ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Umsetzung eines Editors zur Erstellung von Auswertungsregeln elektronischer Fragebögen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Lars Miltkau
lars.miltkau@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Johannes Schobel

2016

Fassung 16. Februar 2016

Satz: PDF-LaTeX 2ε

Kurzfassung

Heute ist es bei psychologischen Studien nach wie vor üblich, papierbasierte Fragebögen zur Datenerhebung einzusetzen. Dies kann vor allem bei einer großen Anzahl an auszufüllenden Fragebögen schnell zu Problemen führen. So ist unter anderem der Aufwand für die Auswertung und Analyse der ausgefüllten Fragebögen sehr hoch, da diese beispielsweise von Hand digitalisiert werden müssen. Als Alternative etablieren sich immer mehr digitale Lösungen. Im Gegensatz zu den herkömmlichen papierbasierten Fragebögen, kann die Auswertung und Analyse bei elektronischen Fragebögen hochgradig automatisiert werden. Damit die Auswertung und Analyse der Fragebögen auch von Personen ohne Programmierkenntnisse durchgeführt werden kann, werden zuvor erstellte Auswertungsregeln verwendet.
Das Ziel dieser Arbeit ist es, einen bestehenden Editor weiterzuentwickeln, um eine leichtere Erstellung und Bearbeitung der Auswertungsregeln zu ermöglichen. Dazu werden Anforderungen festgelegt, mit welchen dann ein neues, auf Bäumen basierendes, Konzept entworfen und umgesetzt wird. Darüber hinaus wird die Bedienbarkeit und Benutzerfreundlichkeit des Editors verbessert.

Danksagung

Zuerst möchte ich mich beim Institut für Datenbanken und Informationssysteme der Universität Ulm bedanken, bei denen ich diese Abschlussarbeit geschrieben habe. Ein besonderer Dank gebührt Johannes Schobel, der meine Arbeit engagiert betreut hat, immer als Ansprechpartner zur Verfügung stand und mich ausgiebig unterstützt hat. Des Weiteren gilt mein Dank meinen Kommilitonen, die mir mit ihrer Motivation und Hilfsbereitschaft während des Studiums zur Seite standen. Abschließend möchte ich meiner Familie danken, die mir mein Studium durch ihre Unterstützung ermöglicht haben.

Inhaltsverzeichnis

1

Einleitung

In der heutigen Zeit werden in vielen Bereichen des Lebens, wie zum Beispiel bei psychologischen Studien oder bei medizinischen Umfragen, Fragebögen zur Datenerhebung eingesetzt. Dabei ist es üblich, diese in Papierform auszuteilen. Dies hat einen hohen Aufwand und Papierverbrauch zur Folge, da die Fragebögen zumeist digital erstellt, im Anschluss ausgedruckt und verteilt werden. Zudem sind papierbasierte Fragebögen nicht interaktiv und irrelevante Fragen können nicht einfach ausgeblendet oder versteckt werden. Dabei könnten die Fragebögen auch direkt digital verteilt und bearbeitet werden. Die Verwendung mobiler Geräte ermöglicht, dass von überall auf die elektronischen Fragebögen zugegriffen werden kann. Dies ist wesentlich schneller und spart somit Zeit. Zudem können die Fragebögen interaktiv während der Bearbeitung verändert und so auf den Benutzer angepasst werden. Ein weiterer Vorteil der elektronischen Fragebögen betrifft die Auswertung. Während bei den papierbasierten Fragebögen jeder Fragebogen zuerst digitalisiert werden muss, können bei den digitalen Fragebögen Auswertungsre-

geln festgelegt werden. Diese können dann zur sofortigen automatischen Analyse und Auswertung der bearbeiteten Fragebögen verwendet werden. An der Universität Ulm wird ein System entwickelt, welches alle genannten Aspekte von der Erstellung bis zur Auswertung eines elektronischen Fragebogens prozessorientiert umsetzt.

1.1 Zielsetzung

Im Rahmen des Projektes soll es möglich sein mit Hilfe von Auswertungsregeln elektronische Fragebögen zu analysieren. Diese Regeln müssen jedoch zuerst erstellt werden, wofür bereits ein erster Editor entwickelt wurde. Das Ziel der hier vorliegenden Arbeit ist es, diesen Editor zur Erstellung von Auswertungsregeln weiterzuentwickeln. Dazu soll die Struktur des Editors geändert, sowie ein neues Konzept für die Erstellung von Regeln entwickelt und umgesetzt werden. Dafür wurde aus mehreren verschiedenen Konzepten ein auf Bäumen basierender Ansatz ausgewählt. Darüber hinaus soll die Bedienbarkeit des Editors verbessert, sowie die Benutzerfreundlichkeit erhöht werden.

1.2 Struktur der Arbeit

Diese Arbeit ist in sechs weitere Kapitel eingeteilt. In Kapitel 2 werden die Grundlagen bezüglich dem Gesamtprojekt und die Ausgangsbasis der Arbeit beschrieben. Kapitel 3 führt funktionale und nicht-funktionale Anforderungen des Editors auf. Diese fließen in die Entwicklung des Konzepts in Kapitel 4 mit ein. Zusätzlich wird der allgemeine Aufbau einer Regel beschrieben. Die Umsetzung und Implementierung des neuen Konzeptes und den damit einhergehenden Änderungen, folgt in Kapitel 5. Kapitel 6 stellt verwandte Arbeiten vor. Als Letztes wird in Kapitel 7 in einem Fazit nochmals ein Blick auf die Anforderungen geworfen. Zusätzlich werden weitere mögliche Verbesserungen und Erweiterungen des Editors aufgeführt.

2

Grundlagen

In diesem Kapitel werden einige Grundlagen der Arbeit beschrieben. Zuerst wird in den allgemeinen Grundlagen das Gesamtsystem *QuestionSys* beschrieben. Anschließend wird der als Grundlage für diese Arbeit dienende Editor zur Erstellung von Regeln näher erläutert.

2.1 Allgemeine Grundlagen

In diesem Abschnitt werden die allgemeinen Grundlagen erläutert. Dazu wird zuerst ein kurzer Überblick über mögliche Einsatzmöglichkeiten von Regeln gegeben, bevor dann näher auf das System *QuestionSys* eingegangen wird.

2.1.1 Einsatzmöglichkeiten von Regeln

Unsere heutigen technologischen Möglichkeiten erlauben es, immer größere Mengen an Daten zu sammeln und zu bearbeiten. Im Bereich der *Big Data* [1] wird versucht, diese großen Datenmengen möglichst schnell auszuwerten und zu analysieren, um so Daten optimal einzusetzen. Es ist denkbar, dass dafür speziell erstellte Regeln eingesetzt werden, die die Daten beispielsweise nach bestimmten Kriterien durchsuchen. Dies bringt auch für Unternehmen Vorteile, zum Beispiel im Bereich der *Business Intelligence* [2], indem durch Analysieren von Geschäftsprozessen Unternehmen bei Entscheidungen oder Zielsetzungen unterstützt werden. Regeln könnten auch beim *Business Activity Monitoring* [3] verwendet werden, um zum Beispiel Risiken von Unternehmen einzuschränken oder zu minimieren, indem Aktivitäten von Unternehmen in Echtzeit analysiert werden.
In dieser Arbeit dagegen, werden die Regeln im Rahmen des Systems *QuestionSys* zum Auswerten und Analysieren von Fragebögen verwendet.

2.1.2 QuestionSys

QuestionSys ist ein System, welches wie bereits in der Einleitung beschrieben, den kompletten Lebenszyklus eines elektronischen Fragebogen begleitet. Dabei dient ein Fragebogen zur Erhebung von Daten. QuestionSys umfasst von der Erstellung eines Fragebogens über das Ausfüllen bis zur Analyse des Fragebogens, alles was notwendig ist, um mit Fragebögen vollständig digital zu arbeiten. Um dies zu ermöglichen, stehen für die verschiedenen Schritte im Lebenszyklus eines Fragebogens verschiedene Programme zur Verfügung. Alle Programme sollen dabei möglichst einfach zu benutzen und selbsterklärend sein. Der Fragebogen selbst wird prozessorientiert erstellt [4]. Manche dieser Schritte, wie zum Beispiel das Ausfüllen von Fragebögen kann auch von mobilen Geräten, wie Smartphones oder Tablets durchgeführt werden. QuestionSys ist, wie in Abbildung 2.1 zu sehen ist, in drei Module aufgeteilt:

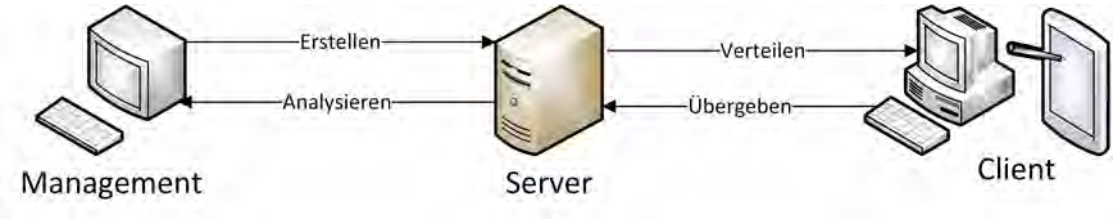

Abbildung 2.1: Verschiedene Module des QuestionSys [5]

Management

Das Management umfasst alle Programme, die an der Erstellung oder Bearbeitung von Fragen und Fragebögen beteiligt sind. Dazu gehört auch der in dieser Arbeit weiterentwickelte Editor, mit welchem passend zu den Fragebögen Regeln zur Auswertung erstellt werden können.

Server

Der Server verknüpft das Management und den Client, indem er Daten austauscht und bereitstellt. So verteilt er zuvor vom Management erstellte Fragebögen und Auswertungsregeln an verschiedene Clients. Diese übergeben die fertig ausgefüllten Fragebögen wieder an den Server. Zudem ist es möglich, dass der Server automatisch Fragebögen auswertet und analysiert. Zum Beispiel, wenn Fragebögen von verschiedenen Clients gemeinsam ausgewertet werden sollen oder eine große Menge an Daten vorliegt, wie es beispielsweise bei Studien der Fall ist.

Client

Zu diesem Teil gehören alle Programme, die clientseitig ausgeführt werden. Dazu zählt das Ausfüllen der Fragebögen, sowie das mögliche Analysieren dieser, mit den vom Server enthaltenen Regeln. Das schnelle clientseitige Analysieren hat den Vorteil, dass der Benutzer sofort und ohne nötige Internetverbindung eine Auswertung der Fragebögen

vorliegen hat. Der Client kann dann die Ergebnisse zum Beispiel für Sammlungen oder weitere Analysen an den Server übergeben.

2.2 Ausgangsbasis der Arbeit

Wie bereits erwähnt, werden zum Analysieren und Auswerten von Fragebögen Auswertungsregeln verwendet. Diese Regeln müssen zu den jeweiligen Fragen beziehungsweise Fragebögen erstellt werden. [6] beschreibt ein Modul des Gesamtsystems, den Editor *Questionrule*, und dient als Grundlage für diese Arbeit. In diesem Editor lassen sich Auswertungsregeln durch verknüpfen von vorgegebenen Komponenten erstellen. Dazu stehen verschiedene mehrfach verwendbare Grafikkomponenten zur Verfügung, die jeweils einen Teil der Regel darstellen und sich miteinander verbinden lassen. Die so erzeugte Regel kann anschließend in eine textuelle Repräsentation überführt werden. Dabei wird sie auch auf Ihre Korrektheit überprüft. Eine Regel ist dabei nichts anderes als ein boolescher Ausdruck, der validiert werden kann. Als Ergebnis können die Möglichkeiten *die Regel trifft zu* oder *die Regel trifft nicht zu* auftreten. Die fertigen Regeln können dann exportiert werden. Damit die Regeln auch passend zu den Fragebögen entwickelt werden können, bekommt der Editor von diesen Informationen in Form von Datenelementen [7], die dann beim Erstellen als Variablen verwendet werden können.

