Fakultät Wirtschaftsinformatik und Angewandte Informatik

Otto-Friedrich-Universität Bamberg

Prototypische Realisierung eines Systems zum Erwerb und zur Anwendung von Expertenregeln

Philipp Grandl (Matr. Nr. 1678189)
Johannes Hofmann (Matr. Nr. 1576462)
Tim Rütermann (Matr. Nr. 1694792)

Projektbericht

WS 2014/2015

22. Mai 2015

Betreuer: Prof. Dr. Ute Schmid

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Grippewellen gibt es in Deutschland jedes Jahr. Einen Schutz gegen die Influenza bieten Impfungen. Da die Erreger sich ständig verändern, müssen die Impfstoffe nicht nur laufend angepasst werden, sondern auch in ausreichender Menge vorhanden sein. Als sich 2009 die neue Grippe, allgemein bekannter als die 'Schweinegrippe' in Deutschland verbreitete und sich mehr und mehr Menschen impfen ließen, führte dies zu einem Impfstoff-Engpass (C. Hohmann, 2009).
Die herkömmliche Methode der Pharmaindustrie, um die zur Herstellung von Impfstoffen nötigen Proteine mittels Hühnereiern zu gewinnen, dauert jedoch im Notfall zu lange und liefert zu wenig Impfstoff.

Diese Probleme adressiert ein Verfahren das als 'Molecular Farming' bezeichnet wird. Hierbei wird das notwendige Protein aus Pflanzen gewonnen. Im Labor ist diese Methode schon länger bewährt. Um diese Methode auch für die Massenproduktion tauglich zu machen, haben amerikanische Forscher des Fraunhofer Instituts in einem ersten Schritt eine voll-automatisierte Pflanzenfabrik entwickelt. Die dabei verwendeten Tabakpflanzen werden unter optimalen Bedingungen gezüchtet. Unter anderem sind Bewässerung, Nährstoffversorgung, sowie Beleuchtung automatisch auf die Pflanzen angepasst (Fraunhofer-Institut, 2013).

Das optimale Wachstum von Tabakpflanzen ist jedoch auch im kontrollierten Umfeld noch ausbaufähig. Die Pflanzen müssen nach wie vor manuell von Biologen auf deren Gütekriterien analysiert werden. Speziell geht es dabei um einzelne Blätter der Tabakpflanzen und deren Beschaffenheit. Um ein optimales Ergebnis zu erzielen, sollten die Blätter zum Beispiel möglichst groß sein und frei von Verfärbungen, Löchern und weiteren Kriterien, die ein Blatt als Ausschuss klassifizieren würden. Um diese Regeln weiter spezifizieren zu können, arbeiteten deutsche Forscher des Fraunhofer Instituts (EZRT) mit Biologen zusammen. Schlussendlich soll ein System entwickelt werden, welches diese, mittels Machine Learning Verfahren gelernten Expertenregeln zur Qualitätsprüfung, automatisiert. Blätter beziehungsweise ganze Pflanzen die nicht dem Qualitätsanspruch der aufgestellten Expertenregeln genügen, werden entfernt oder ausgetauscht, um eine maximale Auslastung der Fabrik, mit dem Ziel den Wirkungsgrad der bereits automatisierten Impfstoff-Produktion aus Tabakpflanzen weiter zu erhöhen, zu gewährleisten.

Diese Arbeit beschäftigt sich mit einem ersten Versuch der Umsetzung eines solchen Systems zur Evaluation von Pflanzen aufgrund von Expertenregeln. Als Grundlage hierfür wurde die freie Software Rapidminer verwendet, welche mit Funktionalität zur genaueren Analyse verletzter Regeln durch unser eigenes System 'Damage-Control' erweitert wurde.
Im Folgenden gehen wir zuerst auf das Ziel des Projektes ein, bevor das konzeptionelle Design unseres Systems besprochen wird. Danach geht es um die Implementierung von Damage Control und schließlich bildet ein Fazit den Abschluss des Berichts.

Kapitel 2

Ziel des Projekts

Das Ziel des Projekts ist ein Programm für das Frauenhofer-Institut in Fürth. Das Institut forscht daran autonome Fabriken zur Herstellung von bestimmten Mitteln, in diesem Fall speziell ein Impfstoff gegen Malaria, mit Hilfe von Programmen so zu unterstützen, dass diese autark laufen können.

Um völlige Autarkie zu gewährleisten, sollen alle Arbeitsschritte von Maschinen durchgeführt werden. Kein Mensch soll die Anlage betreten müssen. Da Pflanzen aber organische Objekte sind und individuell auf Einflüsse reagieren, aber auch unterschiedlichen Einflüssen ausgesetzt werden, wird es immer zu Abweichungen unter den Pflanzen geben. Auch können selbst Teile einer Pflanze, also einzelne Blätter oder der Stiel, unterschiedlichen Bedingungen ausgesetzt sein und dadurch Anzeichen für schädliche Einflüsse aufzeigen.

Damage-Control ist nicht für die Erhebung von Pflanzen- und Blätterdaten zuständig. Andere Programme beziehen die Werte, messen die Größe, zählen die Anzahl von Löcher, erkennen unterschiedliche Färbungen etc.. Jedoch fehlt diesen Werten noch der Zusammenhang und die Bedeutung im Gesamtprozess der Produktion.

Für diesen Prozess soll Damage-Control eingesetzt werden. Der Benutzer soll in der Lage sein, einer Pflanze mit Hilfe von Regeln Schadenspunkte zuzuweisen. Damage-Control soll dem Benutzer erlauben Regeln zu erstellen, nach denen den Blättern der Pflanzen Schadenspunkte zugewiesen werden sollen. Jede Regel, die die Pflanze erfüllt, kann unterschiedliche Menge von Schadenspunkte geben. Dies soll dafür sorgen, dass besonders schädliche Aspekte auch mehr gewichtet sind und schneller auffallen. Regeln sollen so konzipiert werden, dass die Pflanze bei Erfüllung aller Bedingungen den Schaden zugewiesen bekommt und nicht nur bei einer Bedingung, also einem Teilbereich einer Regel.
Dadurch können die Regeln komplex und prägnant und genau bestimmte Szenarien ansprechen unter denen die Blätter Schaden zugewiesen werden. Diese Informationen, also die Schadenspunkte, die ein Blatt erhält, sind der Kern der Entscheidung, welches Blatt welcher Pflanze welche Schäden hat. Die Schäden können auf Basis der Regeln auch rückwirkend hergeleitet werden, da diese als eine Art Klassifikation zu sehen sind. Das Ziel von Damage-Control ist nicht eine wertende Klassifizierung zu erzeugen, jedoch ist das Programm das Werkzeug,

um eine generalisierte und neutrale Klassifizierung abzuleiten. Diese entsteht durch die Verschmelzung der Informationen über die Schäden einer Pflanze oder eines Blattes und wissenschaftlichen Wissen über die Materie. Zusätzlich sollen diese Regeln auch später noch verändert oder auch gelöscht werden können, denn das konzipieren der Regeln bleibt dem Nutzer selbst überlassen und wird von Damage-Control nicht erzeugt, denn dafür ist tieferes Verständnis der Pflanzen in Zusammenhang mit deren Gesundheit und Umgebung von Nöten. Das Verändern der Regeln ist dauerhaft und kann nur durch ein erneutes Verändern rückgängig gemacht werden. Für den Benutzer ist es ebenso wichtig die Regeln abspeichern und später wieder importieren zu können. Dies ist eine Funktion, deren Bedeutung klar zu erkennen ist. Komplexere Regelstrukturen benötigen Zeit um sie zu konzipieren und zu erstellen, deshalb ist es essentiell sie speichern und laden zu können.

