Fachbereich Informatik und Medien

BACHELORARBEIT

Repräsentation medizinischen Wissens mit Drools am Beispiel der Adipositas-Leitlinie

Vorgelegt von: Jonas Preckwinkel

am: 20.09.2012

zum Erlangen des akademischen Grades

BACHELOR OF SCIENCE

(B. Sc.)

Erstgutachter: Prof. Dr.-Ing. Jochen Heinsohn

Zweitgutachter: Dipl.-Inf. Ingo Boersch

Vielen Dank

an meine Familie, die mich während meines Studiums durchgehend unterstützt hat und besonders lieben Dank an meine Mutter Doris, für die aufbauenden Briefe und das nächtliche Korrekturlesen der Arbeit.

Danke auch an meine beiden Betreuer Prof. Heinsohn und Herr Boersch, ohne die diese Arbeit nicht zustande gekommen wäre.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Durch die in den letzten Jahrzehnten schnell fortschreitende Entwicklung in der Medizin und Forschung und die damit verbundene Entstehung von neuem Fachwissen und immer komplexeren Systemen, werden Ansätze zur Lösung gebraucht. Einer dieser Ansätze ist die Entwicklung von Expertensystemen. In Expertensystemen können Fachexperten, die eventuell nicht immer zur Verfügung stehen, ihr Wissen der breiten Masse zur Verfügung stellen. In Krankenhäusern in Bangladesh und anderen Regionen, herrscht ein permanenter Mangel an Fachärzten. Es werden immer neue Ärzte ausgebildet, die allerdings in den Armenregionen Indiens nicht lange verweilen, da ihre Gehälter in einer großen Stadt und Privat-Klinik um ein Vielfaches höher sein können. Um den speziellen Fall der Behandlung von Kinderkrankheiten ohne Fachpersonal zu lösen, wurde IMCI (Integrated Management of Childhood Illness[1]) eingesetzt. IMCI setzt das Fachwissen von Ärzten in Regeln um, denen das medizinische Personal Schritt für Schritt folgen kann, um eine fachgerechte Diagnose zu stellen. [12]
Durch den schnellen Fortschritt in der Medizin wird es auch schwerer für das medizinische Personal, sich immer die neuesten Techniken und Behandlungsmethoden anzueignen. Ein aktuell gehaltenes Expertensystem kann in diesem Fall das Personal bei Entscheidungen unterstützen.

1.2 Zielsetzung

Die Rule Engine Drools soll in dieser Arbeit auf die Verarbeitung und Repräsentation von Wissen untersucht werden, um herauszufinden, ob sich Drools für regelbasierte Wissensverarbeitung in medizinischen Anwendungen eignet. Dazu soll die Leitlinie zur Prävention und Therapie von Adipositas in einem wissensbasierten Beispielsystem modelliert werden. Des Weiteren soll ein Benutzerhandbuch für Droolsanfänger entstehen, das exemplarisch die

[1] IMCI setzt kein Informationstechnisches System um, sondern ist ein schriftliches „Regelwerk"

ersten Schritte in der Programmierung von drools zeigt, um das wissensbasierte System an einen anderen Anwendungsbereich anzupassen.

1.3 Aufbau der Arbeit

Im ersten Abschnitt werden die Grundlagen eines regelbasierten Systems erläutert. Was Regeln und Fakten sind, sowie die bekannten Verarbeitungsmöglichkeiten und Konfliktlösungsstrategien von Regeln, werden dem Leser mit Beispielen näher gebracht. In Kapitel 3 wird die Rule-Engine Drools vorgestellt. Hier wird wieder auf Verarbeitungsmöglichkeiten und Konfliktlösungsstrategien eingegangen, jedoch unter dem Aspekt wie sie in Drools umgesetzt werden. Des Weiteren werden .drl Dateien vorgestellt und ihre zum Erstellen benötigte Syntax. Im darauf folgenden Kapitel wird die Umsetzung der Modellierung der Adipositas Leitlinie mit Drools durchgeführt. Anschließend werden die nötigen Schritte und verwendeten Werkzeuge zum Erstellen einer lauffähigen Drools Rule-Engine dargelegt, sowie die Funktionen der entstandenen GUI[2] vorgestellt. Kapitel 6 ist ein Handbuch, das Benutzern von Drools mit keiner bis wenig Erfahrung Schritte aufzeigt, die nötig sind, um einen Anwendungsbereich zu modellieren. Zum Abschluss wird die einleitende Fragestellung geklärt, sowie die nötigen Schritte um aus der modellierten Adipositas Leitlinie ein einsetzbares System zu entwickeln.