Der Editor hat einige Schwächen, welche durch diese Arbeit behoben werden sollen und im folgenden aufgeführt sind.

- So ist es zwar möglich die Regel zu speichern, jedoch gehen dabei alle Grafikkomponenten verloren, so dass die Regel nicht geändert werden kann, außer die Regel wird komplett neu erstellt.
- Die Grafikkomponenten werden nicht nur zum Anzeigen (View) verwendet, sondern auch als Datenstruktur (Model). Damit die Komponenten einfacher ersetzt werden können, zum Beispiel beim Ändern der Oberfläche, sollte eine bessere Trennung von Model und View erreicht werden.

- Die Bedienbarkeit des Editors soll verbessert werden. So ist das Verbinden der Grafikkomponenten mühsam, da dafür viele Klicks notwendig sind, um die einzelnen Komponenten zu markieren.

In der folgenden Arbeit treten Begriffe auf, die im Kontext des Editors mehrere Bedeutungen haben können. Deshalb sind im Anschluss einige wichtige Begriffe definiert beziehungsweise Äquivalenzen aufgezählt.

Eine *Regel* existiert zum einen in Textform und zum anderen in Baumform. Im grafischen Editor wird sie als Baum erstellt. Dabei muss in diesem Zusammenhang der Unterschied zwischen dem sichtbaren Baum, und der nicht sichtbaren internen Datenstruktur (siehe Kapitel 4) beachtet werden. Der sichtbare Baum umfasst dabei die verbundenen Grafikelemente, die als Baum angeordnet werden. Die Datenstruktur kann zur sichtbaren Form visualisiert werden (siehe Abschnitt 5.3.3). Zusätzlich lässt sie sich zur Regel in Textform konvertieren. Die einzelnen Bestandteile der Regel sind im Baum als Knoten dargestellt (genauer in Kapitel 4).

3

Anforderungen

In diesem Kapitel werden die Anforderungen an den Editor zur Erstellung von Regeln beschrieben. Diese werden in funktionale und nicht-funktionale Anforderungen aufgeteilt und später vom Konzept wieder aufgegriffen.

3.1 Funktionale Anforderungen

Funktionale Anforderungen sind Anforderungen an eine Anwendung, die die Funktionalität beschreiben, beziehungsweise festlegen was die Anwendung alles können soll. Nachfolgend werden die funktionalen Anforderungen des Editors aufgezählt.

FA1 Serialisierung: Eine Regel sollte sich zu jedem Zeitpunkt, auch während der Entstehungsphase, speichern lassen.

FA2 Deserialisierung: Eine gespeicherte Regel soll mit ihrem Baum wieder geladen werden können, damit diese einfach weiter bearbeitet werden kann.

FA3 Erstellung der Regeln: Damit auch Personen ohne Informatik-Kenntnisse den Editor benutzen können, sollen die Regeln einfach erstellt werden können. Dies soll in Form eines Baumes umgesetzt werden, der sich aus verschiedenen Elementen der Regel zusammensetzt.

FA4 Ändern der Regeln: Die Regeln sollten sich ohne großen Aufwand ändern und wieder bearbeiten lassen.

FA5 Interpretation der Regeln: Die Auswertungsregeln sollen einfach interpretierbar sein, damit auch Personen ohne Informatik-Kenntnisse damit arbeiten können.

FA6 Correctness by Construction: Da bei der Erstellung von Regeln schnell Fehler auftreten können, sollte dies möglichst bereits bei der Erstellung verhindert werden, indem der Benutzer Regeln nur korrekt zusammenbauen kann. Zusätzlich soll dem Benutzer durch Meldungen laufend Feedback gegeben werden, z.B. wenn etwas nicht korrekt zusammengebaut wird.

FA7 Fehlermeldungen: Der Editor soll möglichst sinnvolle Fehlermeldungen ausgeben. Diese sind vor allem beim Erstellen und Bearbeiten der Regeln wichtig, damit nicht bereits dabei Fehler entstehen.

3.2 Nicht-funktionale Anforderungen

Zu nicht-funktionale Anforderungen zählen alle Anforderungen, die nicht die Funktionalität beeinflussen. Sie beschreiben mehr die Qualität und Leistung der Anwendung. Im folgenden werden die nicht-funktionalen Anforderungen des Editors aufgeführt.

NFA1 Erweiterbarkeit von Funktionen: Der Editor sollte sich leicht erweitern und anpassen lassen. So können, falls gewünscht, schnell neue Funktionen hinzugefügt werden.

NFA2 Erweiterbarkeit des Models: Neben den Funktionen sollte sich auch das Model der Regeln verändern und erweitern lassen. Dies schließt das Hinzufügen von neuen Regelelementen mit ein.

NFA3 Intuitive Bedienbarkeit: Man sollte den Editor ohne große Eingewöhnungszeit und großen Lernaufwand verwenden können. Um die Bedienbarkeit zu steigern, sollen Hilfsmittel wie Drag & Drop verwendet werden.

NFA4 Zuverlässigkeit: Der Editor sollte zuverlässig arbeiten und die Regeln, sowie Bäume korrekt anzeigen.

NFA5 Robustheit und Stabilität: Damit der Benutzer keine Probleme mit dem Editor hat, sollte dieser stabil und robust laufen, sowie Abstürze vermieden werden.

NFA6 Sicherheit: Da der Editor nur lokal auf dem Gerät des Benutzers läuft und keine Internetverbindung benötigt wird, muss nicht besonders auf die Sicherheit geachtet werden.

4

Konzept

In diesem Kapitel wird das Konzept beschrieben, bevor es im Implementierungskapitel angewendet wird. Zuerst wird in Abschnitt 4.1 eine Regel mit ihren verschiedenen Bestandteilen erklärt. Anschließend wird diese dann in Abschnitt 4.2 als Baum dargestellt.

4.1 Regel

In diesem Abschnitt wird anhand einer Beispielregel erklärt, welche Bestandteile in einer Regel vorkommen können und wie diese miteinander verbunden sind.

Ein Beispiel für eine einfache Regel ist in Abbildung 4.1 dargestellt. Dies könnte zum Beispiel eine Regel zur Auswertung eines medizinischen Fragebogens sein. Die Regel trifft zu, wenn der Patient angegeben hat, dass er raucht oder trinkt und gleichzeitig

mindestens 40 Jahre alt ist. Dann könnten zum Beispiel weitere Schritte, wie zusätzliche Untersuchungen eingeleitet werden. Die Regel selbst besteht aus verschiedenen Bestandteilen, welche in Abbildung 4.1 farblich markiert sind. Beispielsweise sind diese Bestandteile `age` (Variable) und `40` (Konstante), die dann miteinander verglichen werden sollen. Dazu werden sie durch den Vergleichsoperator `>=` verbunden, wobei weitere Vergleichsoperatoren zum Beispiel `>`, `<`, `<=`, `==` und `!=` sind. Der Vergleich besteht immer aus exakt zwei Komponenten und dem Vergleichsoperator, welcher sich in der Mitte der beiden befindet. Eine Komponente kann dabei eine Variable, Konstante oder Funktion sein. Variablen sind in der Beispielregel grün markiert, während Konstanten blau und Vergleichsoperatoren rot markiert sind. Konstanten haben verschiedene Datentypen, wie Strings oder Integer.
Der Letzte noch fehlende Bestandteil einer Regel ist die Verknüpfung (Boolean). Diese verknüpft die Vergleiche mit den booleschen Operatoren `&&` für `und` und `||` für `oder`. In Abbildung 4.1 sind sie mit der Farbe Türkis markiert. Neben den Vergleichen können auch Verknüpfungen untereinander verbunden werden. In Abschnitt 4.2.2 wird darauf noch mehr eingegangen. Die Verknüpfung und der Vergleich können als Hauptbestandteile der Regel angesehen werden, da sie die anderen Bestandteile verbinden.

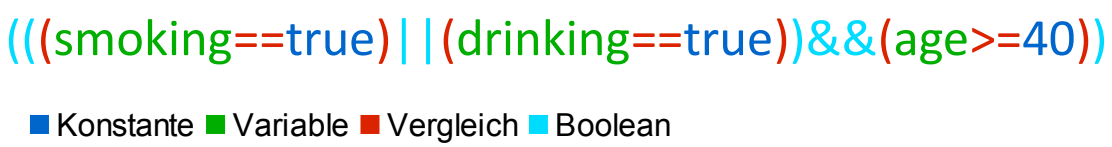

Abbildung 4.1: Beispielregel mit Hervorhebung der Bestandteile

4.2 Regel in Baumform

In diesem Abschnitt wird zuerst die Regel als Baum dargestellt und anschließend alle möglichen Bestandteile des Baumes aufgezählt und erklärt.

4.2.1 Transformation einer Regel in einen äquivalenten Baum

Eine Regel sollte sich nach der funktionalen Anforderung FA3 möglichst einfach erstellen lassen. Dazu bietet es sich an, die Regel als Baum darzustellen, da Bäume leicht erweitert werden können, indem man weitere Knoten und Blätter hinzufügt. Außerdem ist eine Baum-Darstellung, wie in Abbildung 4.2 zu sehen ist, übersichtlicher. Dort ist die Beispielregel von Abbildung 4.1 als Baum dargestellt.
Transformiert wird die Regel, in dem die Verknüpfung gesucht wird, welche andere Regelbestandteile verknüpft, aber selbst kein Teil einer anderen übergeordneten Verknüpfung ist. Dies ist in der Beispielregel die Verknüpfung `&&`. Sie ist gleichzeitig die Wurzel des Baumes, da sie im Baum keinen Vater hat [8]. Ausgehend von dieser, werden alle von ihr verknüpften Regelbestandteile durchlaufen. Auf der linken Seite ist die Verknüpfung `||` und auf der rechten Seite ein Vergleich. Diese ergeben nun den linken und rechten Teilbaum. Die Teilbäume werden nun nach dem selben Schema durchgegangen, bis alle Bestandteile der Regel in einen Baum überführt worden sind. Ein Vergleich wird dabei genauso wie eine Verknüpfung behandelt. Zuerst wird der Vergleichsoperator transformiert und anschließend der linke und rechte Bestandteil.
Wie zu sehen ist, stehen im Baum die verbindenden Bestandteile, wie Verknüpfung oder Vergleich, immer über den anderen Bestandteilen. Sie sind also der Vater der darunterliegenden Bestandteile, können jedoch auch gleichzeitig Kind des darüberliegenden Bestandteils sein. Bestandteile werden im weiteren Verlauf der Arbeit im Baum auch als Knoten bezeichnet.