Da auch die Übersichtlichkeit ein Ziel von Damage-Control ist, sollen alle Attribute eines Blattes Kompakt angezeigt werden. Die Schadenspunkte einer Pflanze und deren Blätter nehmen hier eine Sonderrolle ein, denn diese sollen schon durch auf den ersten Blick hervorgehoben werden. Auch deren Ursache soll in moderater Geschwindigkeit angeschaut werden können. Dies ist ein Designziel von Damage-Control, da die Größe eines Datensatzes nicht unerheblich ist und eine falsch konzipierte Darstellung zu einem nicht benutzbaren Programm führen könnte.
Eine besonders wichtige Funktion von Damage-Control soll eine Art Tracking-Möglichkeit sein, die es dem Benutzer erlaubt zu sehen wie die Schadenspunkte eines Blattes zustande gekommen sind. Durch das Tracking können unregelmäßige Einflüsse leichter erkannt werden. Der Schaden eines Blattes spielt dabei eine wesentlich untergeordnete Rolle als die Informationen woher der Schaden kommt, da die Herkunft Aufschluss über die Ursache geben kann. Die Herkunft wird dabei durch die Regeln ausgedrückt, die das Blatt erfüllt hat.

Kapitel 3

Konzept

3.1 Rapidminer extension

Zu Beginn des Projektes mussten neben den gewünschten Funktionen vor allem die verschiedenen Implementierungsmöglichkeiten erörtert werden. Zur Auswahl stand die Realisierung als RapidMiner-Erweiterung oder als Software mit eigener GUI unter Verwendung zentraler RapidMiner-Klassen. Um eine fundierte Entscheidung treffen zu können, mussten Vor- und Nachteile beider Vorgehensweisen und deren Umsetzbarkeit durch die Teammitglieder ausreichend betrachtet werden.

Nach eingehender Recherche hat sich das Team für eine Implementierung mit eigener GUI unter Verwendung von RapidMiner-Klassen und somit gegen eine RapidMiner-Erweiterung entschieden. Diese Implementierungsvariante wurde vom Projektteam favorisiert, da sie den programmiertechnischen Fähigkeiten der Mitglieder entsprach und sich darüber hinaus die Anforderungen des Lastenheftes mit ihr am besten umsetzen ließen. Weiterhin konnte auf eine umfassende Einarbeitung in RapidMiner verzichtet werden, da nur einige wenige RapidMiner-Klassen essentiell für die Realisierung des Projektes benötigt wurden, welche ohnehin nach Belieben verändert werden konnten.

Die Verwendung bestimmter RapidMiner-Klassen zusammen mit einer eigens konzipierten GUI erlaubt Nutzern die Blätter zu klassifizieren ohne dabei auf die RapidMiner Software zurückgreifen zu müssen. Des Weiteren sind die Benutzeroberflächen auf die gewünschten Kernfunktionen ausgerichtet und somit auch leicht zu bedienen. Die Verwendung bestimmter RapidMiner-Klassen erlaubt es jede gewünschte Funktionalität oder Datenstruktur der RapidMiner Software zu verwenden und gegebenenfalls zu Erweitern. Allerdings können bei dieser Variante der Implementierung keine weiteren RapidMiner-Funktionen, wie etwa bei einer RapidMiner Erweiterung, weiterhin genutzt werden. Diese müssten dann entweder vom Frauenhofer-Institut selbst oder in einer weiteren studentischen Projektarbeit hinzugefügt werden.

Als Informationsquelle für eine mögliche RapidMiner Erweiterung wurden das White Paper „How to Extent RapidMiner 5“ (Rapidminer-GmbH, 2012), sowie

das offizielle RapidMiner-Forum (http://forum.rapid-i.com/) herangezogen. Bei einer Erweiterung würde RapidMiner um einen neuen Operator oder neue Datenobjekte erweitert werden, während alle anderen von RapidMiner bereitgestellten Funktionalitäten wie zum Beispiel Machine-Learning Algorithmen weiterhin verfügbar sind. Ferner kann diese Erweiterung innerhalb der RapidMiner-Community wiederverwendet oder verbessert werden. Die Realisierung als RapidMiner-Erweiterung würde sich somit bei einer tatsächlichen Weiterverwendung unserer Projektarbeit im Rahmen der RapidMiner-Community oder des Frauenhofer Instituts als sinnvoll erweisen.

Allerdings geht aus dem Lastenheft für unser Projekt hervor, dass lediglich eine Klassifikation der Blätter anhand von Regeln vorgenommen werden soll, gleichzeitig Machine-Learning Aspekte auf Grund fehlender Kenntnisse der Teammitglieder nicht berücksichtig werden müssen. Weiterhin hat das Team nur in einer kurzen Vorbereitungsphase Erfahrungen mit der RapidMiner Software gesammelt, so dass die Erklärungen im White Paper kaum nachvollzogen werden konnten. Somit wäre eine Implementierung der Software als RapidMiner-Erweiterung mit einem erheblichen Arbeitsaufwand verbunden, welcher unter Umständen zu keinem zufriedenstellenden Ergebnis führt. Somit fällt auch der Aspekt der Weiterverwendung innerhalb der RapidMiner-Community oder des Frauenhofer-Instituts weg, da eine ungenügende Erweiterung schlichtweg nicht weiterverwendet oder -entwickelt werden würde.

