit vergleichsweise wenig Aufwand auf JavaFX umzustellen.

Basierend auf den in der Arbeit ermittelten Fakten, wird empfohlen, das Projekt ePEP auf das JavaFX UI Framework umzustellen. Zuerst sollten alle Views umgestellt werden, die bereits mit dem ePEP UI Framework arbeiten. Als nächsten Schritt sollten die Views, die nicht mit dem ePEP UI Framework arbeiten, auf dieses umgestellt und dann auch auf JavaFX migriert werden.

Dadurch kann Swing in absehbarer Zeit abgeschafft und ePEP, von einem Rich-Client zu einer RIA, umgebaut werden.

7 SCHLUSS

In diesem Kapitel wird ein Fazit der Arbeit gezogen und ein Ausblick gegeben, der langfristige Ziele für das Projekt ePEP beschreibt.

7.1 FAZIT

In der Einleitung der Arbeit wurde die Frage gestellt, warum ein etabliertes Framework wie Swing ausgetauscht werden sollte.

Die Antwort darauf ist, dass Swing nicht mehr das offizielle Java UI Framework ist. Dadurch werden Projekte veralten, wenn sie nicht auf ein neues UI Framework umsteigen. Da allerdings viele Projekte vor dieser Herausforderung stehen, wird der Umstieg weiterhin von Oracle und der Community gefördert.

Welchen Aufwand bringt eine Umstellung des UI Frameworks mit sich?

Durch den Status Quo des Projekts ergab sich, dass die Views in ePEP auf zwei verschiedenen Architekturen aufbauen. In einem Teil der Views sind Logik und UI nicht voneinander getrennt. Bevor hier das UI Framework ausgetauscht werden kann, sollte mithilfe eines Refactorings die Logik von der UI getrennt werden. Letzteres wurde im Verlauf der Arbeit nicht weiter betrachtet, da bereits einschlägige Literatur zu diesem Thema existiert.

In dem anderen Teil ist eine Trennung von UI und Logik gegeben, wodurch der Aufwand, das UI Framework auszutauschen, minimiert wird. Die Generatoren des ePEP UI Frameworks müssen angepasst und die Views von Hand nachgebaut werden.

Allgemein kann gesagt werden, dass Komponenten eines Systems immer bestmöglich voneinander getrennt und nur über klar definierte Schnittstellen angesprochen werden sollten. Dadurch wird ein Austausch einer Komponente vereinfacht.

Aber welchen Mehrwert bringt das neue Framework für das Projekt ePEP?

Was den Mehrwert von JavaFX betrifft, so konnte durch den Prototyp gezeigt werden, dass JavaFX mit dem integrierten Data Binding und dem JavaFX Service einen Teil der Probleme von Swing löst. Die Form Validierung kann mithilfe des Data Bindings sauber implementiert werden. Die Form Validierung in verschiedenen Threads kann mit den JavaFX Services selbst gelöst werden.

Aus der Implementierung mit JavaFX resultiert eine vereinfachte UI Architektur, welche das Model-View-ViewModel Pattern benutzt. Die Komponenten des Patterns kommen nun wie von den Erfindern gedacht zum Einsatz, was das Verständnis der Architektur erleichtert.

Weiterhin wurde deutlich, dass JavaFX Applikationen als Rich-Client und RIA, bspw. als eine Webapplikation im Browser, ausgeführt werden können, ohne die Implementierung anzupassen.

Ich persönlich halte dieses Thema allgemein für sehr interessant und denke, dass es viel Potential für die Forschung bietet. Es werden immer neue Frameworks entwickelt werden, die dem Benutzer bessere Unterstützung bieten. Daher sollte der Aufwand auf ein neues Framework umzustellen minimiert werden. Eine Fragestellung in diesem Zusammenhang ist, ob der Aufwand nur minimiert werden kann, indem der Programmierer Logik und UI getrennt von einander implementiert? Oder können hier auch Automatismen oder Techniken gefunden werden, die dem Programmierer diese Arbeit abnehmen? Dadurch könnten Projekte immer mit der besten Unterstützung arbeiten und so die Softwareentwicklung effizienter gestaltet werden.

7.2 AUSBLICK

Durch die Einführung des ePEP UI Frameworks wurde in 49 der 196 Views Logik und View sauber von einander getrennt. Die Views, die diese Trennung nicht haben, sollten auf das ePEP UI Framework und dann auf JavaFX umgestellt werden.