4.2.2 Bestandteile des Baumes

Nachdem bereits eine Beispielregel als Baum dargestellt wurde, wird nun in diesem Abschnitt allgemein erklärt, welche Bestandteile ein solcher Baum haben kann und wie sich diese verbinden lassen. Da der Baum nichts anderes als eine Regel in anderer Form ist, ist es nur logisch, dass dieser die gleichen Bestandteile, wie die Regel aus Abschnitt 4.1 hat. Eine Regel soll aus beliebig vielen Bestandteilen bestehen können. Damit dies funktioniert, muss festgelegt werden, welche Bestandteile sich mit welchen

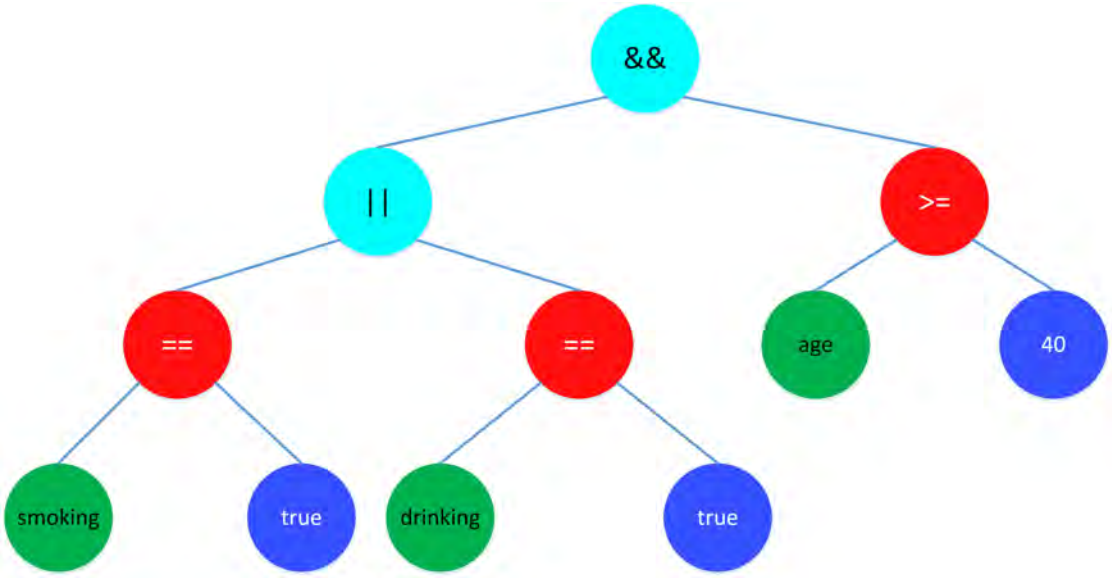

Abbildung 4.2: Beispielregel als Baum

Bestandteilen verbinden lassen, damit die Regel immer syntaktisch korrekt ist. Zu beachten ist, dass jeder Knoten nur einen Vater-Knoten haben kann und es beim Verbinden egal ist, ob die Knoten selbst bereits Kinder haben.

Verknüpfung: Die Verknüpfung ist in Abbildung 4.3 zu sehen. Sie kann Vergleiche oder andere Verknüpfungen als Kinder haben. Dabei kann sie beliebig viele Kinder haben, solange sie mindestens zwei besitzt. Daraus folgt auch, dass die Verknüpfung immer ein innerer Knoten [9] des Baumes ist. Es ist auch egal, in welchem Verhältnis sich die Kinder zusammensetzen.

Vergleich: Der Vergleich kann, wie in Abbildung 4.4 zu sehen ist, folgende drei Bestandteile als Kinder haben: Variable, Konstante und Funktion. Es ist egal, welche der Bestandteile in welcher Reihenfolge und in welcher Kombination vorkommen, solange ein Vergleich immer exakt zwei Kinder hat. Zum Beispiel ist es möglich, dass ein Ver-

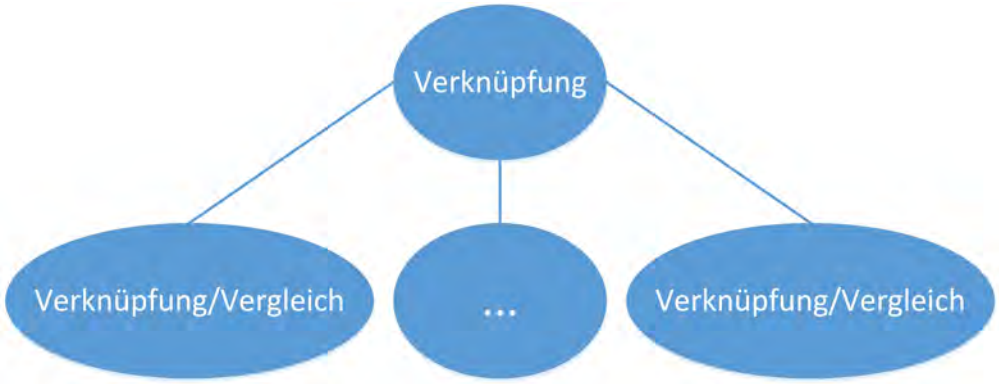

Abbildung 4.3: Mögliche Kindknoten für einen Knoten Verknüpfung

gleich zwei Variablen als Kinder hat. Im Baum wird der Vergleich, wie in Abbildung 4.2 zu sehen ist, mit dem Vergleichsoperator angezeigt. Beim Vergleich handelt es sich wie bei der Verknüpfung um einen inneren Knoten.

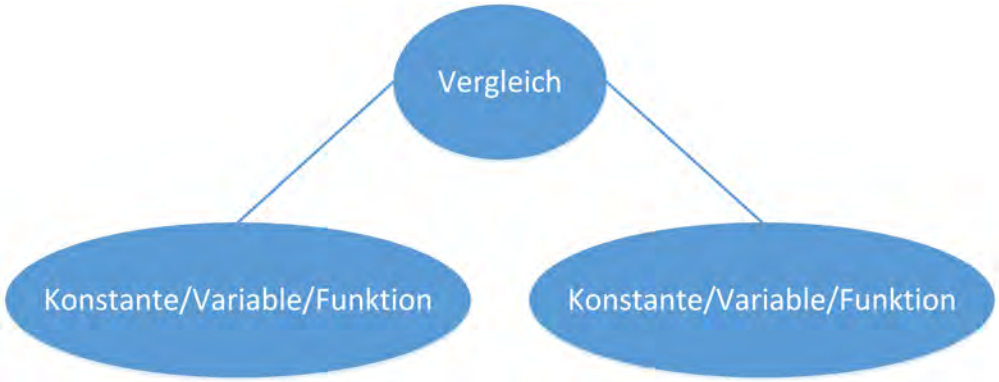

Abbildung 4.4: Mögliche Kindknoten für einen Knoten Vergleich

Variablen, Konstanten und Funktionen können selbst keine anderen Knoten als Kinder haben. Daher sind sie im Baum immer Blätter [9]. Sie haben nur den Vergleich als Vater. Variablen und Funktionen werden im Baum mit Ihrem Namen angezeigt, während Konstanten mit dem Wert angezeigt werden.

Um die Bedienbarkeit und den Komfort zu erhöhen, soll es egal sein, in welcher Reihenfolge die einzelnen Knoten verbunden werden, solange die obigen Regeln eingehalten werden. Dadurch kann es vorkommen, dass verschiedene Teilbäume entstehen oder Bestandteile unverbunden bleiben. Damit der Editor weiß, welcher der Teilbäume der Hauptbaum ist, existiert ein extra Wurzelknoten.

Wurzel (Root): Der Wurzelknoten ist ein Bestandteil des Baumes, der nur einmal vorkommen darf. Er hat selbst keinen Vater und dient im Editor als Einstiegspunkt, um aus dem Baum eine Regel, wie in Abbildung 4.1 abzuleiten. Als Kinder kommen, wie in Abbildung 4.5 zu sehen ist, nur Verknüpfungs- oder Vergleichsbestandteile in Frage. Dabei darf die Wurzel aber nur maximal ein Kind haben.

Abbildung 4.5: Mögliche Kindknoten für einen Knoten Wurzel

Dadurch, dass sich die verschiedenen Knoten des Baumes nur auf bestimmte Weisen verbinden lassen, ist die funktionale Anforderung FA6 bereits durch das Konzept erfüllt. Der Benutzer kann nur syntaktisch korrekte Regeln erstellen.

4.2.3 Struktur des Baumes

In diesem Abschnitt wird untersucht, welche Struktureigenschaften ein Baum unter den Bedingungen aus Abschnitt 4.2.2 besitzt. Dafür werden verschiedene Baumstrukturen angeschaut.

Wie oben festgelegt wurde, kann der Verknüpfungsknoten beliebig viele Kinder haben, solange er mindestens zwei davon hat. Da er auch der einzige Knoten ist, der mehr als zwei Kinder haben kann, liegt es an ihm, ob der Baum ein Binärbaum ist. Sind es mehr als zwei, kann der Baum in diesem Fall kein Binärbaum sein, da er dafür maximal zwei Kinder pro Vaterknoten haben darf [10]. Es kann jedoch durchaus vorkommen, dass der Baum unter bestimmten Bedingungen ein Binärbaum darstellt. Und zwar dann, wenn jeder Verknüpfungsknoten genau zwei Knoten als Kinder hat. Dies ist zum Beispiel in Abbildung 4.2 der Fall.

Wenn es darum geht, ob der Baum balanciert ist, kommt es, wie schon beim Binärbaum, auf den Baum beziehungsweise auf die Bestandteile und deren Verteilung im Baum an. Es können also Fälle auftreten, in denen der Baum balanciert ist, aber auch Fälle, in denen er nicht balanciert ist. Ein Beispiel für einen balancierten Baum ist in Abbildung 4.6 dargestellt. Der Baum der Beispielregel (siehe Abbildung 4.2) ist währenddessen unbalanciert.

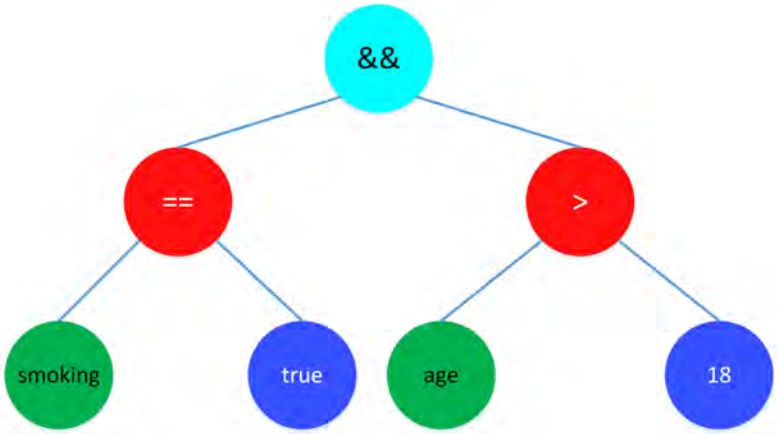

Abbildung 4.6: Beispiel für einen balancierten Baum

Um den Baum jederzeit zu einer Regel in Textform transformieren zu können, muss der Baum in irgendeiner Weise durchlaufen werden. Dazu können verschiedene Traversierungsarten verwendet werden. Nachfolgend werden drei Arten anhand der Beispielregel aus Abbildung 4.2 abgebildet.

Die erste Variante ist die `PreOrder` Traversierungsart. Von der Wurzel ausgehend, wird zuerst die Wurzel, dann der linke Teilbaum und anschließend der rechte Teilbaum durchlaufen [11]. Die Durchlaufreihenfolge für die Beispielregel ist in Abbildung 4.7 dargestellt.

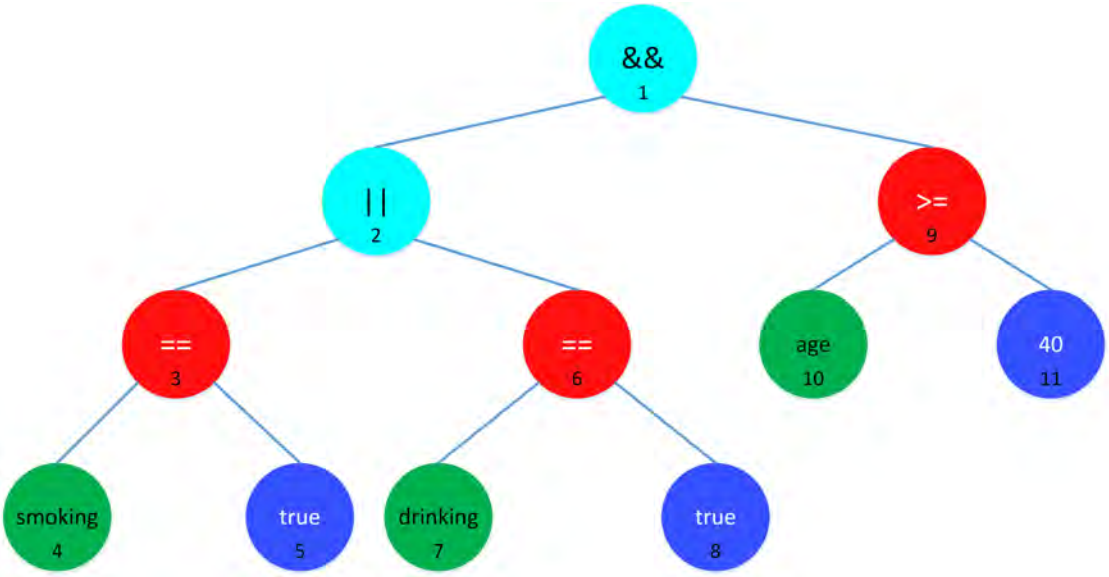

Abbildung 4.7: Beispielregel mit Durchlaufreihenfolge für PreOrder

Die nächste Art ist `PostOrder`. Von der Wurzel ausgehend, wird zuerst der linke Teilbaum, dann der rechte Teilbaum und dann die Wurzel traversiert [11]. In Abbildung 4.8 ist wieder die Durchlaufreihenfolge eingezeichnet.