3.2 GUI

Um den Nischenplatz von Damage-Control hervorzuheben, ist die Benutzeroberfläche essentiell.
Um den Gesamtumfang gering zu halten und wenig über zusätzliche Datein zu arbeiten, ist die Anleitung der Nutzung des Programs im Program enthalten. Dies führt nur zu einer unwesentlich größeren GUI ohne jedoch deren kompexität zu erhöhen, lässt den Nutzer aber schnell zwischen den Arbeitsschritten die Anleitung betrachten, für was ansonsten sowohl das Öffnen des Dokuments nötig wäre als auch das Finden und Auswählen, falls der Nutzer mehrere Dokumente bereits geöffnet hat.
Für eine Übersichtlichkeit in der Darstellung der Daten auf der einen Seite und den Regeln auf der anderen, muss der Bereich für die Anzeige von Daten eines Datensatzes getrennt von der Erstellung und Bearbeitung von Regeln dargestellt werden. Gleichzeitig ist die Verknüpfung der Regeln zu den Daten zu beachten, da auch jedes Blatt potenziell Regeln verletzen kann und diese auch angezeigt werden sollen. Um das Program weiterhin so kompakt wie möglich und die Trennungen beizubehalten, muss ein Bereich geschaffen werden, der alle anderen Funktionen und Anleitung beinhaltet. Funktionen sind hier zum einen das Importieren eines Datensatzes und Berechnen der Schadenspunkte auf Basis des Datensatzes und der erstellten Regeln. Die Zusammenstellung dieser drei Komponenten (Button für Berechnung, Import eines Datensatzes, Anleitung) im selben Bereich, ist begründet auf der Annahme, dass keine Regeln ohne vorher gegebenen Daten erstellt werden sollen, da sonst keine Aussagen über die beinhalteten Attribute getroffen werden können. Auch soll die Darstellung und Bearbeitung der Regeln so übersichtlich wie möglich gehalten werden, da dieser

Bereich die meisten Funktionen zu bieten hat und somit schon recht voll ist. Alle Funktionen, die dem Nutzer zur Verfügung stehen, können per Button und/oder Mausklick oder einer Kombination ausgewählt werden.

Um keine komplexeren Bereiche in die bis dahin einfach und konsistent gehaltene GUI zu bringen, wird das Erstellen und Bearbeiten von Regeln auf ein neues Fenster ausgelagert. Dieses soll intuitiv benutzbar sein. Deshalb wird ein visuelles Erstellen von Regeln dem Erstellen per If-Bedingungen vorgezogen. Der Benutzer benötigt keinerlei Erfahrung in irgendeiner Programmiersprache, jedoch muss sie wissen, was sie einstellen will. Dabei sind nur drei Dinge zu beachten. Attribut, Vergleichsoperator und Vergleichswert. Die Operatoren folgen simplen mathematischen Regeln und sollen nicht selbst eingetragen, sondern aus einer Liste ausgewählt werden, um die Freiheit des Nutzers nur insofern einzuschränken, dass sie keine sinnlosen Operatoren nutzen kann. Auch die Attribute sollen aus einer Liste ausgewählt werden, um mögliche Schreibfehler und nicht vorhandene Attribute von vorn herein auszuschließen. Das Textfeld, das den Vergleichsoperator beinhalten soll, unterliegt hierbei nur geringen Einschränkungen, da es sowohl nominale als auch Zahlenwerte zum Vergleich gewählt werden können.

Kapitel 4

Implementierung

4.1 Struktur

Das Klassifikations-Programm wurde nach dem MVC-Pattern konzipiert und setzt sich aus drei verschiedenen Hauptkomponenten zusammen. Mit der Wahl dieses Architekturmusters soll sichergestellt werden, dass der Programmentwurf stets flexibel ist und einzelne Komponenten leicht ersetzt werden können. Die MVC-Struktur hat es zudem den Projektmitgliedern erlaubt unabhängig voneinander an Komponenten zu arbeiten und sich dabei auf das jeweilige Aufgabenfeld zu spezialisieren. Dies hat allerdings auch zur Folge, dass fehlerhafte oder verbesserungsbedürftige Komponenten lediglich nur vom jeweiligen Teammitglied effizient bearbeitet werden können. Im Folgenden sollen die das Datenmodell und die Funktionalität implementierenden Klassen MVController und MVCModel erläutert werden.

4.2 Datenmodell

Das Datenmodell, also die MVCModel Klasse des Programms, besteht aus zwei wesentlichen RapidMiner Datenstrukturen, dem ExampleSet und dem RuleModel. Aufgabe des RuleModels ist es die Klassifikationsregeln des Benutzers zu speichern und zu verwalten. Diese Klassifikationsregeln sind Typen des RapidMiner Objektes Rule und besitzen eine Liste des Typs SplitCondition, welche die vom Benutzer definierten Bedingungen repräsentieren(siehe Grafik 1). Die Rule Klasse wurde zum Zwecke der Schadensberechnung um die Klassenvariable damage, sowie den entsprechenden Gettern und Settern erweitert. Dies ermöglicht es vom Nutzer definierte Schadenswerte abzuspeichern und beim Verletzen einer Regel den jeweiligen Schaden aufzurufen.

ExampleSet

Die zweite Datenstruktur ist das ExampleSet und dient der Speicherung der Beispieldaten, also die Attribute und Werte der Pflanzenblätter, in Form einer Tabelle. Diese Tabelle beruht auf einer weiteren RapidMiner Struktur, dem ExampleTable, welche für die Speicherung der Rohdaten zuständig ist. Auf der niedrigsten Abstraktionsebene werden die Daten der Pflanzen als Zahlen in