[2] Graphical User Interface

2 Regelbasierte Systeme

Ein regelbasiertes System ist ein Softwaresystem, in dem die Geschäftslogik von der Programmlogik abgekoppelt ist. Als Geschäftslogik wird der Teil des Programms bezeichnet, der die Verarbeitung der Daten steuert, während die Programmlogik den Ablauf des Programms steuert. Dadurch, dass die Geschäftslogik (im regelbasierten System durch Regeln ausgedrückt) getrennt ist, lässt sich diese anpassen, ohne dass Änderungen am Programm-Code vorgenommen werden müssen. Ein weiterer Vorteil ist, dass Regeln auch mit nur wenig Informatikwissen formuliert werden können. So können Experten aus einem bestimmten Fachbereich die Regeln für ihre Software selbstständig anpassen und erweitern.

Ein regelbasiertes System besteht aus einer Faktenbasis, Regelbasis und einem Regelinterpretierer (Inferenzmaschine). Die Faktenbasis und Regelbasis ergeben zusammen die Wissensbasis und stellen damit die Anwendungsdomäne dar. Der Regelinterpretierer bestimmt mithilfe des Pattern Matcher die anwendbaren Regeln. Diese werden mithilfe von Konfliktlösungsstrategien (Section 2.4) sortiert in eine Liste geschrieben, von der dann die erste Regel angewendet wird.

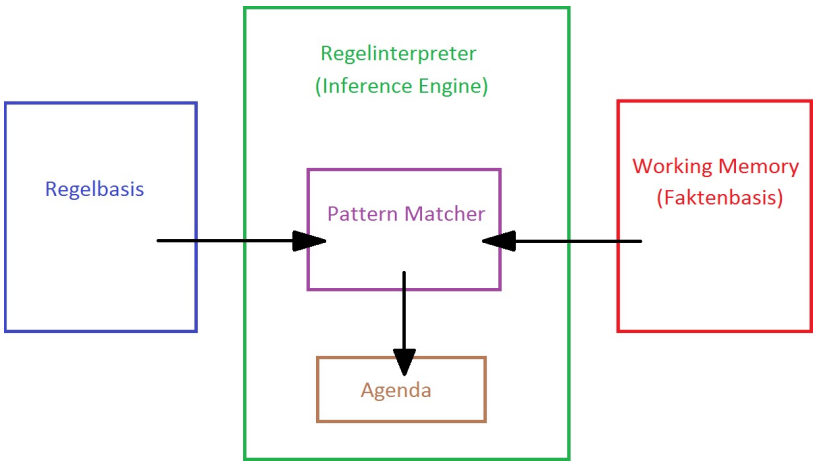

Abbildung 2.1: Regelbasiertes System

2.1 Fakten und Regeln

Regeln stehen als Sammlung von Wissen in einer Regelbasis. Der Aufbau einer Regel ist immer gleich. Sie haben einen Bedingungsteil „wenn", auch Left Hand Side genannt (LHS) und einen Folgerungsteil „dann" der Right Hand Side genannt wird (RHS). Eine solche Regel könnte so aussehen:

Wenn die Straße nass ist und die Temperatur unter 0 Grad, *dann* herrscht Rutschgefahr.

Wenn die Banane gelb ist, *dann* ist sie reif.

Bei der Interpretation von Regeln gibt es zwei unterschiedliche Arten. Bei der Vorwärtsverkettung wird geprüft, ob die Bedingungen *„Die Straße ist nass"* und *„Die Temperatur ist unter 0 Grad"* wahr sind, woraus dann *„Es herrscht Rutschgefahr*" geschlossen werden kann. Mehr dazu im Abschnitt Vorwärtsverkettung. Die Rückwärtsverkettung hingegen stellt eine Frage auf *„Herrscht Rutschgefahr?"* und versucht diese dann anhand von vorhandenen Fakten zu beantworten. Näheres dazu im Abschnitt der Rückwärtsverkettung. Dadurch dass die Menschen Konditionalsätze häufig intuitiv benutzen, um Handlungsanweisungen oder Prognosen auszudrücken, ist es einfach für den Benutzer, sein Wissen in Form von Regeln darzustellen [1].

Fakten sind fest definiertes Wissen, welches der Regelinterpretierer dazu verwendet, den Bedingungsteil der Regeln zu überprüfen. Ein Faktum wäre zum Beispiel: „die Straße ist nass", oder „die Banane ist gelb". Das dem Regelsystem zur Verfügung stehende Faktenwissen liegt in der Faktenbasis. Regeln werden dazu verwendet, um neue Fakten zu erschließen: *„Wenn die Banane gelb ist, dann ist sie reif."* (Deduktionsregel) oder eine Aktion auszuführen *„Wenn die Banane reif ist, dann verspeise sie.*" (Aktionsregel).