Als Letztes wird der `InOrder` Durchlauf erklärt. Von einem Knoten ausgehend, wird zuerst der linke Teilbaum, dann der Knoten selbst und dann der rechte Teilbaum abgearbeitet [11]. Hier ist im Gegensatz zu den anderen Traversierungsmethoden gewährleistet, dass die Vergleichsoperatoren und boolesche Operatoren immer in der Mitte der Kinder

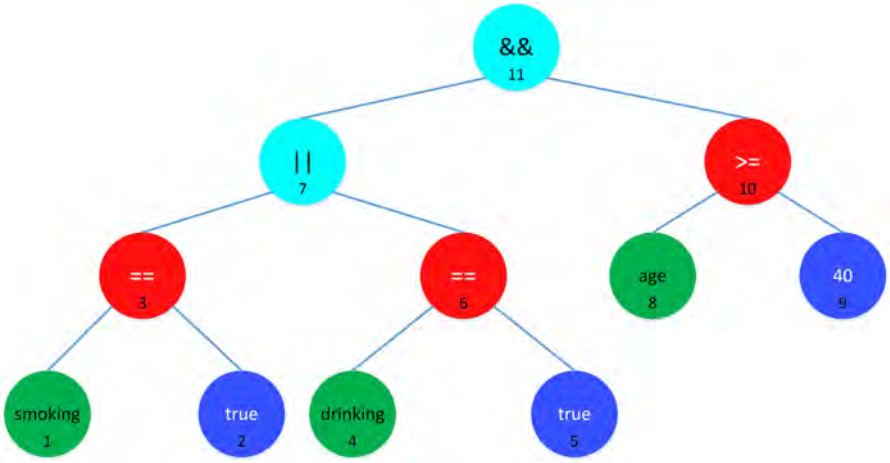

Abbildung 4.8: Beispielregel mit Durchlaufreihenfolge für PostOrder

sind. Durchläuft man Abbildung 4.9 auf diese Weise, erhält man, mit zusätzlicher Klammerung, die Regel aus Abbildung 4.1.

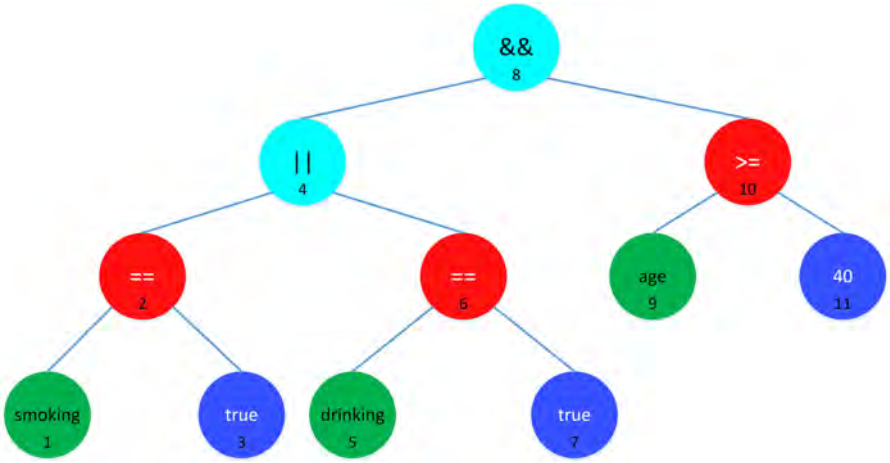

Abbildung 4.9: Beispielregel mit Durchlaufreihenfolge für InOrder

Ein Baum, der nach obigen Regeln erstellt wird, kann verschiedene Eigenschaften haben. Diese hängen jedoch stark davon ab, auf welche Art und mit welchen Bestandteilen der Baum erstellt wird. Dies liegt vor allem am Verknüpfungsknoten, da durch diesen der Baum in die Tiefe und in die Breite beliebig groß werden kann.

5

Implementierung

In diesem Kapitel wird die Implementierung der Arbeit beschrieben. Zuerst wird in Abschnitt 5.1 ein kleiner Überblick über Eclipse RCP, sowie die Architektur des Editors gegeben. Im nächsten Abschnitt wird erklärt, wie das Konzept aus Kapitel 4 umgesetzt wurde. Im letzten Abschnitt werden dann einige an das Konzept angepasste Funktionen des Editors vorgestellt.

5.1 Eclipse RCP

Eclipse RCP steht für *Rich Client Platform* und ist eine Plattform, die zur Entwicklung von Desktop-Anwendungen eingesetzt wird. Dazu stellt sie ein Grundgerüst zur Verfügung, welches um eigene Anwendungen ergänzt werden kann [12]. Die Eclipse IDE ist mit dem Fokus auf Softwareentwicklung mit Eclipse RCP gebaut. Beide basieren auf

einem dynamischen Plug-in Model. Zusätzlich steht bei Eclipse RCP zum Aufbau der Benutzeroberfläche die Workbench als Grundlage zur Verfügung. Diese kann beliebig angepasst werden. Dazu sind viele verschiedene Grafikkomponenten bereitgestellt. Eclipse RCP Anwendungen können ohne Probleme in Eclipse IDE entwickelt werden. Darüber hinaus bietet Eclipse RCP den Vorteil, dass bereits entwickelte Anwendungen einfach zu erweitern sind [12, 13].

5.1.1 Architektur

In [6] wurde die Architektur von Eclipse RCP dargestellt und anschließend die Architektur des Editors ausführlich beschrieben. In der hier vorliegenden Arbeit wird der Editor weiterentwickelt, weshalb die beschriebene Architektur weiterhin gilt. In Abbildung 5.1 ist diese Architektur dargestellt. [6] benutzt für die verschiedenen Teilbereiche des Editors unterschiedliche Manager. Die in den nachfolgenden Abschnitten beschriebene Umsetzung des Konzeptes aus Kapitel 4 gliedert sich dabei hauptsächlich im Bereich des `Regel Manager` ein. Es gibt aber in der Folge auch kleine Änderungen an der `Benutzeroberfläche`, sowie beim `Projekt Manager`. Der Änderungsbereich ist in der Abbildung rot eingezeichnet.

5.1.2 Model-View-Presenter

Eclipse RCP wird mit dem Model-View-Presenter (MVP) Konzept verwendet. MVP ist aus dem Model-View-Controller (MVC) entstanden, trennt Model und View allerdings wesentlich strikter als MVC. Beide können dabei nur mit dem Presenter kommunizieren. Das bringt den Vorteil, dass Model und View beliebig ausgetauscht werden können, ohne den jeweils anderen Teil ändern zu müssen [14]. In Abbildung 5.2 ist das MVP Konzept abgebildet. Die Pfeile zwischen den verschiedenen Teilen symbolisieren die Kommunikation.

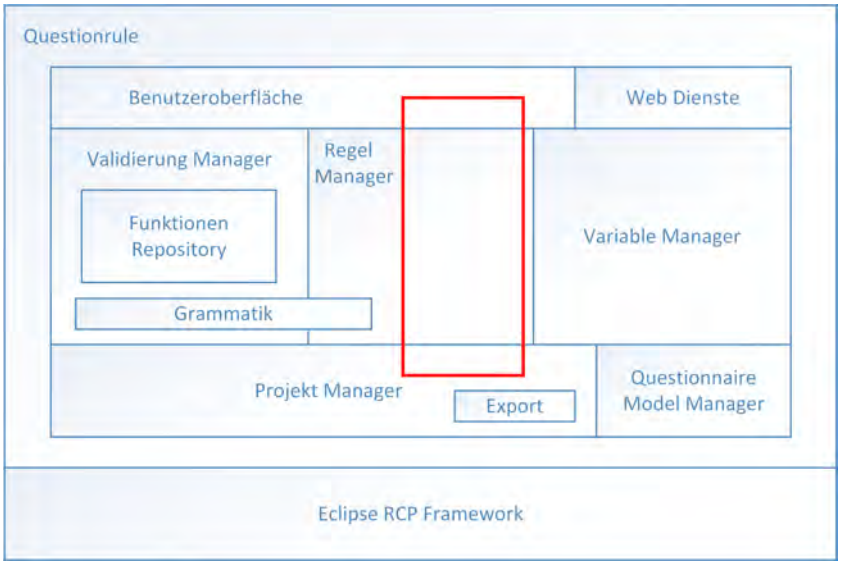

Abbildung 5.1: Architektur der Anwendung nach [6]

Model: Enthält die verschiedenen Daten und die Logik des Programms. Kann dabei aber nur mit dem Presenter kommunizieren.

View: Die View zeigt die Oberfläche an und nimmt Benutzereingaben entgegen [15]. Diese leitet sie an den Presenter weiter.

Presenter: Der Presenter vermittelt zwischen View und Model. Dazu nimmt er Anfragen der View entgegen und leitet sie an das zuständige Model weiter. Bearbeitete Anfragen leitet er, falls nötig, wieder an die View zurück.

Abbildung 5.2: Konzept des Model-View-Presenter

Vergleicht man nun das MVP Konzept mit dem MVC Konzept, welches in Abbildung 5.3 dargestellt ist, kommt man zum Ergebnis, dass das MVP Konzept für diese Anwendung besser ist. Dies liegt vor allem daran, dass beim MVC View und Model untereinander kommunizieren und es deshalb schwieriger ist einzelne Komponenten auszutauschen.

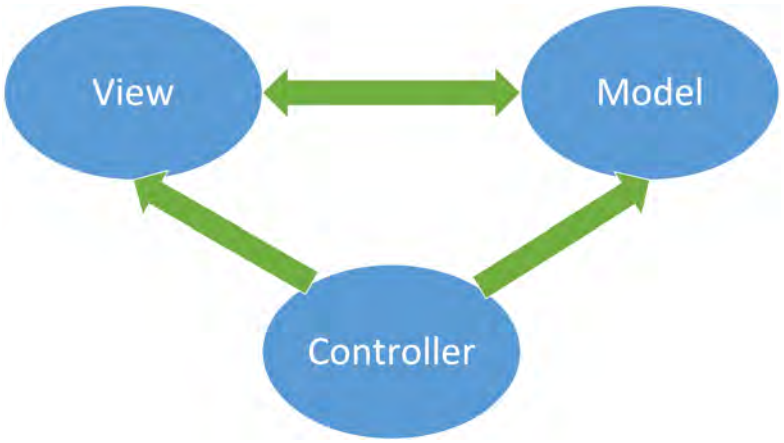

Abbildung 5.3: Konzept des Model-View-Controller

5.2 Umsetzung des entwickelten Konzepts

In diesem Abschnitt wird beschrieben, wie der bereits vorhandene Editor an das Konzept angepasst wurde. Zusätzlich wird ein Überblick über die neue Funktionsweise des Editors vermittelt.

Erstellt man eine Regel, so wird automatisch eine leere Regel generiert. Diese ist in Abbildung 5.4 mit eingezeichneten Bereichen der Regel zu sehen. Es empfiehlt sich im Meta Daten Bereich den Namen der Regel zu ändern, sowie eine Beschreibung hinzuzufügen.