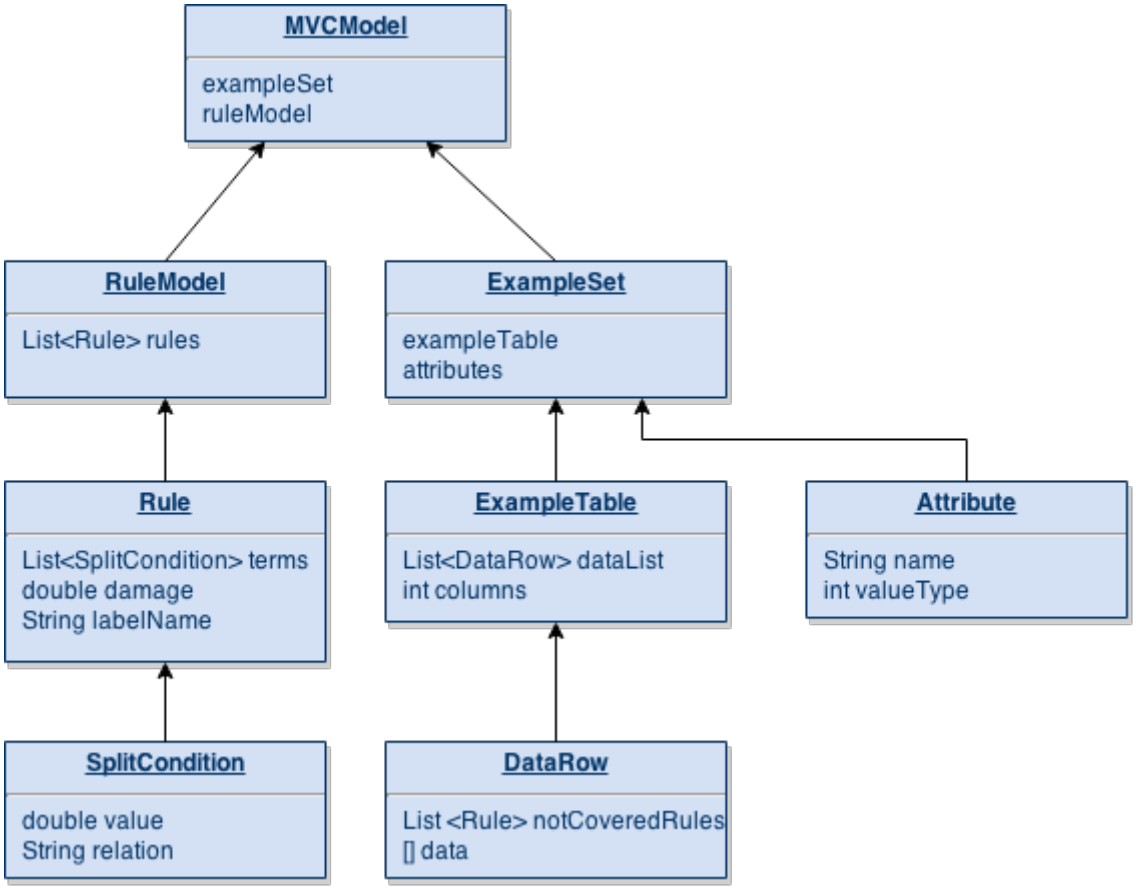

Abbildung 4.1: Datenmodell

Zeilen gespeichert. Das ExampleTable besteht aus mehreren Zeilen des Typs

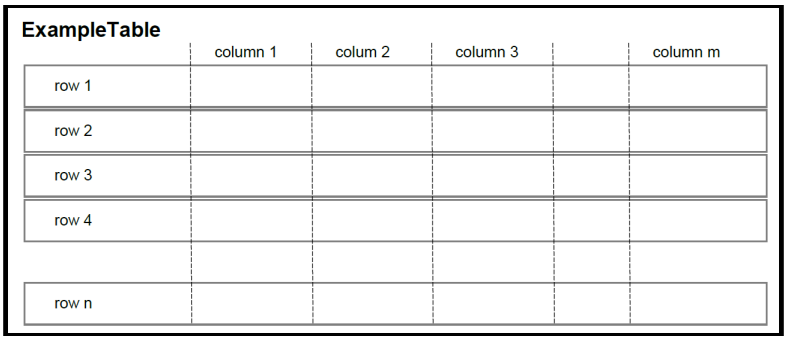

Abbildung 4.2: Aufbau des Example Tables

DataRow, während Spalten nur logisch existieren und durch einen Index repräsentiert werden(siehe Figur 2). Da das ExampleTable samt seinen DataRows für das Abspeichern der Rohdaten zuständig ist, wurde die DataRow Klasse um eine Liste des Typs Rule namens coveredRules erweitert, um verletzte Regeln eines Pflanzenblattes abspeichern zu können. Da das ExampleTable samt seinen DataRows für das Abspeichern der Rohdaten zuständig ist, wurde die DataRow Klasse um eine Liste des Typs Rule namens coveredRules erweitert, in welcher erfüllte Regeln gespeichert werden können.

ExampleTable

In der nächsten Ebene, dem ExampleSet, werden Semantik und Typisierung basierend auf den Daten des ExampleTables bereitgestellt. Im ExampleSet werden die Spalten des Tables als Attribute und die Zeilen als Example Objekte gehandelt. (siehe Figur 3). Wie in Figur 3 zu sehen ist referenzieren Attribute und Examples auf Daten aus dem ExampleTable und stellen somit eine Sicht auf die Daten des ExampleTables bereit. Eine wichtige Funktion der Attribute ist das Auslesen von Tabelleneinträgen mit unterschiedlichen Typen. Da im ExampleTable lediglich Zahlen gespeichert können, müssen String Einträge über ein Mapping-Verfahren in Zahlen umgewandelt werden. Im Attribut wird in der Variable valueType (siehe Grafik 1) der Typ einer ExampleTable Spalte, dies können numerische, nominelle oder zeitformatierte Typen sein, gespeichert. Die Werte eines Examples im ExampleSet werden anhand der verschiedenen Typen der Attribute über das Mapping korrekt ausgelesen und abgespeichert. Somit können auch nominelle Werte von Pflanzen wie etwa Farben korrekt abgespeichert und über das Mapping aufgerufen werden.
Die Methoden des MVCModels sind, wie beim MVC-Pattern üblich, Getter und Setter und stellen die nötigen Funktionen für die Methoden des MVC-Controllers bereit. Da für das Programm die Annahme getroffen wurde, dass ein Eintrag aus Pflanzennummer, Blattnummer und den verschiedenen Blatteigenschaften besteht, wurden die Getter um die Funktion der semantischen Trennung der einzelnen Examples erweitert. Für die Methoden des Controllers wurden die Example-Einträge mehrfach in Listen verschachtelt um den Redundanzen des Beispieldatensatzes entgegenzuwirken. Ziel dieses Vorgehens war

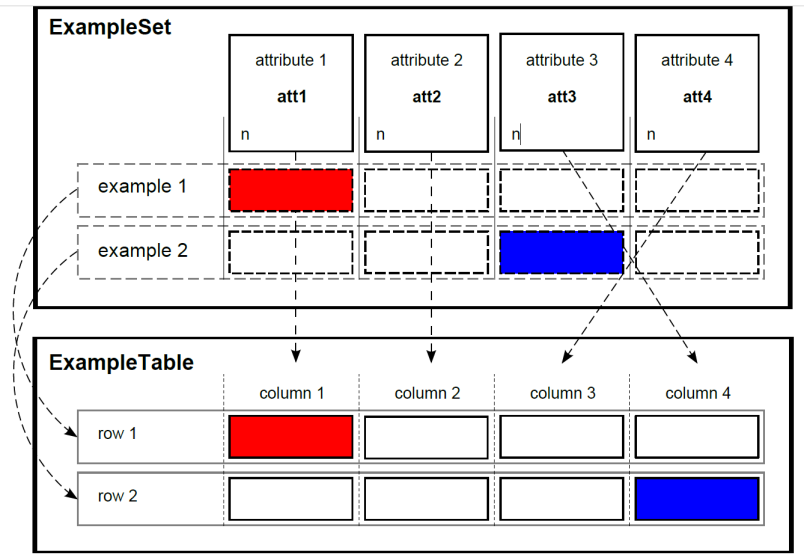

Abbildung 4.3: Verbindung zwischen Example Table und Example Set

es anderen Teammitgliedern, welche an der View oder dem Controller arbeiten sollten und kaum Kenntnisse über das ExampleSet und dessen Examples besaßen, die Implementierung zu erleichtern, indem sie mit ihnen bekannteren Strukturen, wie zum Beispiel Listen, arbeiten können.

4.3 Funktionalität

Die gewünschten Anforderungen des Lastenhefts, sprich die Kernfunktionalitäten des Programms, wurden im MVCController realisiert. Der Controller wird in der main-Methode initialisiert und kann auf das Model zugreifen und durch Methoden dessen Daten verändern und neu abspeichern. Des Weiteren implementiert er die in der MVCView definierten JavaFX Elemente mit Programmlogik und ist somit das Bindeglied zwischen GUI und dem Datenmodell. Die folgenden Abschnitte sollen deutlich machen wie die gewünschten Funktionen implementiert und welche Annahmen dabei getroffen wurden.