Ein langfristiges Ziel sollte sein, ePEP als Webapplikation auszuliefern, da so die Vorteile einer Rich Internet Application genutzt werden können.

Anzumerken ist, dass die Komplexität des Prototyps in keinem Verhältnis zu Applikationen mit voller Funktionalität wie ePEP steht. Bei einem Projekt mit der Größe von ePEP

entstehen andere Probleme als bei Applikationen, die nur einen Bruchteil des Quellcodes umfassen. Darunter können z. B. Geschwindigkeitsprobleme der UI oder das Managen von mehreren Threads fallen. Ebenfalls existieren bei großen Projekten Altlasten, welche die Arbeit mit neuen Frameworks erschweren können und bei dem Austausch eines UI Frameworks beachtet werden müssen.

Oracle hat mit der Entwicklung von JavaFX einen Nachfolger für Swing präsentiert, der nicht nur in der Liga der Rich-Clients, sondern auch bei den RIAs mitspielen kann. Durch die Integration von JavaFX in die JSE treibt Oracle die Verbreitung von JavaFX voran. Da Java Entwickler auf ihre gewohnten Fähigkeiten bei der Entwicklung von Webapplikationen zurückgreifen können, hat JavaFX eine gute Chance, sich in der Community zu etablieren.

8 ANHANG

8.1 QUELLCODE

Der Quellcode befindet sich auf einem USB-Stick, der der Bachelorarbeit beigelegt ist.

8.2 PROTOTYP SCREENSHOT

Hier werden Screenshots gezeigt, die das Verhalten vom Prototyp zeigen.

8.2.1 VALIDIERUNG

Die Screenshots zeigen Fehler in der Validierung.

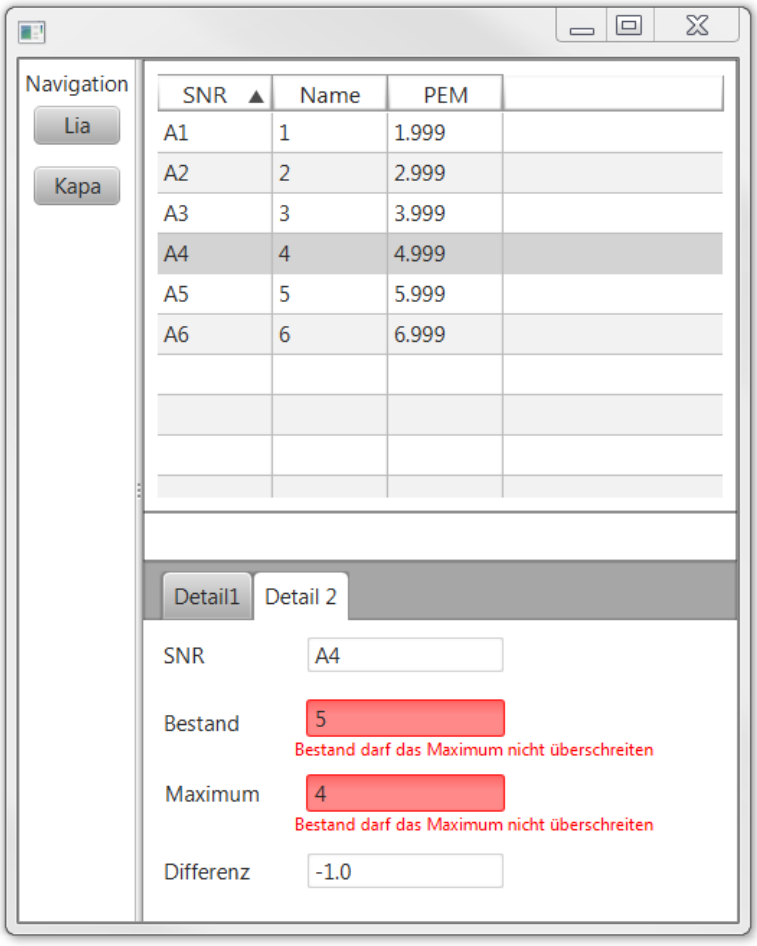

Abbildung 21: Fehler in der Validierung: Der Bestand darf nicht größer als das Maximum sein.