Auf der rechten Seite der Abbildung 5.4 sieht man im Bereich der Werkzeuge bereits die einzelnen Bestandteile des Konzeptes. Diese können per Drag & Drop in den Bereich der Zeichenfläche gezogen werden. In diesem Bereich befindet sich bereits der Wurzelknoten, welcher direkt beim Erstellen der neuen Regel generiert wird. Beim Einfügen

der Knoten laufen zwei Vorgänge ab. Zum einen wird für den jeweiligen Knoten eine passende graphische Komponente erstellt. Und zum anderen wird ein passender Knoten zur internen Verwaltung erstellt. Was dabei genau passiert, wird in den folgenden Abschnitten erklärt.

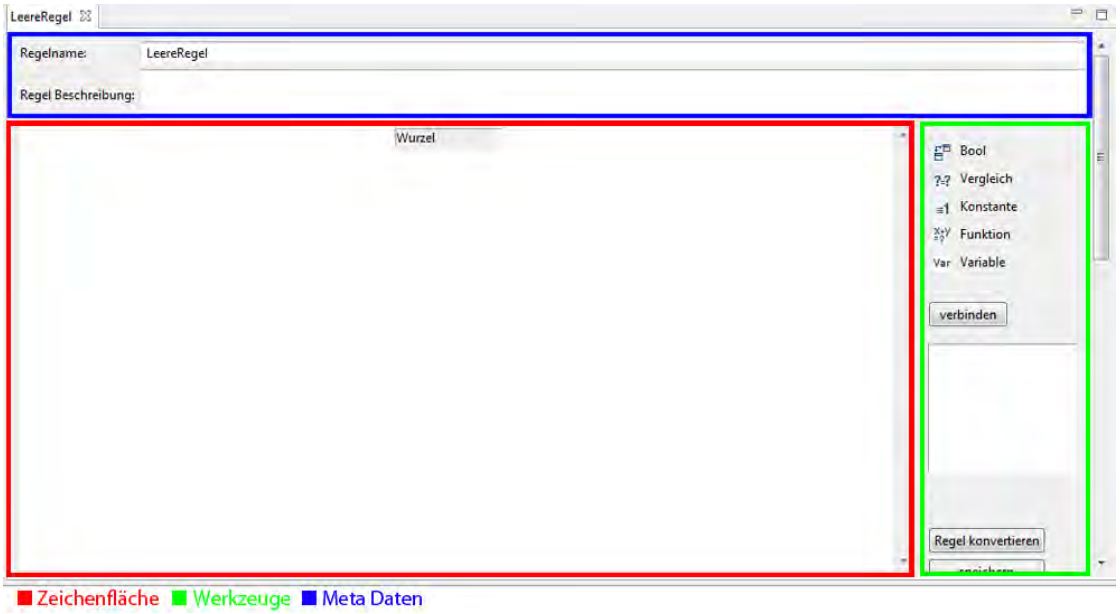

Abbildung 5.4: Start beim Erstellen einer neuen Regel

Für alle möglichen Bestandteile des Baumes aus Abschnitt 4.2.2 wurde eine eigene Klasse angelegt. Jede dieser Klassen erbt von einer Basisklasse namens `BasicRuleNode` und muss den passenden Typ aus `EnumNodes` enthalten.

```
1 public enum EnumNodes {
2       ROOT, BOOL, COMPARSION, CONSTANT, FUNCTION, VARIABLE;
3 }
```

Listing 5.1: Enum der verschiedenen Knotentypen

Diese Klassen sind alle im neu eingefügten Package `BasicTreeModel`, welches in der Tabelle 5.2 dargestellt ist, zusammengefasst. Damit nun aus den Knoten ein Baum

erstellt werden kann, gibt es zusätzlich die Klasse `RuleTree`. Diese verwaltet den Baum, wobei die einzelnen Knoten in einer `HashMap<Long, BasicRuleNode>`, mit einem `Identifier` vom Datentyp `long` als Schlüssel, verwaltet werden. Vater-Kind-Beziehungen zwischen den Knoten werden dadurch dargestellt, dass jeder Knoten eine Referenz zu seinem Vaterknoten und eine Referenz zu seinen vorhandenen Kindknoten hat. Zusätzlich hat der Baum eine Variable vom Typ `BasicRuleNode` zur Darstellung des Wurzelknoten. Dieser wird benötigt, um schnell einen Eintrittspunkt in den Baum zu haben. Beim Hinzufügen der Knoten zum Baum, wird darauf geachtet, dass die Vater-Kind-Beziehungen aus dem Konzept eingehalten werden.

Klasse	Beschreibung
BasicRuleNode	Alle anderen Knotenklassen erben von dieser Klasse. Sie enthält alle nötigen Informationen, die ein Knoten braucht.
BoolRuleNode	Repräsentiert den Verknüpfungsknoten und ist immer ein innerer Knoten.
ComparsionRuleNode	Stellt den Vergleichsknoten dar. Kann im Baum nur ein Blatt sein.
ConstantRuleNode	Stellt eine Konstante dar. Ist immer ein Blatt.
FunctionRuleNode	Stellt den Funktionsknoten dar. Ist immer ein Blatt.
RootRuleNode	Repräsentiert den Wurzelknoten. Darf nur einmal pro Regel vorkommen.
VariableRuleNode	Stellt den Variablenknoten des Baumes dar. Ist ein innerer Knoten.
RuleTree	Verwaltet die Knoten, die zu einem Baum zusammengefügt werden. Muss immer eine Wurzel haben.

Tabelle 5.1: Klassen aus dem Package BasicTreeModel

Damit das Verbinden und Zusammenführen von Knoten oder ganzen Teilbäumen nicht eingeschränkt wird, sollen, wie bereits in Abschnitt 4.2.2 beschrieben, mehrere Teilbäume parallel existieren können. Dazu wurde im zuständigen `Presenter` eine Liste zur Verwaltung der Bäume implementiert. Diese enthält in der ersten Stelle den Hauptbaum, welcher bereits beim Erstellen einer neuen Regel angelegt wird. Dieser hat als einziger Baum seine Wurzel vom Typ `RootRuleNode`, welche sich nicht ändern lässt. Der Hauptbaum ist auch der Baum, welcher beim Klick auf die Schaltfläche `Regel`

`konvertieren` in eine Regel umgewandelt wird. Der Wurzelknoten dient dann als Einstieg um den Baum zu durchlaufen. Alle anderen Teilbäume werden, falls nötig, beim Verbinden von Knoten (näheres in Abschnitt 5.3.1) erstellt und gegebenenfalls wieder gelöscht, beziehungsweise mit anderen Teilbäumen fusioniert. In Abbildung 5.5 ist ein Beispiel zu sehen, wie Teilbäume bei der Erstellung einer Regel entstehen können. Dort wird die Beispielregel aus Abbildung 4.1 dargestellt und einzelne Knoten sind bereits zu Teilbäumen verbunden.

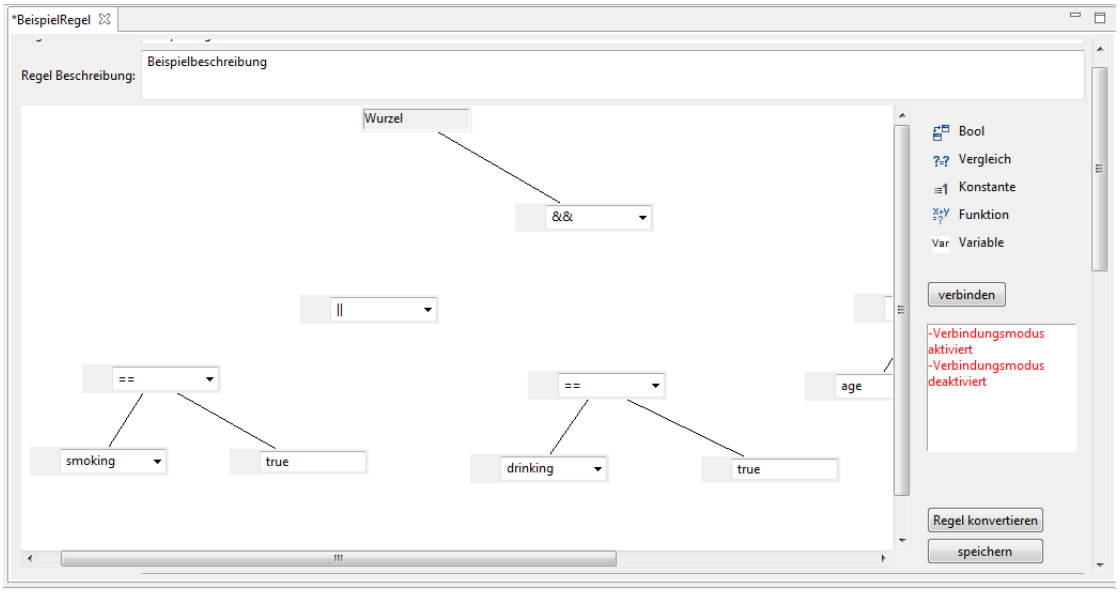

Abbildung 5.5: Teilbäume bei der Erstellung einer Beispielregel

Damit überhaupt Teilbäume entstehen können, müssen zuerst die einzelnen Knoten hinzugefügt werden. Dazu wird beim Platzieren eines Knotens neben einem Grafikobjekt ein passendes Knotenobjekt erstellt. Dieses sieht man in Listing 5.2 am Beispiel eines Variablenknotens. Zuerst wird eine Instanz der Klasse `VariableRuleNode` angelegt. Anschließend wird mit dem in Zeile 1 übergebenen `Rectangle bounds` die Größe und Position des zuvor angelegten Grafikobjektes gesetzt. Danach wird ein `Identifier` anhand einem Zähler `nodeCount` generiert. Dieser entspricht der Anzahl der bereits hinzugefügten Knoten und wird anschließend erhöht. Davor wird der Knoten noch mit dem `Identifier` als Schlüssel zur `HashMap<Long, BasicRuleNode> unusedNodes`

hinzugefügt. Diese verwaltet alle Knoten, die keinem Baum zugeordnet sind. Am Schluss wird noch der `Identifier` zurückgegeben, damit die jeweilige Grafikkomponente zugeordnet werden kann.

```
1 public long addVariableNode(Rectangle bounds) {
2         VariableRuleNode variableNode = new VariableRuleNode();
3         variableNode.setRectangleBounds(bounds);
4         variableNode.setIdentifier(nodeCount);
5         unusedNodes.put(nodeCount, variableNode);
6         nodeCount++;
7         return variableNode.getIdentifier();
8     }
```

Listing 5.2: Hinzufügen von neuen Knoten

Wird ein Knoten in der Nähe des Randes der Zeichenfläche platziert, erweitert sich diese, damit immer ausreichend Platz zur Verfügung steht, um weitere Knoten hinzufügen zu können. Um zu verhindern, dass der komplette Bildschirm von der Fläche bedeckt wird, erscheinen ab einer bestimmten Größe Scrollbars.

5.3 Anpassung der Funktionen des Editors

In diesem Abschnitt wird beschrieben, wie die wichtigen Funktionen auf das neue Konzept angepasst und erweitert wurden. Dazu wird zuerst das Verbinden von zwei Knoten beschrieben, dann das Laden und Speichern der kompletten Regel mit Baum. Anschließend wird diskutiert, wie die Baumstruktur zum Beispiel nach dem Laden visualisiert werden kann.

5.3.1 Verbinden von zwei Knoten

In diesem Abschnitt wird darauf eingegangen, wie sich das Verbinden von zwei Knoten verändert hat und was passiert, wenn man eine Verbindung wieder löschen will.