Importieren eines Beispieldatensatzes

Für das Importieren von .csv Dateien hat die Projektgruppe die RapidMiner Klasse CSVFileReader benutzt, welche aus .csv Dateieinträgen ein ExampleSet mit den entsprechenden Attributen generieren kann. Um Schadenswerte für Pflanzen speichern zu können, wurde das ExampleSet, welches die Pflanzendaten in Tabellenform repräsentiert, um das numerische Attribut „Schaden“ erweitert. Hierfür wurde die RapidMiner Klasse CSVFileReader um die Methode createExampleSetWithDamage() erweitert. Hierbei werden die ausgelesenen Metadaten, also Attribute, samt Namen und Typ in einer Liste gespeichert und um das numerische Attribut „Schaden“ erweitert. Anschließend wird ein ExampleTable Objekt aus der Attributs Liste und einem Reader, welcher über

die Zeileneinträge der .csv Datei iteriert, im Arbeitsspeicher erzeugt. Im letzten Schritt wird aus dem ExampleTable ein ExampleSet erzeugt und im Model des Controllers gespeichert. Anschließend wird die Methode clearDamage() des Model aufgerufen um alle Einträge des Attributs „Schaden“ auf 0.0 zu setzen, da diese auf Grund fehlender Einträge bei der Erzeugung des ExampleSets mit „NaN“ gefüllt wurden und semantisch keinen Sinn ergeben.

Erzeugen von Regeln

Nachdem ein Panel über die Methode openAddRuleWindow() erzeugt wurde, kann der Benutzer verschiedene Parameter für eine Regel definieren. Hierbei können durch die Funktion addCondition() neue JavaFX Objekte, sogenannte Nodes, im Panel erzeugt werden, welche aus zwei Combo-Boxen und einem Text-Field bestehen und die Parameter für eine Bedingung der Regel repräsentieren. Mit der finishAddingRule() wird ein Rule Objekt erzeugt, welches mit den vom Benutzer definierten Eingaben Damage, Name und verschiedener Bedingungen parametrisiert und im Datenmodell abgespeichert wird.

```
List splitConditionList
Boolean correctOperators
for each node

if node.Attribute is numerical then

     if parameters not null/value is string/Attribute is numerical then

SplitConditionList add numerical splitCondition(node.attribute,
    node.relation, node.value)

else
correctOperators = false
break

   else
           if parameters not null/value is not a string/Attribute is
               nominal then

splitConditionList add nominal splitCondition(node.attribut,
    node.relation, node.value)
else
correctOperators = false
break

If correctOperators = true then

     create Rule with splitConditionList

     model add the new rule
else

   Open dialog with errormessage
```

Die Methode durchläuft alle Node Objekte des RuleWindows mit einer for-Schleife und speichert die Werte der zwei ComboBoxen und des Textfields als Strings in einer Liste, wobei das erste Element das Attribut, das zweite die Relation und das dritte der Wert einer Bedingung oder auch Terms ist. Jeder Schleifendurchgang soll ein SplitCondition Objekt erzeugen, welches dem Rule Objekt hinzugefügt werden kann. Hierbei werden die gewählten Attributs Namen auf ihren Typ im Model überprüft, da nur für nominelle Attribute die Relation ‚=' und für numerische Attribute die Relationen ‚¿' und ‚=¡' zulässig für die jeweiligen SplitCondition Objekte sind. Des Weiteren wird der Wert eines Terms anhand eines ReGex Ausdrucks auf Zahlen beziehungsweise Strings überprüft. Sollte die Überprüfung fehlschlagen wird die Variable correctOperators auf false gesetzt und die Schleife verlassen. Nur wenn correctOperators true ist wird eine Regel nach durchlaufen aller Nodes im Model gespeichert, während bei false ein Dialogfenster mit den nötigen Nutzerinformationen geöffnet wird.

Editieren und Löschen von Regeln

In der GUI ausgewählte Regeln werden anhand ihres Namens mit der Methode deleteRuleByLabel(String) aus dem Model gelöscht. Beim Editieren von Regeln, wird das openAddRuleWindow aufgerufen und automatisch mit den entsprechenden Werten wie damage, name und den Bedingungen geladen. Durch Klick auf den finish-Button wird die ausgewählte Regel mit den veränderten Bedinungen oder damage editiert, es sei denn, der Nutzer hat den Namen der Regel geändert. Eine Namensänderung führt zur Beibehaltung der ursprünglich ausgewählten Regel und zur einfachen Erzeugung einer neuen Regel. Dieses Verhalten ist beabsichtigt und soll es Nutzern möglich machen Regeln mit leichten Veränderungen zu duplizieren.

Exportieren von Regeln

Da die RapidMiner API Methoden für das Importieren von .csv Dateien und das Exportieren von ExampleSets bereitstellt hat sich das Projektteam dazu entschieden, die Regeln in ein ExampleSet umzuwandeln und im .csv Format zu speichern. Das ExampleSet besitzt die nominellen Attribute Rule, Attribute, Relation, sowie die numerischen Attribute Value und Damage. Die Examples des ExampleSets stellen die Parameter der SplitCondition sowie den Namen der zugehörigen Rule bereit. Die Umwandlung und das Schreiben der Regeln werden mit Hilfe der hierfür konzipierten Klasse RuleListWriter realisiert. Der Konstruktor dieser Klasse erhält eine Liste von Regeln und wandelt diese in die entsprechenden Attribute um. Mit dem Aufruf der writeRuleListToFile(String path) Methode wird ein ExampleSet erzeugt und in eine .csv Datei geschrieben. Hierbei wird ein ExampleTable mit den Attributen erzeugt und jede SplitCondition jeder Regel zusammen mit dem Regelnamen als String Array gespeichert. Anschließend wird mit Hilfe der DataRowFactory unter Berücksichtigung der Attribute des ExampleTables DataRow Objekte erzeugt und dem ExampleTable hinzugefügt. Mit der Methode createExampleSet() des ExampleTables wird ein ExampleSet erzeugt und anschließend über die writeCSV() der CSVExampleSetWriter Klasse der RapidMiner API in eine csv. Datei geschrieben.

Importieren von Regeln

Das Importieren von Regeln funktioniert prinzipiell genauso wie der Import eines Datensatzes von Pflanzen, da beide in Tabellenform aufgebaut sind. Somit müssen nur noch die Examples des generierten ExampleSets in SplitConditions und anschließend in Rules umgewandelt werden. Diese Aufgabe übernimmt die Klasse RuleExampleSetReader, welche mit Hilfe der statischen Methode getRulesFromExampleSet(), eine Liste von Rules aus einem ExampleSet generiert. Im ersten Schritt durchläuft eine for-Schleife alle Example des ExampleSets und wandelt die Zeilenwerte in eine Liste des Typs Strings um. Diese Liste beinhaltet die Parameter der einzelnen SplitConditions sowie den Namen der zugehörigen Regel. Anhand dieser Parameter werden Rule Objekte erzeugt und in einer Liste gespeichert, welche nachdem alle Examples durchlaufen wurden, ans Datenmodell übergeben wird.