2.2 Rückwärtsverkettung

Wie im Abschnitt 2.1 schon erwähnt, beginnt die Rückwärtsverkettung von Regeln mit dem Aufstellen einer Frage / Hypothese, die dann anhand von Fakten beantwortet bzw. bewiesen werden soll. Gehen wir von folgender Regelbasis aus:

R1: Wenn die Straße nass ist und die Temperatur unter 0 Grad, dann herrscht Rutschgefahr.

R2: Wenn Rutschgefahr herrscht, dann müssen Autos langsam fahren.

Bei dem Beispiel könnte die Hypothese *„Autos müssen langsam fahren"* gestellt werden. Der einfachste Fall wäre, wenn sie als Faktum in der Faktenbasis wäre, wodurch die Hypothese

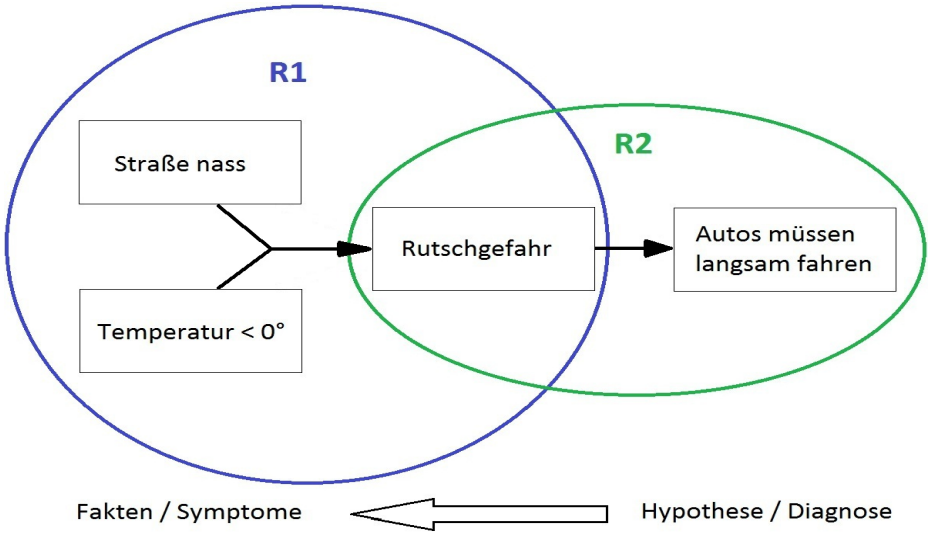

Abbildung 2.2: Rückwärtsverkettung

direkt belegt werden könnte. Ist dem nicht so, so sucht die Rückwärtsverkettung nach einer Regel bei der die RHS, also der Folgerungsteil, der Hypothese entspricht. In unserem Fall wäre das *R2*, mit der sich eine neue Hypothese *„Es herrscht Rutschgefahr"* aufstellen lässt. An dem Punkt geht der Prozess wieder von vorne los (Rekursion), solange bis entweder die Ausgangshypothese belegt wurde oder bis die Suche bei den Fakten endet, die nur noch vom Benutzer erfragt werden können (z.B. Messdaten oder Symptome). Diese Regelinterpretation hat den Vorteil, dass nur die Fakten vom Benutzer erfragt werden müssen, die das Problem lösen können.

2.3 Vorwärtsverkettung

Im Gegensatz zur Rückwärtsverkettung, die zielgetrieben vorgeht und sich von der Hypothese zu den Fakten / Symptomen des Problems durcharbeitet, geht die Vorwärtsverkettung von den Fakten aus zur Schlussfolgerung. Diese Art der Interpretation hat kein festes Ziel und endet daher erst, wenn alle Möglichkeiten ausgeschöpft wurden, neue Fakten zu erschließen. Dazu wird eine der Regeln ausgewählt, die „feuerbereit" sind, das heißt deren Bedingungen alle zutreffen. Deren RHS wird ausgeführt, man sagt „sie feuert". In unserem Autobeispiel können wir von einer Faktenbasis ausgehen, in der wir *„Die Straße ist nass"* und *„Die Temperatur ist unter 0 Grad"* zu gelten haben. Somit ist unsere *R1* feuerbereit und ein neues Faktum kann erschlossen werden *„Es herrscht Rutschgefahr"*. Daraufhin wird *R2* feuerbereit. Dadurch, dass die Fakten, die *R1* feuerbereit machen, noch in der Faktenbasis enthalten sind, haben wir jetzt eine Konfliktsituation. Mehrere Regeln sind feuerbereit und es muss