Die verschiedenen Schritte beim Verbinden sind im Sequenzdiagramm (siehe Abbildung 5.6) zu sehen. Zuerst muss der Verbindungsmodus aktiviert werden, indem auf den `Verbinden` Button geklickt wird. Solange der Verbindungsmodus aktiviert ist, können im nächsten Schritt zwei Knoten ausgewählt werden, die miteinander verbunden werden sollen. Dazu wählt man zuerst den Vaterknoten (dieser wird rot markiert) und anschließend den Kindknoten aus. Dieser wird ebenfalls markiert, allerdings wird diese Markierung nicht dargestellt, da sofort zu Schritt drei übergegangen wird, in dem der Vaterknoten mit dem Kindknoten verbunden wird.

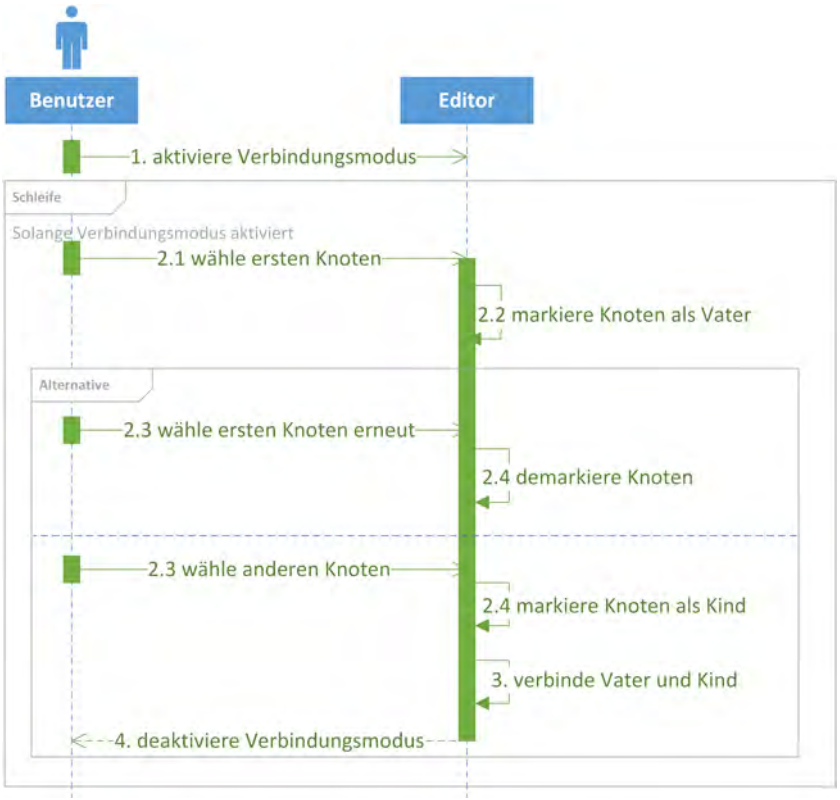

Abbildung 5.6: Sequenzdiagramm zum Verbinden zweier Knoten

Dabei geschehen mehrere Vorgänge. Zum einen wird geprüft, ob sich die Knoten nach dem Konzept verbinden lassen. Trifft dies zu, werden die Knoten zu einem Baum zusammengefügt. Dabei kann es sein, dass mindestens einer der Knoten bereits Teil eines Baumes ist. In diesem Fall wird kein neuer Baum erstellt, sondern der nicht im Baum enthaltene Knoten in den Baum integriert. Gegebenenfalls werden zwei bereits existierende Bäume miteinander fusioniert. Anschließend wird eine visuelle Verbindung gezeichnet und die Markierungen entfernt. Abschließend wird der Verbindungsmodus deaktiviert. Dieser wird ebenfalls deaktiviert, falls Fehler auftreten. Zum Beispiel, wenn der Kindknoten bereits einen Vaterknoten hat. Zusätzlich gibt es auf der rechten Seite (siehe Abbildung 5.7) ein `FeedbackLabel`, in dem angezeigt wird, wann der Verbindungsmodus aktiviert, deaktiviert oder ob ein Fehler aufgetreten ist.

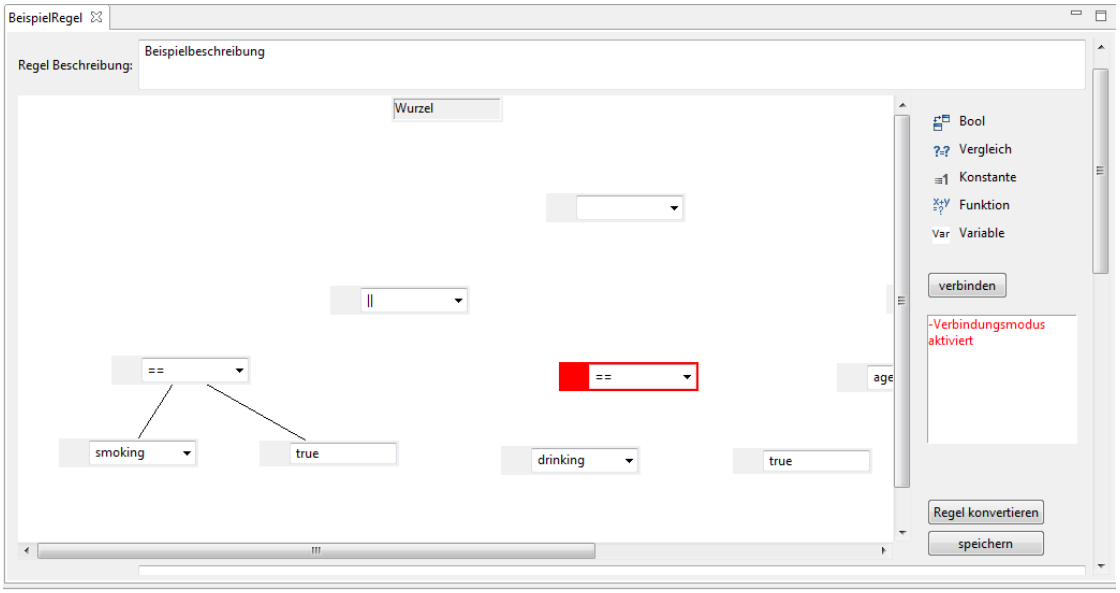

Abbildung 5.7: Vorgang beim Verbinden zweier Knoten. Vaterknoten ist bereits markiert.

Um die Bedienbarkeit zu erhöhen, können Verbindungen auch entfernt werden. Dazu muss mit einem Rechtsklick auf einen Knoten geklickt werden. Es öffnet sich dann ein Fenster, welches die Auswahlmöglichkeit `Verbindung zu Vater löschen` beinhaltet. Im Hintergrund wird dann, je nach ausgewähltem Knotentyp und Position im Baum, der Knoten aus dem Baum entfernt, beziehungsweise der Baum getrennt. Bleibt der

Knoten alleine übrig, wird er wieder zu den unbenutzten Knoten hinzugefügt.

5.3.2 Laden und Speichern der Regel

Bisher war es so, dass nur die Regel in Textform gespeichert wurde, ohne den davor erstellten Baum. Dies hatte zur Folge, dass eine Regel, die einmal geschlossen wurde, nicht mehr bearbeitet werden konnte, ohne den Baum erneut zu erstellen. Deshalb wurde das Speichern und Laden folgendermaßen erweitert:

Speichern des Regelbaumes

Wenn der Benutzer den Speichervorgang einleitet, werden zuerst Informationen (Name und Beschreibung der Regel), sowie die konvertierte Regel, von der View an den zugehörigen Presenter geleitet. Bisher war es so, dass diese Informationen dann in eine XML-Datei geschrieben werden. Das Speichern wurde nun, so wie im Sequenzdiagramm in Abbildung 5.8 dargestellt, geändert. Im dritten Schritt wird ein `SaveObject` erstellt, welches die Liste aller Bäume, die Hashmap mit allen unverbundenen Knoten und den Zähler `nodeCount` enthält. Der Zähler wird benötigt, damit dieser nach dem Laden wieder auf dem gleichen Stand ist und somit die Eindeutigkeit der Identifier vorhanden bleibt. Das `SaveObject` wird im nächsten Schritt an den `SaveObjectWriter` weitergeleitet. Dieser serialisiert das `SaveObject` mit Hilfe eines `ObjectOutputStream`. Dazu implementieren alle Knotenklassen, sowie die Klasse RuleTree das Interface `java.io.Serializable`. Das `SaveObject` und der `SaveObjectWriter` befinden sich beide ebenfalls im Package BasicTreeModel, siehe dazu auch Tabelle 5.3.2. Im nächsten Schritt wird, wie bisher auch, die XML-Datei erstellt. Es wird darauf geachtet, dass beide Dateien den gleichen Namen haben, damit diese eindeutig der gleichen Regel zugeordnet werden können. Beim Löschen einer Regel, werden beide Dateien gelöscht.

Klasse	Beschreibung
SaveObject	Ermöglicht alle vorhandenen Regelkomponenten zu speichern.
SaveObjectWriter	Stellt Methoden zum Speichern und Laden des SaveObjects zur Verfügung

Tabelle 5.2: Klassen aus dem Package BasicTreeModel Teil 2

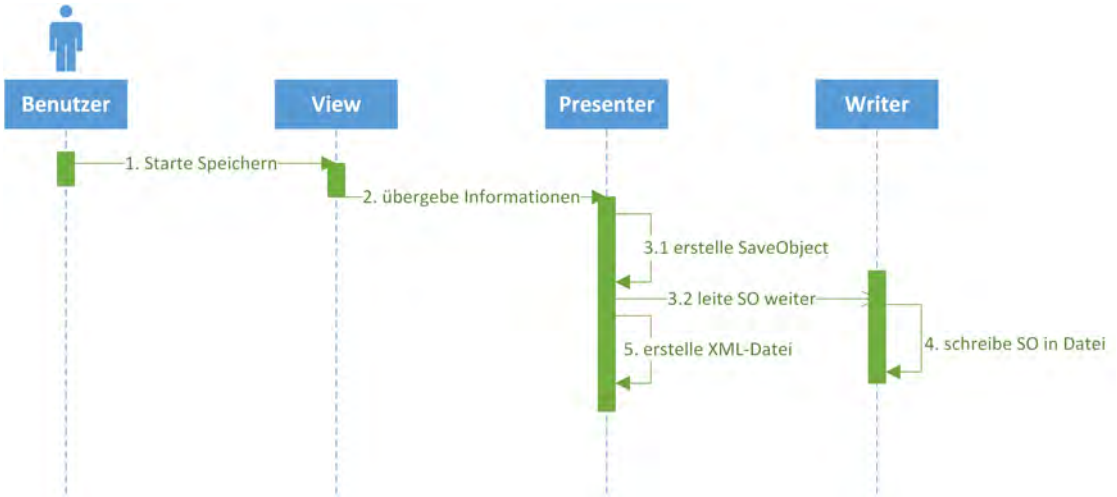

Abbildung 5.8: Sequenzdiagramm zum Speichern der Regel

Laden des Regelbaumes

Das Laden einer Regel wurde auf die neue Funktionsweise des Speicherns einer Regel angepasst. Dazu wurden zusätzlich zum bisherigen Laden folgende Schritte, die im Sequenzdiagramm in Abbildung 5.9 dargestellt sind, hinzugefügt. Zuerst muss der Ladevorgang gestartet werden. Dies geschieht normalerweise dann, wenn eine bereits bestehende Regel erneut geöffnet wird. Im Presenter gibt es dann die ersten Änderungen. Dort wird geschaut, ob für die zu ladende Regel bereits eine Speicherdatei vorhanden ist. Ist dies der Fall, wird die Lademethode im `SaveObjectWriter` aufgerufen. Diese deserialisiert mit Hilfe eines `ObjectInputStreams` das `SaveObject` und übergibt dieses im vierten Schritt an den Presenter. Im nächsten Schritt wird überprüft, ob das SaveObject überhaupt existiert. Trifft dies zu, wird dieses aufgelöst, indem die

darin enthaltenen Informationen initialisiert werden. Existiert dieses nicht, müssen diese Objekte neu angelegt werden. Dies ist zum Beispiel der Fall, wenn eine neue Regel erstellt wird oder eine Regel nie gespeichert wurde. Im letzten Schritt wird dann das Initialisieren der View eingeleitet, in dem alle Informationen dem Presenter übergeben werden. Dort wird dann der Baum visualisiert.