Klassifikation der Datensätze

Bei der Klassifikation, beziehungsweise der Schadensberechnung der Pflanzen werden alle Examples des ExampleSets auf alle Regeln des Models überprüft. Die RapidMiner-API bietet hierzu die Methode cover() der Rule Klasse an, welche ein Example anhand aller SplitConditions dieser Rule überprüft und true zurückgibt, wenn diese erfüllt werden. Wie im Codebeispiel in Figur 7 zu sehen ist, werden nach einem Nullcheck die Damage-Einträge der Examples mit clearDamage() auf 0 gesetzt, damit die Werte aus alten Klassifikationen bereinigt werden.

Ablauf der runClassification() Methode

```
if (model and rules not null) then

open dialog with errormessage

else

clear damage of all examples

   for each Example in model

      for each Rule in model

      if Rule covers Example then

         add damage of Rule to Example

         add Rule to List of Covered Rules from Example
```

Anschließend werden alle Examples des ExampleSets durchlaufen und für jedes Example nochmal die Liste der Regeln iteriert. Wenn ein Example eine Regel erfüllt, wird der letzte Zeileneintrag des Examples, welcher den Damage-Wert beinhaltet, um den Damage-Wert der Rule erhöht. Um erfüllte Regeln eines Blattes in der GUI anzeigen lassen zu können, werden die erfüllten Regeln in

der DataRow des Examples abgespeichert. Hierbei ist zu beachten, dass Regeln, selbst nachdem Sie verändert oder gelöscht werden, noch immer unter den erfüllten Regeln eines Blattes aufgeführt werden können. Aus Usability Gründen sollte es dem Nutzer jederzeit möglich sein die Ergebnisse der letzten Klassifikation zu betrachten, solange diese nicht erneut durchgeführt wurde. Daher hat sich das Projektteam dazu entschieden, dass die Liste der erfüllten Regeln eines Examples nur über eine erneute Klassifikation manipuliert werden kann.

4.4 GUI

Welcome-Tab

Für die Benutzeroberfläche wurder der GUI-Builder JavaFX verwendet. Das-Gesamtfenster beinhaltet drei TabPanes. Zu Vorderst ist die Startseite von Damage-Control und beinhaltet sowohl eine Anleitung in Form einer uneditierbaren Textpane als auch zwei Buttons, die verknüpft sind mit der Import-Methode einer CSV-Datei und der Methode zur Berechnung der Schadenswerte mit Hilfe der Regeln.

Rule-Tab

Der zweite Tab beinhaltet den Bereich der Regeln. Von dort aus werden alle Funktionen die zur Manipulation von Regeln möglich sind ausgeführt. Dazu gehören das Erstellen, Editieren, Importieren sowie Exportieren und Löschen von Regeln. Diese Funktionen sind mittels Buttons anwählbar und führen die jeweilige Aktion aus. Bei ungültigen Aktionen wird der Nutzer mittels eines Popup Dialoges welches Erklärungen enthält informiert. Für das Erstellen und Editieren von Regeln öffnet sich jeweils ein neues Fenster. Solange man Regeln hinzufügt oder editiert ist der Rest des Programmes nicht parallel nutztbar. Im Add/Edit Fenster können dann die Regeln mit Name, Damage und ihren Conditions definiert werden. Hat man dies getan und mit Finish bestätigt, schließt sich das Fenster wieder und die Regeln werden im Rule-Tab in einer Liste angezeigt. Nun können weiter Aktionen wie Löschen oder Exportieren der Regeln ausgeführt werden.

Plant-Tab

Der letzte Tab beinhaltet die Darstellung der importierten und ausgewerteten Daten. Zur Darstellung der Pflanzen in Kombination mit ihren Blättern wurde eine TreeView verwendet. Die update-Methode im MVCController benutzt zwei verschachtelte For-Schleifen um die TreeView zu füllen. Dabei wird zuerst ein Rootelement erstellt, dem alle Pflanzen hinzugefügt werden können. Da diese Rootelement nicht Teil der Daten ist, wird ist dieses unsichtbar gesetzt. Dadurch alle Pflanzen werden auf der selben Ebene angezeigt. Per getPlants() werden alle Pflanzen ohne Duplikate geholt und mit Schadenspunkten versehen. Diese setzen sich aus den Schadenspunkten aller Blätter einer Pflanze zusammen. Der Name der Pflanze wird der TreeView in der der ersten For-Schleife als String hinzugefügt. Die zweite For-Schleife fügt jeder Pflanze die Namen ihrer Blätter hinzu. Diese Blätter sehen als Rootelement die jeweilige Pflanze und sind des-

halb eine Ebene tiefer dargestellt.
Der Name wurde dabei verändert um in unverwechselbar zu machen. Dieser setzt sich zum einen aus dem Namen der Pflanze, auf der anderen Seite dem Namen des Blattes zusammen, dadurch wird die Usability durch das leichtere Erkennen der Zugehörigkeit verbessert. Bei langen Listen von Blättern kann unter Umständen das Root-Element eines Blattes, also die Pflanze, leichter übersehen werden und somit weiß der Betrachter nicht, wo sie sich gerade befindet und betrachtet fälschlicherweise ein Blatt einer anderen Pflanze. Durch die Kombination aus Pflanze- und Blattname ist zu jedem Zeitpunkt klar, wo sie sich gerade in der Liste befindet. Blatter haben hinter ihrem Namen einen Doublewert, der dem Schaden entspricht, den das Blatt gesammelt hat. Mit getDamageOfLeaf, die den Namen der Pflanze und des Balttes erwartet, wird der Schadne bezogen. Der Schadenswert wird somit zwei Mal angezeigt. Für den schnellen Überblick in der ListView und ein zweites mal in der Tabelle der Attribute.

Die restlichen Attribute einer Pflanze werden Tabelle angezeigt. Eine Tabelle bietet eine schnell und übersichtliche Möglichkeit eine große Menge an Daten kompakt darzustellen. Diese enthält zum einen den Namen alle Attribute und zum anderen den Wert des Attributs. Zum Füllen der Tabelle wird zwei observableArrayLists für Werte und für Attribute erstellt und gefüllt. Für die Anzeige in der Tabelle muss ein Objekt der "LeafKlasse entsprechen und aus einem Attribut und einem Wert bestehen und wird der einer Liste des Typs Leaf hinzugefügt. Diese wird der Tabelle übergeben und der komplette Inhalt wird angezeigt.