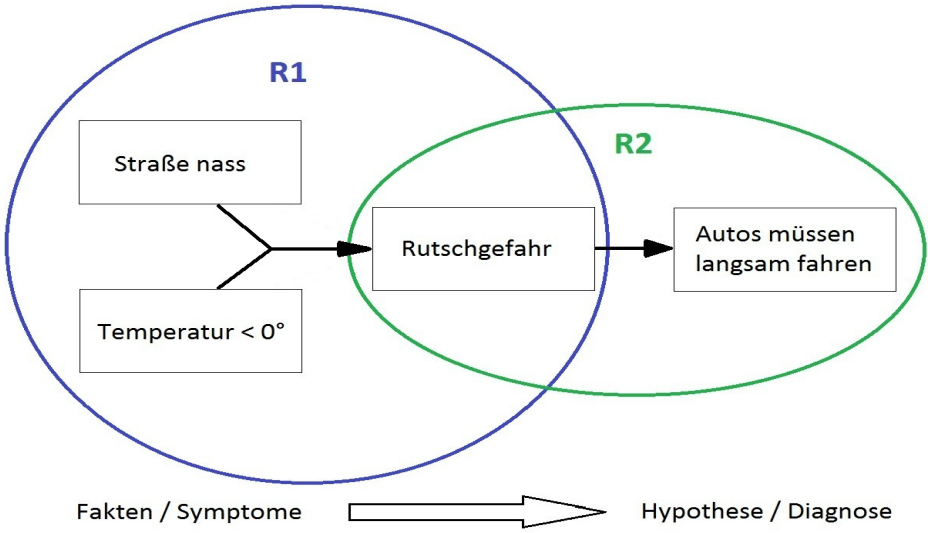

Abbildung 2.3: Vorwärtsverkettung

festgelegt werden, welche als nächste feuern darf. Dazu gibt es Konfliktlösungsstrategien, die im nächsten Abschnitt näher beschrieben werden.

2.4 Konfliktlösungsstrategien

Konfliktlösungsstrategien werden dazu verwendet, um Regelkonflikte aufzulösen. Regelkonflikte treten auf, wenn mehrere Regeln feuerbereit sind. Die Menge aller feuerbereiten Regeln wird Konfliktmenge genannt, und die Auswahl der Regel, die feuern darf, wird Konfliktlösung genannt. Hauptsächlich finden Konfliktlösungsstrategien Anwendung bei der Vorwärtsverkettung. Dazu gibt es eine Reihe von möglichen Strategien, die anwendbar sind.

Refraktionsstrategie

Die Refraktionsstrategie besagt, dass eine Regelinstanz nicht zweimal feuern darf. In unserem Beispiel im Abschnitt 2.3 hatten wir den Fall, dass *R1* nach dem ersten Feuern mit der neu dazugekommenen *R2* die Konfliktmenge bildete. Unendliche Wiederholungen von *R1* könnten die Folge sein. Das wird mit der Refraktionsstrategie verhindert, sodass in der Konfliktmenge nur noch *R2* vorhanden ist und somit als nächstes abgefeuert wird.

Prioritätsstrategie

Mit der Prioritätsstrategie bleiben die Regeln mit der höchsten Priorität in der Konfliktmenge übrig. Dadurch dass es auch Regeln mit gleicher Priorität geben kann, kommt es vor, dass

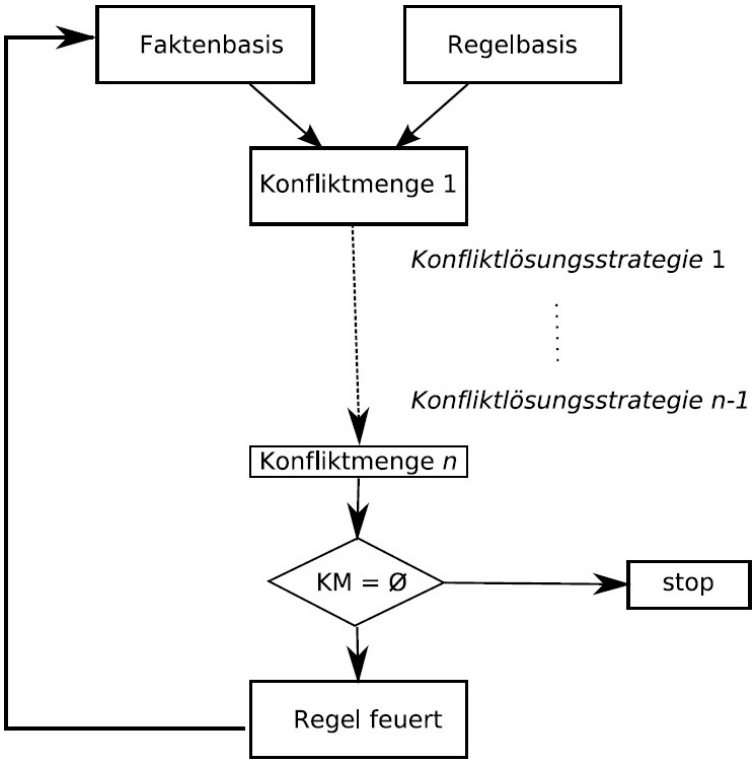

Abbildung 2.4: Vorwärtsverkettung mit Konfliktlösung [2]

nach Anwendung der Prioritätsstrategie keine eindeutige Auswahl getroffen werden konnte. Da können dann entweder andere Strategien greifen oder es muss per Zufall eine der noch vorhandenen Regeln ausgewählt werden.