Die Abbildung zeigt eine Business Rule Validierung. Da die Felder Bestand und Maximum zusammenhängen, werden beide Felder als fehlerhaft markiert. Der Benutzer hat zwei Möglichkeiten um die Fehler zu beheben. Er kann den Bestand heruntersetzen oder das Maximum erhöhen.

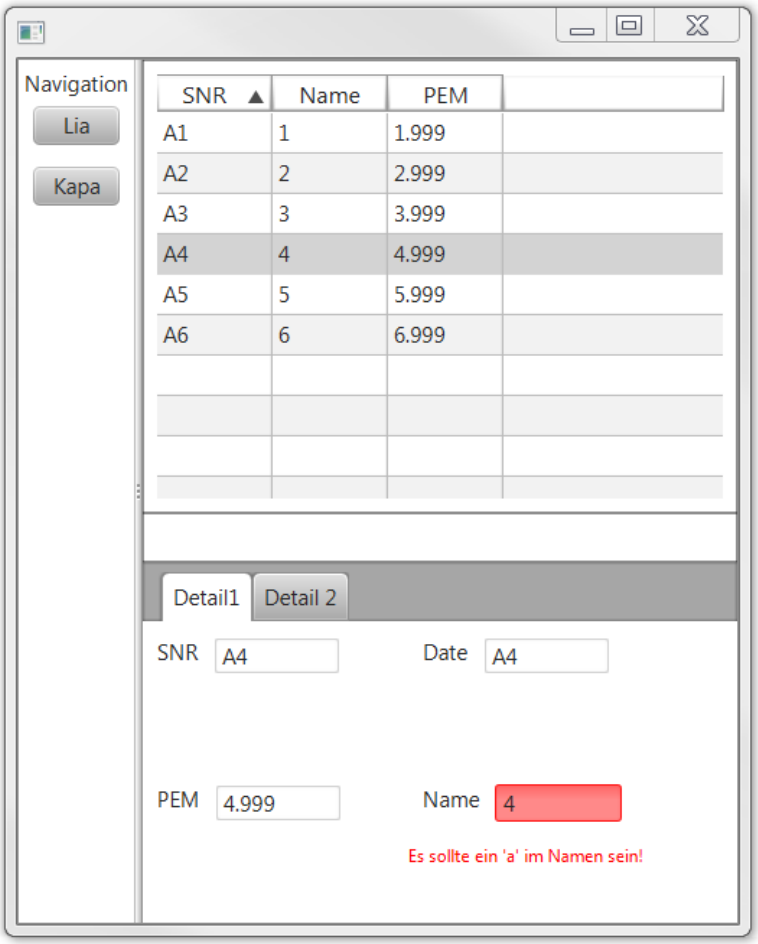

Abbildung 22: Fehler in der Validierung: Der Name muss ein 'a' enthalten

Die Abbildung zeigt eine einfache Validierung, die auf dem Inhalt von einem unabhängigen Feld durchgeführt wird. Hier muss der Name ein „a“ enthalten, um gültig zu sein.

8.2.2 Lade Operation

Während Daten geladen werden ist die UI nicht für Änderungen zugänglich und zeigt dem Benutzer an, dass sie sich in dem „inProgress“ Zustand befindet.