Sobald der Benutzer die ListView anklickt, wird ein Mousevent ausgelöst. Dieses überprüft ob entweder ein Blatt, eine Pflanze oder ein leerer Bereich ausgewählt worden ist. Leere Bereich werden ignoriert. Falls eine Pflanze ausgewählt worden ist, erkennbar am Inhalt des ausgewählten Strings, wird die Tabelle mit leerem Inhalt überschrieben um keine Verwirrung zu erzeugen. Wenn zuvor ein Blatt ausgewählt worden ist und Attribute und Werte anzeigt wurden und der Nutzer nun etwas anderes auswählt, würde es den Benutzer nur irritieren, wenn die Tabelle weiterhin gefüllt ist, obwohl der Objekt keine Attribute und Werte enthalten kann. Diese Objekte sind in diesem Fall nur Pflanzen und somit kann das über in den Inhalt des Strings in der Liste gehandhabt werden. Wenn ein Blatt ausgewählt wurde, wird der String ausgelesen und da sich dieser unter anderem aus der Namen der Pflanze und des Blattes zusammensetzt, kann das Blatt gesucht und dessen Werte geholt werden. Die Regeln, die ein Blatt erfüllt und für die es Schaden bekommt, werden ebenso über den Namen der Pflanze und Blatt geholt und in einer ListeView angezeigt. Die angezeigte Reihenfolge ist dabei die Reihenfolge der Einträge in der Liste für die Regeln, denen das Blatt entsprochen hat.