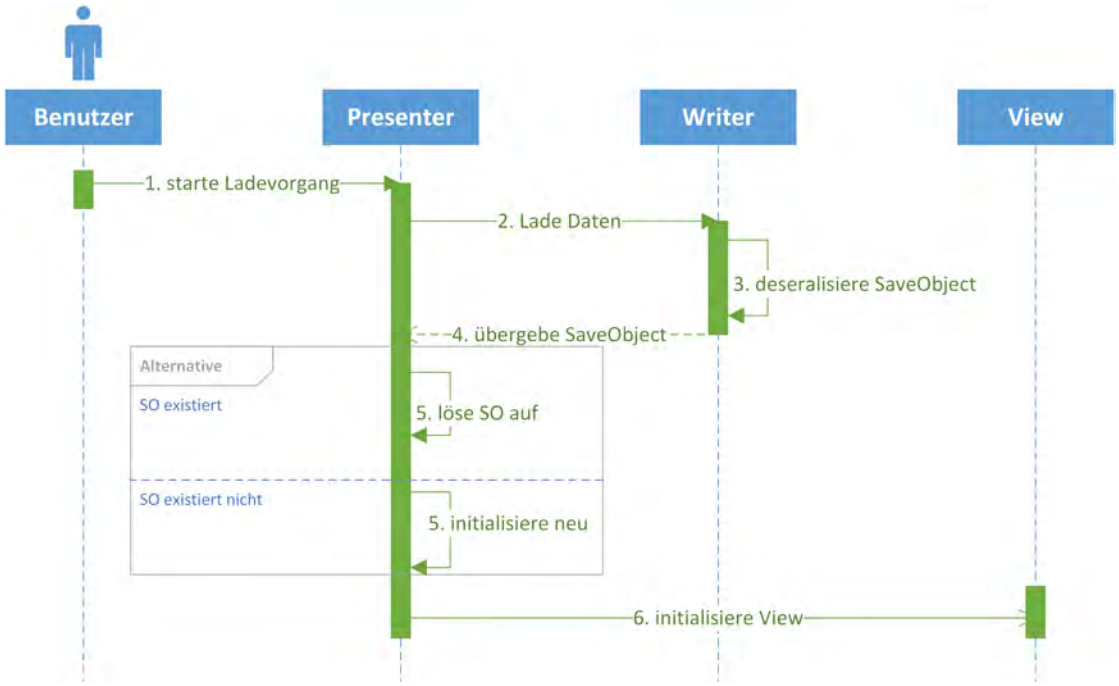

Abbildung 5.9: Sequenzdiagramm zum Laden einer Regel

5.3.3 Baum visualisieren

Beim Visualisieren der internen Baumstruktur, müssen alle Teilbäume, sowie alle unverbundenen Knoten visualisiert werden. Dazu wird jeder Baum, wie im Konzept beschrieben, von der Wurzel per `InOrder` durchlaufen. Für jeden Knoten wird dann entsprechend dem Typ eine passende graphische Komponente erstellt. Damit diese im gleichen Zustand wie im Moment des Speicherns erstellt werden kann, müssen alle Knoten der Baumstruktur bei einer Änderung aktualisiert werden. Zum Beispiel

muss beim Verschieben eines Knotens, die Position von diesem sowohl in der internen Baumstruktur als auch im sichtbaren Grafikelement geändert werden. Beim Durchlaufen eines Baumes werden dann noch zusätzlich die Verbindungen zwischen den Knoten visualisiert. Sind alle Bäume durchlaufen und damit vollständig visualisiert, werden die noch unverbundenen Knoten auf die gleiche Weise platziert und dargestellt.

6

Verwandte Arbeiten

In diesem Kapitel werden verwandte Arbeiten zu dem in dieser Arbeit entwickelten Editor vorgestellt. Zuerst wird in Abschnitt 6.1 das kommerzielle System *Visual Rules* erläutert. Im darauffolgenden Abschnitt wird das Open-Source-System *Huginn* vorgestellt. Anschließend werden beide in Abschnitt 6.3 mit dem in dieser Arbeit weiterentwickelten Editor verglichen.

6.1 Visual Rules BRM

Visual Rules BRM [16] ist ein von Bosch Software Innovations entwickeltes Business Rules Management System. Mit diesem ist es möglich, Geschäftsregeln zu erstellen, um so Geschäftsentscheidungen darzustellen und zu automatisieren. Zur Erstellung der Regeln steht ein grafischer Editor zur Verfügung. Abbildung 6.1 zeigt einen Screenshot

der Erstellung einer Preisberechnung. Zur Modellierung der Regeln stehen verschiedene Elemente, welche sich in Farbe, Form und Funktion unterscheiden, zur Auswahl. Die gelbe Raute in Abbildung 6.1 steht für eine Entscheidung. Zum Beispiel, dass eine Börsentransaktion erfolgen soll. Mit dem nachfolgenden Element, welches eine andere Regel aufruft, wird anschließend der Preis berechnet. Mit dem blauen Kreis lassen sich bestimmte Aktionen auslösen und mit dem blauen Viereck Werte berechnen, sowie Daten ändern. Zudem ist es möglich Beschreibungen und Notizen hinzuzufügen. Auf der linken Seite hat man eine Übersicht über alle verwendeten Regeln, die in einer Ordnerstruktur geschachtelt sind. Auf der rechten Seite sind alle verfügbaren Daten aufgelistet. Diese können auch während dem Erstellen der Regeln angelegt werden.

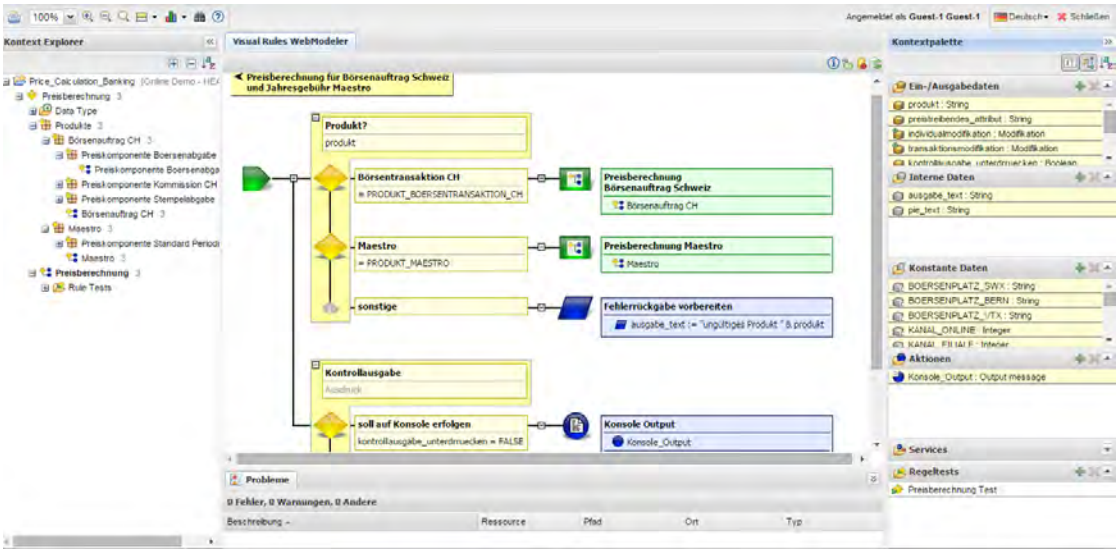

Abbildung 6.1: Beispiel im Visual Rules BRM Editor [17]

Visual Rules BRM verfügt neben der Regelmodellierung auch noch über andere Funktionalitäten. So ist es möglich, dass Entscheidungen in Entscheidungstabellen modelliert werden. Außerdem lassen sich für die Regeln Testfälle definieren und organisieren, sowie Ausführungsstatistiken und Testergebnisse anzeigen. Zusätzlich stehen Analysewerkzeuge zur Verfügung, die Abhängigkeiten zwischen Regeln und Datenmodellen zeigen sollen [18].

6.2 Huginn

Huginn ist eine Open-Source-Software, mit der sich *Agenten* erstellen lassen [19], die beliebige Aktionen ausführen. Zum Beispiel lassen sich Daten im Web sammeln, die dann mit anderen Agenten analysiert werden können. Abbildung 6.2 zeigt eine Übersicht aller von einem Benutzer verwendeten Agenten. Wie zu sehen ist, können die Agenten nach einem Zeitplan eingeteilt werden. Sie werden dann zu einem bestimmten Zeitpunkt oder in festgelegten Abständen automatisch ausgeführt.

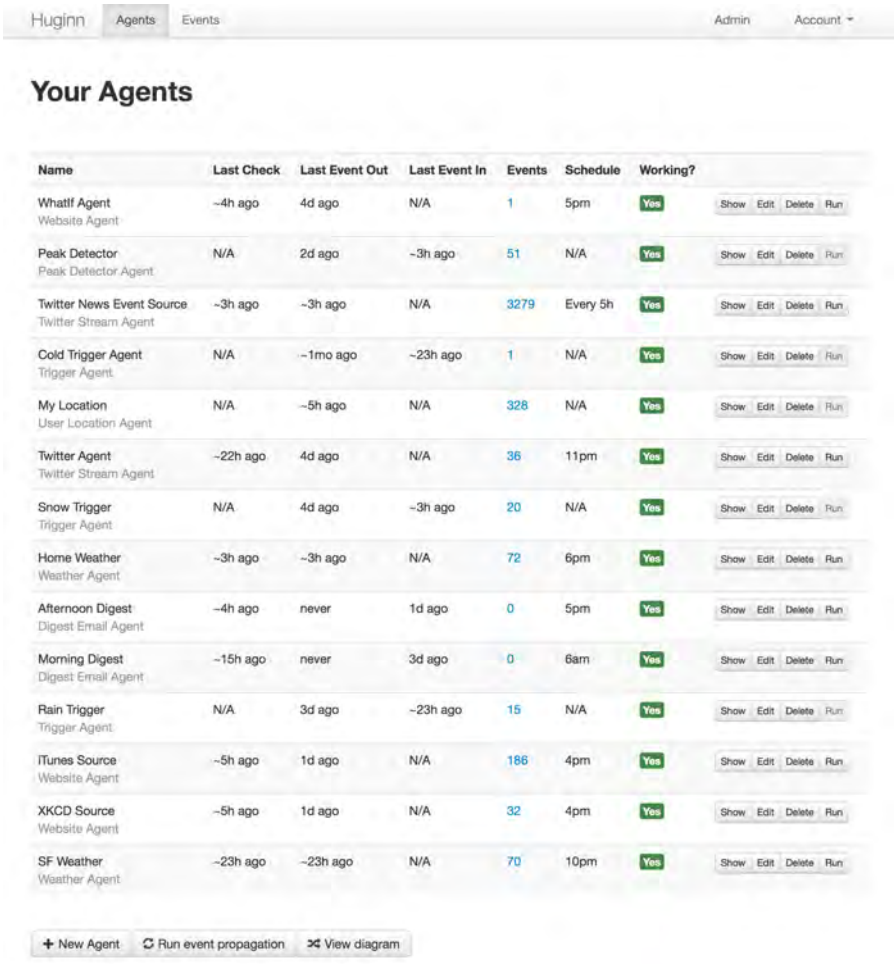

Abbildung 6.2: Huginn - Übersicht erstellter Agenten [19]

Eine Anwendung für Huginn ist zum Beispiel, dass mit einem Agenten *SF Weather* zu einem bestimmten Zeitpunkt, Daten über das Wetter gesammelt werden. Diese Daten werden dann von anderen Agenten nach bestimmten Kriterien, zum Beispiel Regenwahrscheinlichkeit ausgewertet. Ein weiterer Agent schickt zu einem späteren Zeitpunkt eine Mail mit den ausgewerteten Daten an den Benutzer. Die Abläufe beziehungsweise Abhängigkeiten zwischen den verwendeten Agenten sind in Abbildung 6.3 dargestellt. Das Wetterbeispiel ist auf der rechten Seite abgebildet, während auf der linken Seite ein weiteres Beispiel zu sehen ist. Dabei werden Agenten benutzt, um auf Twitter Daten zu analysieren. Fällt ein festgelegter Begriff in kurzer Zeit besonders oft, wird der Benutzer benachrichtigt [19, 20].