Spezifitätsstrategie

Die Spezifität lässt sich am besten an einem weiteren Beispiel darlegen. Haben wir die folgenden zwei Regeln:

R1: Wenn Hans in der Mathearbeit eine gute Note hat, dann bekommt er Fischstäbchen mit Nudeln.

R2: Wenn Hans in der Mathearbeit und im Deutschaufsatz gute Noten hat, dann bekommt er eine Schwarzwälder Kirschtorte.

Im Fall, dass beide Regeln im Konflikt stehen, würde die Spezifitätsstrategie *R2* auswählen, da die Menge der Bedingungen von *R1* auch in *R2* enthalten sind. Die Definition lautet also: Lösche eine Regel r aus der Konfliktmenge, wenn diese eine spezifischere Regel r' enthält.

Aktualitätsstrategie

Bei der Aktualitätsstrategie merkt sich das Regelsystem zu welchem Zeitpunkt eine Regel durch neu dazu gekommene Fakten der Konfliktmenge hinzugefügt wurde. Dadurch lässt sich in jeder Konfliktmenge die aktuellste Regel bestimmen.

2.5 Zusammenfassung

Regelbasierte Systeme bestehen aus drei Komponenten. Die Faktenbasis und Regelbasis, die zusammen die Wissensbasis darstellen. Hier steht das Wissen, das den Anwendungsbereich beschreibt. Die dritte Komponente ist der Regelinterpretierer, der durch das Anwenden von Konfliktlösungsstrategien die Verarbeitung von Regeln steuert.
Bei der Regelinterpretation wird zwischen Rückwärts- und Vorwärtsverkettung unterschieden. Die Rückwärtsverkettung beginnt damit, eine Hypothese aufzustellen und versucht diese anhand von Fakten zu belegen. Dazu sucht sie nach einer Regel, deren Folgerungsteil der Hypothese entspricht. Die Bedingungen dieser Regel wiederum werden als neue Hypothese aufgestellt. So hangelt sich die Rückwärtsverkettung durch die Regeln, bis alle relevanten Fakten überprüft worden sind und eventuell fehlende Fakten oder Symptome vom Benutzer erfragt wurden. Bei der Vorwärtsverkettung werden die Regeln gewählt, deren Bedingungen durch die Faktenbasis belegt werden können. Daraufhin wird der Folgerungsteil bei einer Deduktionsregel als neues Faktum in die Faktenbasis eingetragen oder bei einer Aktionsregel ausgeführt. Durch die vorwärtsverkettende Interpretation werden alle Möglichkeiten, neue Fakten zu erschließen, ausgeschöpft. Erst dann endet der Prozess.
Strategien zur Konfliktlösung werden hauptsächlich bei der Vorwärtsverkettung eingesetzt, dadurch dass entschieden werden muss, welche Regel als nächstes feuern darf. Eine Möglichkeit der Konfliktlösung ist die Refraktionsstrategie. Sie verhindert das mehrmalige Feuern einer Regelinstanz, um Schleifen zu verhindern. Die Prioritätsstrategie erlaubt es, jeder Regel eine Priorität zuzuordnen, sodass die mit der höchsten Priorität ausgewählt werden kann und feuern darf. Wenn eine Regel nur einen Teil der Bedingungen einer anderen Regel hat, so lässt die Spezifitätsstrategie nur die spezifischere Regel feuern. Als letzte Möglichkeit zum Steuern der Regelverarbeitung gibt es die Aktualitätsstrategie. Hier wird die Regel zum Feuern gebracht, deren Bedingungsteil sich zuletzt als wahr herausgestellt hat.

Wer sich mit dem Thema „Regelbasierte Systeme" weitergehend beschäftigen möchte, findet in *Eine Einführung in die Künstliche Intelligenz [2]* detailliertere Beispiele und Übungsaufgaben.