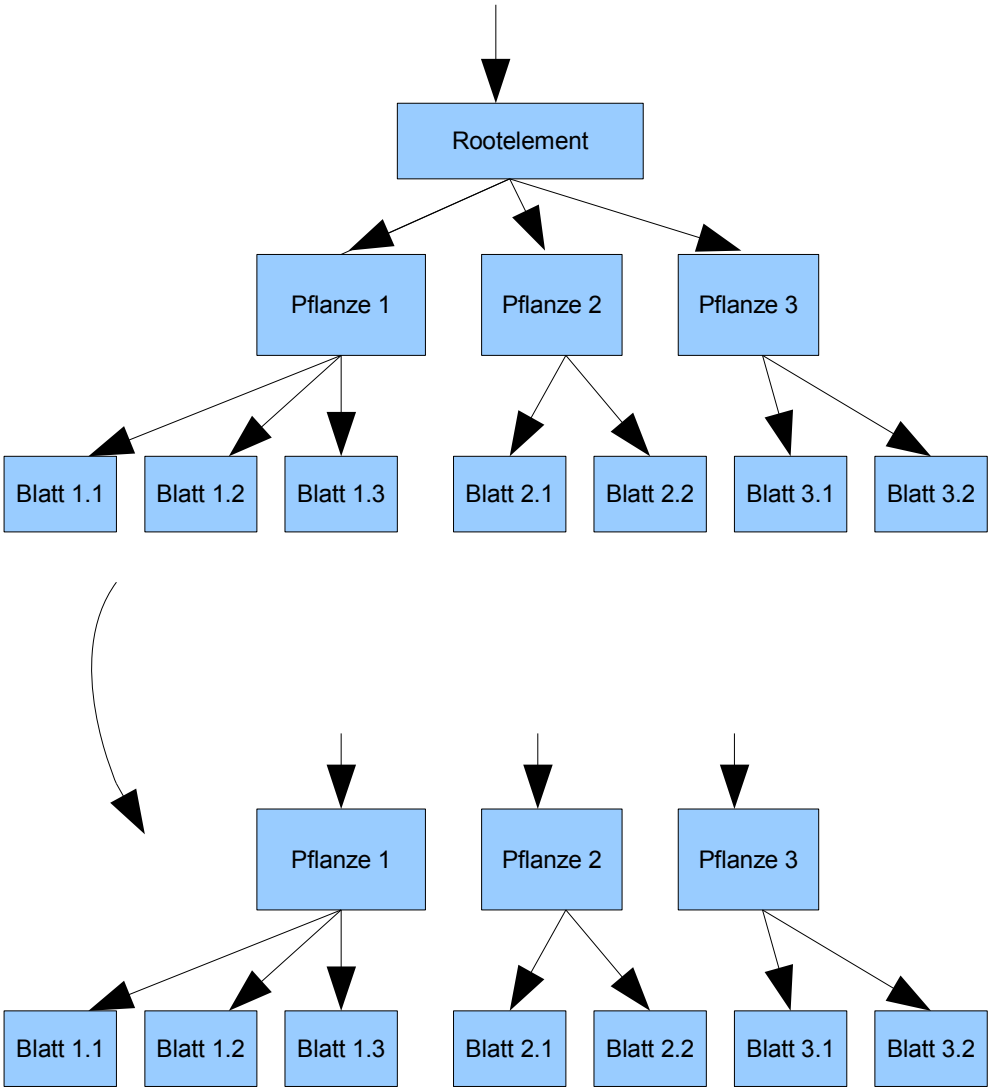

Abbildung 4.4: Struktur der TreeView

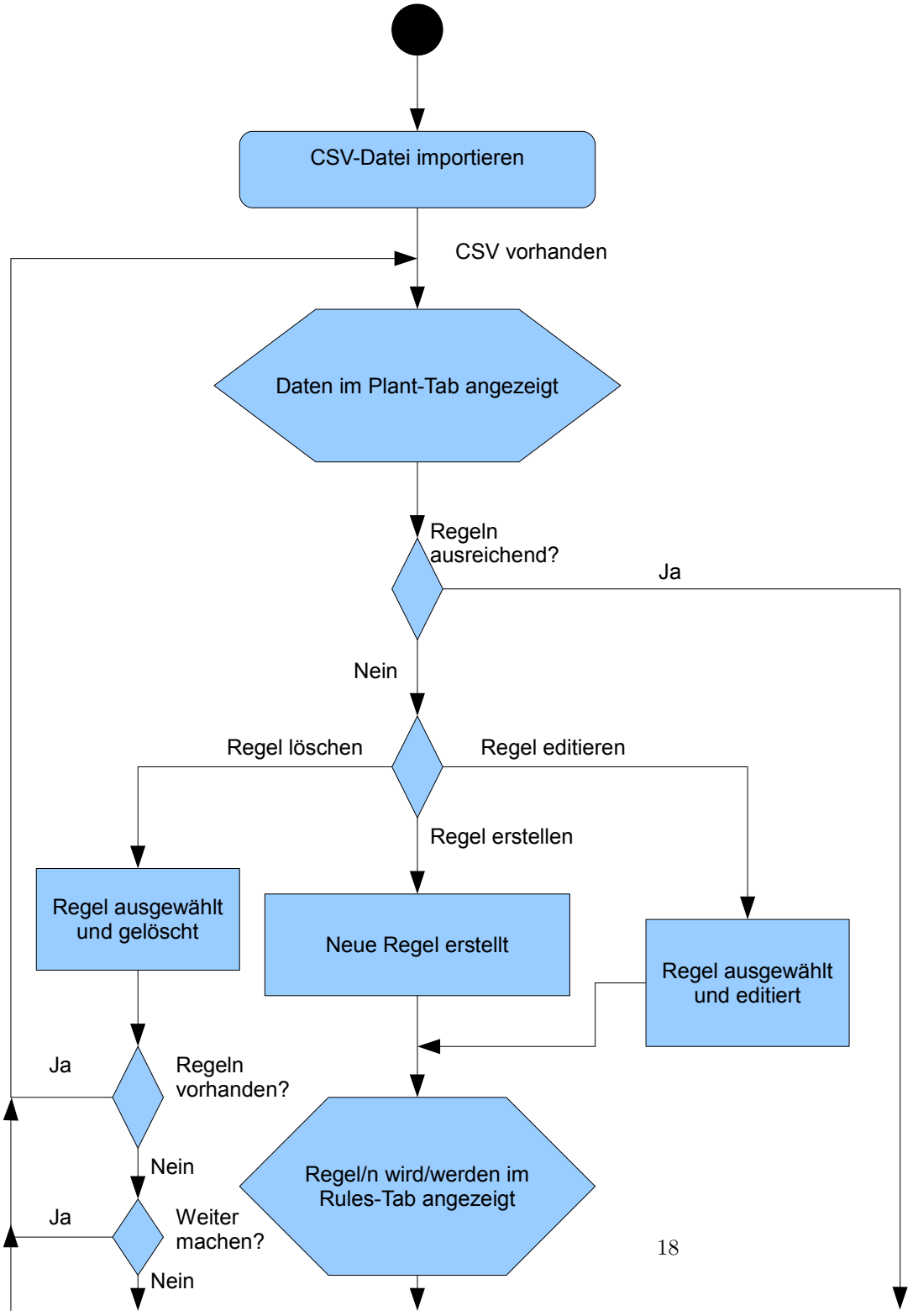

Abbildung 4.5: Ablaufdiagramm der GUI Seite 1

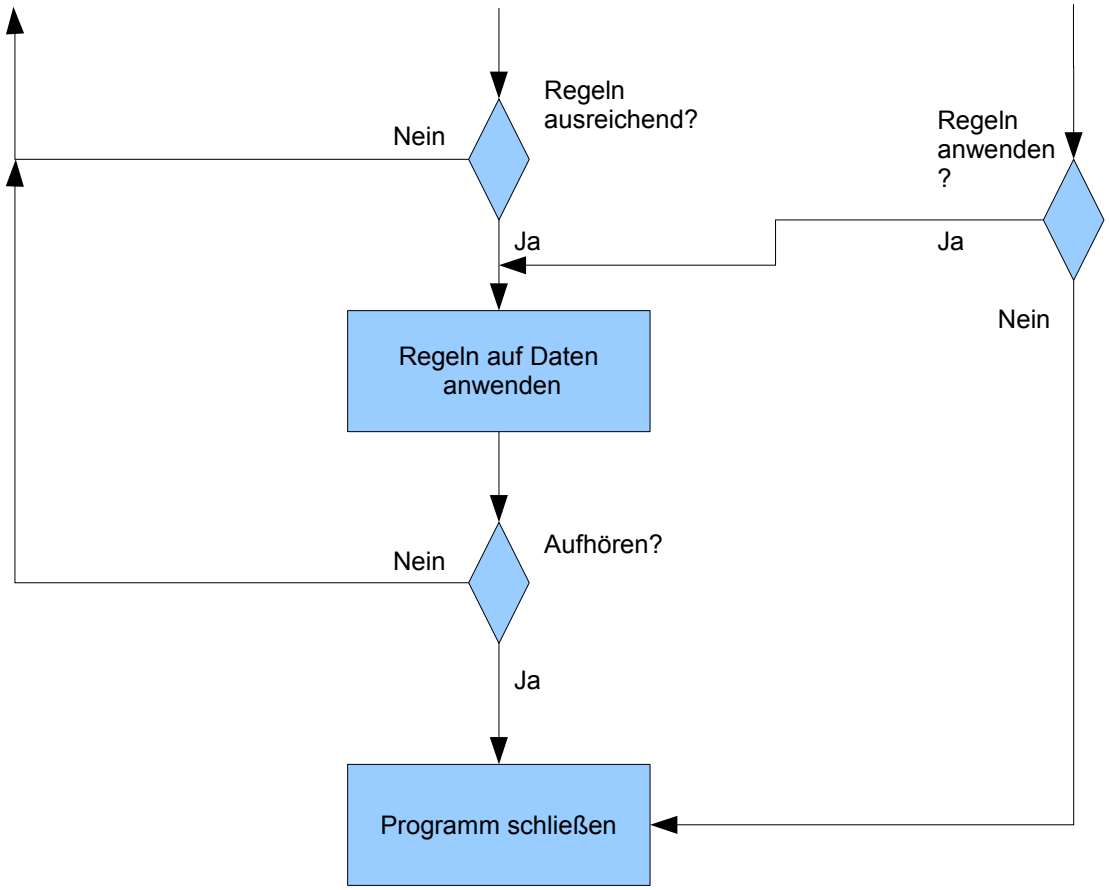

Abbildung 4.6: Ablaufdiagramm der GUI Seite 2

Kapitel 5

Fazit

Damage-Control ist eine Prototypische Realisierung eines Systems zum Erwerb und zur Anwedung von Expertenregeln. Es stellt einen ersten Schritt zur selbständigen Analyse von Schäden an Pflanzen beziehungsweise anderen Objekten, je nach Datensatz dar. Aktuell ist das Regelsystem noch statisch und muss per Nutzereingaben definiert werden.
Da Damage-Control auf Basis von Klassen der freien Software Rapidminer entwickelt wurde, kann Damage-Control beliebig erweitert werden. Dies ist insofern interessant, da das Ziel ganz am Ende ja eine vollautomatisierte Fabrik zur Herstellung von Impfstoff aus Tabakpflanzen ist. Für die Zukunft ist dadurch eine Erweiterung von Damage-Control mit Machine-Learning Methoden möglich, was einen weiteren Schritt auf dem Weg zur autonomen, vollautomatisierten Fabrik darstellen würde.

Literaturverzeichnis

C. Hohmann, S. S. (2009). Pandemieimpfstoff - engpässe in der versorgung. http://www.pharmazeutische-zeitung.de/index.php?id=31631. Last checked on: 2015-05-22.

Fraunhofer-Institut (2013). Forschung kompakt - sonderausgabe 06—2013. https://www.fraunhofer.de/content/dam/zv/de/presse-medien/2013/Juni/Forschung_Kompakt_Sonderausgabe/Sonderausgabe_Preise_2013_ForschungKompakt.pdf. Last checked on: 2015-05-22.

Rapidminer-GmbH (2012). How to extend rapidminer 5. https://rapidminer.com/wp-content/uploads/2013/10/How-to-Extend-RapidMiner-5.pdf. Last checked on: 2015-05-22.