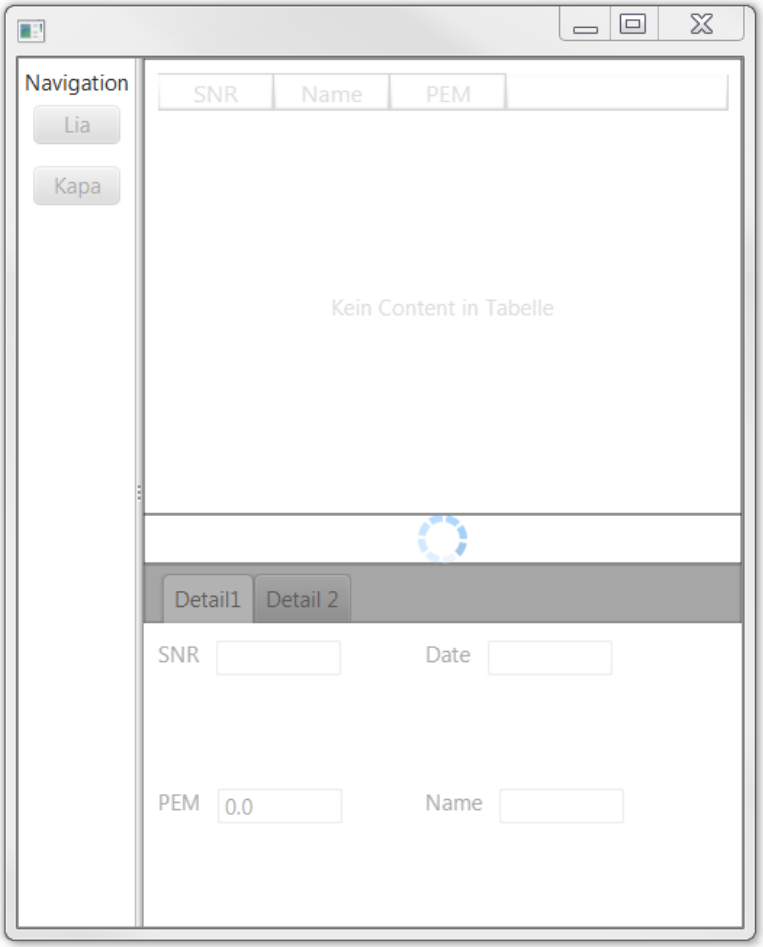

Abbildung 23: Prototyp während einer Ladeoperation

Während Daten geladen werden steht die UI nicht für eine Bearbeitung zur Verfügung. Dies wird dem Benutzer durch ein „in Progress“-Control, und die Sperrung der UI signalisiert.

9 LITERATURVERZEICHNIS

Anderson, Gail, und Paul Anderson. *Essential JavaFX™.* Prentice Hall, 2009.

Balzert, Helmut, und Heide Balzert. *Basiskonzepte und Requirements-Engineering.* Bd. 1. Heidelberg: Spektrum, Akad. Verl., 2009.

Bibliographisches Institut GmbH. *Duden.* 2013. http://www.duden.de/rechtschreibung/ (Zugriff am 20. 02 2014).

Brumfield, Bob, Geoff Cox, David Hill, Brian Noyes, Michael Puleio, und Karl Shifflett. *Developer's Guide to Microsoft® Prism 4: Building Modular MVVM Applications with Windows® Presentation Foundation and Microsoft Silverlight®.* Microsoft Press, 2011.

Champ, Sean. *JFace.* Herausgeber: The Eclipse Foundation. 06. 08 2010. http://wiki.eclipse.org/JFace (Zugriff am 17. 03 2014).

Dahm, Markus. *Grundlagen der Mensch-Computer-Interaktion.* München: Pearson Studium, 2006.

DiMarzio, J. F. *JavaFX : a beginner's guide.* New York, NY: Oracle Press, 2011.

Durville, Thomas, Thorsten Memmel, Nicolas Büring, und Olivier Lalive d'Epinay. „Funktionalität, Usability und schönes Design–Herausforderungen bei der Wahl der Client-Technologie." Whitepaper, 2010.

Essling, Dipl. Ing. Olaf. *Informatik I, Grundlagen der Programmierung, Definitionen und Begriffe.* 20. 09 2004. http://public.fh-wolfenbuettel.de/~essling/save/prog/info1_def.pdf (Zugriff am 20. 02 2014).

Fedortsova, Irina. *Mastering FXML.* Oracle. 01 2014. http://docs.oracle.com/javafx/2/fxml_get_started/why_use_fxml.htm#CHDCHIBE (Zugriff am 26. 02 2014).

Fowler, Martin. *Presentation Model.* 2004. http://www.martinfowler.com/eaaDev/PresentationModel.html (Zugriff am 14. 03 2014).

—. *Refactoring: improving the design of existing code.* Reading, Mass: Addison-Wesley, 1999.

Geihs, Kurt. *Client-Server-Systeme : Grundlagen und Architekturen.* 1. Bonn: Thomson, 1995.

Hammerschall, Ulrike. *Verteilte Systeme und Anwendungen.* Bd. 166. München: Pearson Studium, 2005.

Hatton, Tim. *SWT : a developer's notebook.* Sebastopol, CA: O'Reilly, 2005.

ITWissen. *ITWissen Das große Online-Lexikon für Informationstechnologie.* DATACOM Buchverlag GmbH. 12. 02 2014. http://www.itwissen.info/ (Zugriff am 12. 02 2015).