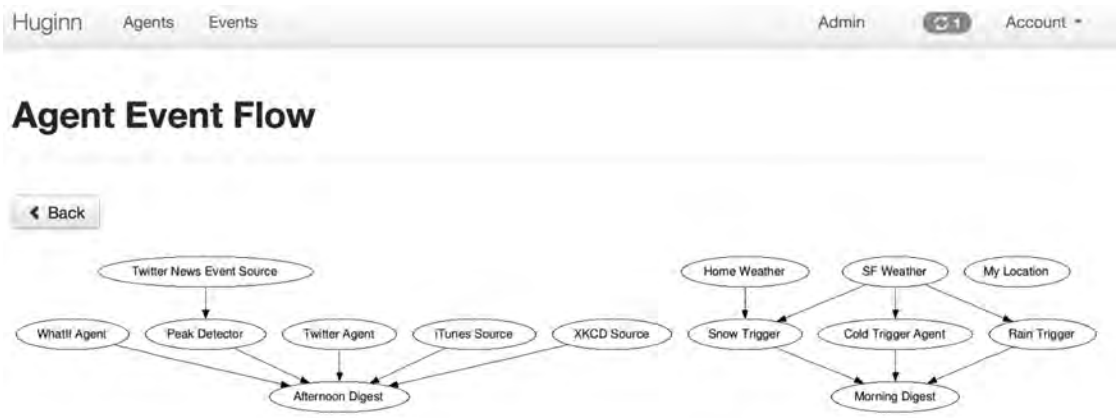

Abbildung 6.3: Huginn - Interaktionen der Agenten [19]

Huginn kann auf eigenen Servern gehostet werden und ist damit auch für Unternehmen zum Analysieren von Daten interessant. Zusätzlich ist es möglich, eigene Agenten zu programmieren, um so die Einsatzmöglichkeiten für einen speziellen Bereich zu erweitern [20].

6.3 Vergleich

Im Vergleich mit den vorgestellten Programmen, lassen sich, mit dem in dieser Arbeit weiterentwickelten Editor, boolesche Regeln einfach mit Drag & Drop als Baum erstellen. Diese Regeln werden bereits bei der Erstellung validiert und können dann zur Analyse von elektronischen Fragebögen verwendet werden. Bei Visual Rules geht es mehr darum Ablaufregeln zu erstellen, um dadurch Geschäftsprozesse zu vereinfachen oder zu automatisieren. Dabei wird mit bereitgestellten Daten gearbeitet. Bei Huginn werden beliebige Daten durch die Agenten selbst gesammelt. Im Unterschied dazu, werden in diesem Editor nur die Datentypen zur Verfügung gestellt, nicht aber die eigentlichen Daten. Die Regel selbst ist durch die Baumform übersichtlich dargestellt. Bei Huginn gibt es zwar eine Übersicht in Baumform, allerdings wird dort nur angezeigt, welche Agenten miteinander verknüpft sind. Visual Rules stellt die Regeln grafisch anschaulich dar. Zusätzlich lassen sich Teile von Regeln auslagern. Außerdem sind, genauso wie für den in dieser Arbeit entwickelten Editor, für die Erstellung von Regeln keine Programmierkenntnisse notwendig. Anders ist dies bei Huginn, dort lassen sich ohne entsprechende Kenntnisse, nur die bereits entwickelten Agenten benutzen.

7

Zusammenfassung

Durch die in dieser Arbeit vorgenommenen Änderungen am Konzept und den damit einhergehenden Änderungen am Editor ist es nun leichter Auswertungsregeln zu entwickeln. Es besteht nun die Möglichkeit eine erstellte Regel in Baumform zu speichern und wieder zu laden. Dies ermöglicht das spätere Bearbeiten und Anpassen der Regeln. Somit sind die funktionalen Anforderungen FA1, FA2 und FA4 aus Abschnitt 3.1 erfüllt. Sollten sich Änderungen an einem Fragebogen ergeben, können nun die passenden Auswertungsregeln schnell geändert und aktualisiert werden.

FA3 und FA5 sind dadurch erfüllt, dass alle Bestandteile einer Regel als Baumknoten umgesetzt sind und die Regel so einfach und übersichtlich als Baum zusammengebaut werden kann. Zudem bietet dies den Vorteil, dass genau festgelegt ist, welche Knoten miteinander verbunden werden können. So können Fehler bereits während der Erstellung der Regeln vermieden werden. Zusätzlich gibt es ein Feedbackfeld, indem

Fehlermeldungen angezeigt werden. Dieses kann allerdings noch verbessert werden, indem detailliertere Texte angezeigt werden. Durch diese Aspekte sind FA6 und FA7, sowie NFA4 ebenfalls erfüllt. Beim Umsetzten des Konzepts wurde außerdem darauf geachtet, dass sich der Editor leicht erweitern lässt und damit die nicht-funktionalen Anforderungen NFA1 und NFA2 erfüllt sind. Außerdem kann das Model um weitere Bestandteile ergänzt werden. Zu beachten ist dabei, dass festgelegt wird, wie die neuen Bestandteile mit den bereits vorhandenen Bestandteilen verbunden werden können. Zum Platzieren der Grafikkomponenten wird Drag & Drop benutzt. Zudem wurde die Bedienbarkeit an einigen Stellen verbessert, zum Beispiel beim Verbinden von zwei Knoten (siehe Anforderung NFA3).

7.1 Ausblick

Für die zukünftige Weiterentwicklung des Editors, sind verschiedene Erweiterungen vorstellbar. Eine sinnvolle Erweiterung wäre beispielsweise, dass eine Regel in Textform eingelesen werden könnte und daraus der passende Regelbaum abgeleitet wird. Dadurch wäre es möglich Regeln, bei denen die Speicherdatei nicht mehr vorhanden ist, ohne großen Aufwand erneut zu generieren. Zusätzlich könnten dadurch per Hand erstellte oder veränderte Regeln im Editor dargestellt werden. Als Voraussetzung müsste die Texteingabe der Regel syntaktisch mit der generierten Textform des Editors übereinstimmen.
Eine weitere denkbare Erweiterung wäre, das Einbinden von bereits existierenden Regeln in den Baum. Das hätte den Vorteil, dass Regelteile ausgelagert und wiederverwendet werden könnten. Dies würde die Übersichtlichkeit, vor allem bei komplexen Regeln, erhöhen. Als Umsetzung könnte ein neuer Baumknoten eingefügt werden, welcher dann die Regel repräsentiert. Ein anderer Ansatz wäre, dass kein extra Knoten die Regel repräsentiert, sondern der Baum der einzufügenden Regel, direkt als Teilbaum eingebunden wird. Um die Übersichtlichkeit zu bewahren, könnte der Editor, um eine neue Funktion zum Ein- und Ausklappen von Baumteilen erweitert werden. Denkbar wäre auch eine Kombination der beiden Ansätze, indem ein neuer Knoten und das Ein- und Ausklappen von Teilbäumen hinzugefügt wird.

Durch die Umsetzung des Konzepts und der damit folgenden Trennung von Model und View ist es vorstellbar, die Grafikkomponenten auszutauschen. Zum Beispiel wäre es denkbar, dass jeder Bestandteil einer Regel eine andere Form und Farbe erhält. Damit könnte ebenfalls die Übersichtlichkeit und Benutzerfreundlichkeit erhöht werden. Zusätzlich könnte der Editor so ergänzt werden, dass Teilbereiche der Regel markiert und verschoben werden können.

Literaturverzeichnis

[1] Gadatsch, A.: Big Data. wisu–das wirtschaftsstudium **41** (2012) 1615–1621

[2] Chamoni, P., Gluchowski, P.D.P.: Integrationstrends bei Business-Intelligence-Systemen. Wirtschaftsinformatik **46** (2004) 119–128

[3] Kang, B., Cho, N.W., Kang, S.H.: Real-time risk measurement for business activity monitoring (BAM). International Journal of Innovative Computing, Information and Control **5** (2009) 3647–3657

[4] Schobel, J., Schickler, M., Pryss, R., Reichert, M.: Process-Driven Data Collection with Smart Mobile Devices. In: 10th International Conference on Web Information Systems and Technologies (Revised Selected Papers). Number 226 in LNBIP. Springer (2015) 347–362

[5] Schobel, J.: QuestionSys - A Generic and Flexible Questionnaire System Enabling Process-Driven Mobile Data Collection. (Website) `https://www.uni-ulm.de/in/iui-dbis/forschung/projekte/questionsys.html`; abgerufen am 26. Januar 2016.

[6] Mertesz, B.: Concept and Implementation of a Rule Component Enabling Automatic Analysis of Process-Aware Questionnaires. Ulm University, Master‘s Thesis (2014)

[7] Schobel, J., Schickler, M., Pryss, R., Maier, F., Reichert, M.: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: 10th Int’l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps. (2014) 371–382

[8] Küchlin, W., Weber, A.: Bäume. In: Einführung in die Informatik. Springer (2000) 291–307

[9] Erk, K., Priese, L.: Theoretische Informatik: Eine umfassende Einführung. Springer-Verlag (2008)

[10] Steger, A.: Diskrete Strukturen: Band 1: Kombinatorik, Graphentheorie, Algebra. Springer-Verlag (2007)

[11] Pepper, P.: Programmieren lernen: eine grundlegende Einführung mit Java. Springer-Verlag (2007)

[12] Ebert, R.: Eclipse RCP-Entwicklung von Desktop-Anwendungen mit der Eclipse Rich Client Platform. Ralf Ebert (2011)

[13] Silva, V.: Practical Eclipse Rich Client Platform Projects. Apress (2009)

[14] Zhang, Y., Luo, Y.: An architecture and implement model for Model-View-Presenter pattern. In: 2010 3rd international conference on computer science and information technology. Volume 8. (2010) 532–536

[15] Potel, M.: MVP: Model-View-Presenter the taligent programming model for C++ and Java. Taligent Inc (1996)

[16] Bosch Software Innovations GmbH: Business Rules Management mit Visual Rules BRM – Regeln intuitiv modellieren, nahtlos integrieren und effizient ausführen. (Website) `https://www.bosch-si.com/de/produkte/business-rules-management-brm/visual-rules.html`; abgerufen am 05. Februar 2016.

[17] Bosch Software Innovations GmbH: Visual Rules BRM Beispiel-Regeln. (Website) `https://www.bosch-si.com/de/produkte/business-rules-management-brm/visual-rules-brm-kostenlose-demoversion/visual-rules-brm-demo.html`; abgerufen am 05. Februar 2016.

[18] Bosch Software Innovations GmbH: Modellierung und Optimierung – Unterschiedliche Regeltypen, ein grafischer Ansatz. (Website) https://www.bosch-si.com/de/produkte/business-rules-management-brm/modellierung-optimierung/modellierung-optimierung.html; abgerufen am 07. Februar 2016.

[19] 2016 GitHub, I.: What is Huginn? (Website) https://github.com/cantino/huginn; abgeufen am 06. Februar 2016.

[20] Rixecker, K.: Huginn: Das kann die Open-Source-Alternative zu IFTTT. (Website) http://t3n.de/news/huginn-ifttt-alternative-open-source-540123/; abgerufen am 06. Februar 2016.

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Lars Miltkau Matrikelnummer: 789732

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Lars Miltkau