3 Drools

3.1 Einleitung

Das Drools Projekt wurde im Jahr 2001 auf SourceForge[1] gestartet. 2005 wurde das Projekt von JBoss übernommen und sollte den Namen JBoss Rules annehmen, der aber nach zwei Jahren wieder verworfen wurde, da die User es immer noch Drools nannten. Bis heute wird Drools von JBoss weiterentwickelt, die aktuelle Version 5.4.0 wurde im Mai diesen Jahres (2012) veröffentlicht. Drools steht unter der Apache Software Lizenz v2 (ASL2). Das ist eine Freie Software Lizenz, das heißt die Software darf genutzt, untersucht, verändert und verbreitet werden unter Übereinstimmung der ASL2[7].

Drools ist eine in Java programmierte Rule Engine. Ein Framework, das, in einer Anwendung integriert, Funktionen zum Verwalten von Geschäftsregeln bereitstellt. Rule Engines finden Nutzen in technischen Anlagen bei der Überwachung von Messwerten oder zum Identifizieren defekter Komponenten. Ein weiteres typisches Einsatzgebiet ist die Lieferung von Diagnose- und Therapievorschlägen in medizinischen Anwendungen. In diesem Kapitel wird kurz auf Drools Möglichkeiten zur Vorwärts- und Rückwärtsverkettung eingegangen. Daraufhin werden die Grundlagen zum Erstellen von Drools Rule Files eingeführt, außerdem wie die bekannten Konfliktlösungsstrategien mit den Möglichkeiten von Drools umzusetzen sind. Der Rete Algorithmus wird dem Leser näher gebracht und erklärt, welche Rolle er in Drools spielt. Zuletzt werden Domain Specific Languages (DSLs) vorgestellt.

3.2 Drools Rule File

Drools Rule Files sind einfache Textdateien mit der Endung .drl. In ihnen wird das Regelwissen, sowie Funktionen, Queries und selbst neue Typen definiert. Aber auch Imports und Packaging wie man sie aus Java kennt sind möglich. Ein genereller Aufbau einer .drl Datei sieht so aus:

[1] SourceForge ist ein Source Code Repository

package *package-name*
imports
globals
functions
queries
rules

Jeder dieser Teile ist optional, außerdem ist die Reihenfolge der einzelnen Teile nicht fest vorgegeben, mit Ausnahme der *package* Deklaration, die ganz am Anfang stehen muss, wenn sie denn genutzt wird.

3.2.1 *import* Statement

Wenn man in seiner Datei Funktionen oder Typen von Java-Klassen verwenden möchte, dann kann man diese per *import* Statement importieren. Klassen, die dem angegebenen Package und `java.lang` angehören, werden automatisch importiert.

```
import java.util.Date;
```

3.2.2 Funktionen

Funktionen werden dazu benutzt, in den Regeln häufig vorkommende Berechnungen auszuführen. Ein Vorteil ist, dass man die Logik in Form von Regeln und Funktionen so an einem gemeinsamen Platz verwalten kann und diese jederzeit anpassbar ist. Eine Funktionsdeklaration sieht zum Beispiel so aus:

```
function int malZwei(int wert){
   return wert*2;
}
```

3.2.3 Typ Deklarationen

Typ Deklarationen sollen dem Benutzer die Möglichkeit geben, sein Datenmodell direkt der Rule Engine zu übergeben, ohne diese vorher in Java implementieren zu müssen oder um einem schon bestehenden Datenmodell die Typen hinzuzufügen, mit denen während des Folgerungsprozesses gearbeitet wird. Neue Datentypen können mit dem Schlüsselwort *declare* definiert werden, gefolgt von den Attributen und dem keyword *end.*

```
declare Person
   name : String
   geburtsJahr : Date
end
```

Die in den .drl Files deklarierten Typen werden Fakt Typen genannt, da ihundre hauptsächliche Aufgabe ist, als Fakten von den Regeln gehandelt zu werden. Zur Kompilierung der .drl Files generiert Drools zu jedem Fakt Typ eine Java-Klasse. Für das oben genannte Beispiel sähe die Klasse folgendermaßen aus[3]:

```
public class Person implements Serializable {
   private String name;
   private java.util.Date geburtsJahr;

   //einen leeren Konstruktor
   public Person(){}

   //einen Konstruktor mit allen Attributen
   public Person(String name, java.util.Date geburtsJahr){...}

   //einen Konstruktor mit allen Schlüsselattributen
   public Person(...){...}

   //setter und getter
   //equals/hashcode
   //toString
}
```

Attribute können durch ein *@key* hinter der Deklaration als Schlüsselattribute festgelegt werden. Damit lässt sich ein weiterer Konstruktor mit allen Schlüsselattributen als Parameter in der Java-Klasse erzeugen.

```
declare Person
   name : String @key
   geburtsJahr : Date
end
```

Drools unterstützt Vererbung für deklarierte Typen und von importierten Java Klassen. Um von einer importierten Klasse zu erben, wird eine Typ Deklaration mit dem Namen der importierten Klasse ohne Attributdeklarationen gebraucht.

```
import meineKlasse.Person

declare Person
end

declare Student extends Person
```

```
    hochschule : String
end
```