JavaFX Frequently Asked Questions. kein Datum. http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html#6 (Zugriff am 28. 03 2014).

Kemper, Alfons, und André Eickler. *Datenbanksysteme: Eine Einführung.* Oldenbourg Verlag, 2011.

Kwok, Yu-Kwong Ricky. *Peer-to-Peer Computing: Applications, Architecture, Protocols, and Challenges.* CRC Press, 2011.

Martin, Robert C. *Clean Code: A Handbook of Agile Software Craftsmanship.* Munich: Prentice Hall, 2009.

McAffer, Jeff, Jean-Michel Lemieux, und Chris Aniszczyk. *Eclipse rich client platform.* Addison-Wesley Professional, 2010.

Microsoft. *Windows Presentation Foundation.* 2014. http://msdn.microsoft.com/de-de/library/ms754130(v=vs.110).aspx (Zugriff am 17. 03 2014).

Nagel, Christian, Bill Evjen, Jay Glynn, Karli Watson, und Morgan Skinner. *Professional C# 4 and .Net 4.* Wrox, 2010.

Northover, Steve, und Mike Wilson. *SWT : the standard widget toolkit.* Boston: Addison-Wesley, 2004.

Noyes, Brian. *Data Binding with Windows Forms 2.0: Programming Smart Client Data Applications with .NET.* Addison-Wesley Professional, 2006.

Oracle. Oracle. 2014. http://docs.oracle.com/javafx/2/api/javafx/concurrent/Service.html (Zugriff am 25. 02 2014).

—. *Java Web Start Overview.* kein Datum. http://www.oracle.com/technetwork/java/javase/overview-137531.html (Zugriff am 28. 04 2014).

—. *JavaFX Third Party Tools and Utilities.* kein Datum. http://www.oracle.com/technetwork/java/javafx/community/3rd-party-1844355.html (Zugriff am 10. 05 2014).

—. *Oracle - JavaFX Architecture and Framework.* Herausgeber: Cindy Castillo. 04 2013. http://docs.oracle.com/javafx/2/architecture/jfxpub-architecture.htm (Zugriff am 04. 03 2014).

Oxford University Press. *Ocfort Dictionaries.* 2014. http://www.oxforddictionaries.com/definition/english/front-end (Zugriff am 20. 02 2014).

Partsch, Helmuth. „Requirements Engineering.“ Ulm, 2014.

Premkumar, Lawrence. *Beginning JavaFX.* Apress, 2010.

Qiao, David. *JideFX.* 06. 06 2013. http://www.jidesoft.com/blog/2013/06/06/jidefx-beta-release/ (Zugriff am 10. 05 2014).

Riehle, Dirk. *Framework Design: A Role Modeling Approach.* Ph.D. Thesis, SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, Zürich: ETH Zürich, 2000.

Rumpe, Prof. B. „TU Braunschweig.“ Herausgeber: Dr. Ina Schaefer. 2011. https://www.tu-braunschweig.de/Medien-DB/isf/sse/vl5_entwurf.pdf (Zugriff am 12. 03 2014).

Seemann, Mark. *Dependency injection in .NET.* Shelter Island, NY: Manning, 2012.

slarti. *JavaFX Field Validation.* 13. 11 2012. http://dev-fjord.blogspot.de/2012/11/javafx-field-validation.html (Zugriff am 01. 04 2014).

Stark, Thomas. *J2EE: Einstieg für Anspruchsvolle.* Pearson Deutschland GmbH, 2005.

Ullenboom, Christian. *Java ist auch eine Insel : das umfassende Handbuch.* 10. Bonn: Galileo Press, 2012.

UMSICHT, Fraunhofer. „Studie: Thin-Clients 2011 - Ökologische und Ökonomische Aspekte virtueller Desktops.“ Dipl.-Inform. (FH) Christian Knermann. 17. 01 2011. https://www.igel.com/fileadmin/user/upload/documents/PDF_files/White_Paper_DE/thinclients2011-de-2.pdf (Zugriff am 20. 02 2014).

Walrath, Kathy. *The JFC Swing Tutorial: A Guide to Constructing GUIs.* Bd. II. Boston, MA: Addison-Wesley Professional, 2004.

Weaver, James L., Weiqi Gao, Stephen Chin, Dean Iverson, und Johan Vos. *Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology.* Apress, 2012.