3.2.4 Regeln

Regeln lösen eine Reihe von Aktionen aus, wenn die durch die LHS bestimmten Zustände eintreffen, welche auf der RHS festgelegt sind. In der Drools Dokumentation wird die Frage geklärt, warum der Bedingungsteil mit dem Schlüsselwort *when* beginnt statt mit *if*:

> «„Why use „when" instead of „if"?" „When" was chosen over „if" because „if" is normally part of a procedural execution flow, where, at a specific point in time, a condition is to be checked. In contrast, „when" indicates that the condition evaluation is not tied to a specific evaluation sequence or point in time, but that it happens continually, at any time during the life time of the engine; whenever the condition is met, the actions are executed.» [3]

Eine Regel hat stets die gleiche Struktur:

```
rule "name"
    ATTRIBUTE
    when
        LHS - Bedingungen
    then
        RHS - Folgerung
end
```

Der Name der Regel kann frei gewählt werden, er muss nur einzigartig im zugehörigen Rule-Package sein.

3.2.4.1 Left Hand Side (LHS) - Bedingungsteil

Hier sind die Bedingungen formuliert, die zur Laufzeit gegen die Fakten abgeglichen werden. Sie können einfach hintereinander geschrieben werden, übersichtshalber aber besser pro Bedingung eine neue Zeile. Zwischen Bedingungen muss nichts stehen, da ein logisches `"AND"` impliziert wird. Ein Beispiel für die Bedingungen *„Es gibt einen Kuchen"* und *„Es gibt eine Person die Hunger hat"* ist:

```
rule "Wenn es einen Kuchen gibt und eine Person mit Hunger, die jünger als 30 ist,
     dann..."
    when
        exists Kuchen()
        Person(hatHunger == true && alter < 30)
    then
```

```
    ...
end
```

Es wird überprüft, ob es ein Faktum vom Typ `Kuchen` und eines vom Typ `Person` mit dem auf `true` gesetzten Attribut `hatHunger` im Working Memory gibt. Es können alle aus Java bekannten Vergleichsoperatoren genutzt werden, die einen booleschen Wert liefern wie: „`> >= < <= == !=`". Mehrere Attribute lassen sich per „`&&`" oder „`||`" logisch verknüpfen. Fakten oder Attribute können auch an Variablen gebunden werden. Sinn dahinter ist, dass die Variablen auf der RHS erreichbar sind. Möchte man die Personen, die Hunger haben und jünger als 30 sind, an eine Variable `$p` binden so schreibt man:

```
rule "Wenn es einen Kuchen gibt und eine Person mit Hunger, die jünger als 30 ist,
    dann..."
   when
      exists Kuchen()
      $p : Person(hatHunger == true && alter < 30)
           /*das '$' Zeichen ist nicht nötig, aber dient der besseren Erkennung von
               Variablen, die auf der LHS gebunden wurden*/
   then
      ...
end
```

„`exists`" ist eins von vielen Bedingungselementen. Während „`exists`" zu „`true`" evaluiert, wenn es mindestens ein Element des angegebenen Typs im Working Memory gibt, so evaluiert das Bedingungselement „`not`" zu „`false`" in demselben Fall.

3.2.4.2 Right Hand Side (RHS) - Folgerungsteil

Im Folgerungsteil werden alle möglichen Aktionen ausgeführt. Anders als im Bedingungsteil wird hier die normale Java Syntax verwendet. Die Aktionen, die ausgeführt werden, sind das Modifizieren, Einfügen und Löschen von Fakten im Working Memory. Dazu stehen folgende Methoden zur Verfügung:

`insert(Object o)`: Wird verwendet, um ein neues Faktum dem Working Memory hinzuzufügen.
`retract(Object o)`: Wird verwendet, um ein Fakt aus dem Working Memory zu löschen.
`update(Object o)`: Wenn man ein Faktum, welches man auf der LHS an eine Variable gebunden hat, auf der RHS modifiziert hat, so kann man mithilfe dieser Methode der Rule Engine sagen, dass die Regeln neu evaluiert werden müssen.
`modify(Object o){...}`: Macht im Grunde dasselbe wie die update-Methode mit dem

Unterschied, dass man das zu updatende Objekt in den geschweiften Klammern mit Setter-Aufrufen modifizieren kann.

Einige Beispiele für Verwendungsmöglichkeiten:

```
rule "Wenn kein Kuchen da ist, dann erstelle einen Kuchen"
   when
      not Kuchen()
   then
      insert(new Kuchen());
end

rule "Wenn es einen Kuchen gibt und eine Person mit Hunger, die jünger als 30 ist,
    dann hat die Person keinen Hunger mehr und lösche den Kuchen"
   when
      $k : Kuchen()
      $p : Person(hatHunger == true && alter < 30)
   then
      retract($k);

      $p.setHatHunger(false);
      update($p);

      modify($p){setHatHunger(false)} //hat denselben Effekt wie der Setter + Update
end
```

3.2.4.3 Regel Attribute

Attribute ermöglichen es, das Verhalten vom Regelfluss zu verändern. Mit Attributen wird auch die Konfliktlösung in Drools gesteuert.

`no-loop`
Typ: boolean
default Wert: `false`

Mit dem Attribut `no-loop` wird verhindert, dass die Konsequenz einer Regel dieselbe Regel durch Verändern der Fakten wieder aktivieren kann.

`salience`
Typ: integer
default Wert: `0`

Mit dem Attribut `salience` werden Regeln Prioritäten zugewiesen. Es entspricht also dem Prinzip der Prioritätsstrategie. Regeln mit einem höheren Wert werden anderen Regeln vorgezogen. Auch negative `salience` Werte lassen sich festlegen.

`activation-group`

Typ: String

default Wert : `N/A`

Aus einer Menge von Regeln, die einer `activation-group` angehören, kann immer nur eine Regel feuern. Sobald die erste Regel gefeuert hat, werden alle anderen Regeln dieser Gruppe, die feuerbereit sind, verworfen.

`agenda-group`

Typ: String

default Wert: `MAIN`

Das `agenda-group` Attribut ermöglicht es, eine Gruppe zu erstellen, deren Regeln nur feuern dürfen, wenn der sogenannte Fokus auf dieser Gruppe liegt. Außerdem werden andere Regeln, die nicht der Agenda Gruppe angehören, daran gehindert zu feuern. Dadurch lässt sich eine bessere Kontrolle über die auszuführenden Regeln erlangen. Damit eine Gruppe den Fokus erlangt, kann der Fokus durch einen Methodenaufruf im Konsequenzteil einer Regel `(drools.setFocus("agenda-group"))` gesetzt werden. Dieser Aufruf pusht die Agenda Gruppe auf einen Stack. Sobald alle aktivierten Regeln dieser Gruppe gefeuert haben, fliegt die Gruppe vom Stack und die nächste bekommt den Fokus.

`ruleflow-group`

Typ: String

default Wert: `N/A`

Ruleflow Gruppen ähneln den Agenda Gruppen. Wenn der Fokus auf sie gesetzt ist, feuern wieder nur die Regeln dieser Gruppe. Es gibt nur keinen Stack von Ruleflow Gruppen, der abgearbeitet wird.

`lock-on-active`

Typ: boolean

default Wert: `false`

Das „`lock-on-active`" Attribut kann Regeln gegeben werden, die einer Ruleflow oder Agenda Gruppe angehören. Diese Regeln können nur einmal feuern während die entsprechende Gruppe aktiv ist. Anders als beim „`no-loop`" Attribut kann eine Regel mit „`lock-on-active`" nicht durch andere Regeln erneut aktiviert werden. Erst wenn die Gruppe ihren Fokus verliert, ist eine Aktivierung wieder möglich.

Beispiel:

```
rule "Die Regel"
```

```
    salience 999
    no-loop
    agenda-group "Die Gruppe"
    ...
```

3.3 Konfliktlösung in Drools

Die aus dem Abschnitt 2.4 bekannten Strategien zum „Auflösen" von Konfliktsituationen werden in Drools entweder automatisch durchgesetzt, oder durch die bekannten Regelattribute aus dem vorangegangenen Abschnitt umgesetzt.

Die Refraktionsstrategie wird von Drools automatisch angewendet. Eine Regel, die den Zustand eines Faktums abfragt, es aber nicht verändert, wird nur einmal gefeuert, obwohl ihr Bedingungsteil auch im Nachhinein noch wahr ist.

```
rule "Runde Reifen rollen"
    when
        exists Reifen(form == "rund")
    then
        System.out.println("Es gibt mindestens einen runden Reifen.");
end
```

Die Prioritätsstrategie wird durch das `salience` Regelattribut umgesetzt. Im Fall, dass eine Menge von Regeln durch dieselben Fakten aktiviert werden, diese aber eine bestimmte Reihenfolge haben sollen, in der sie gefeuert werden, lässt sich durch die Priorität regeln.

```
rule "Wenn eine Person gerade aufgewacht ist, dann geht sie duschen"
    salience 10
    when
        $p : Person(status == "gerade Aufgewacht")
    then
        System.out.println($p.getName() + " geht duschen.");
end

rule "Wenn eine Person gerade aufgewacht ist, dann geht sie frühstücken"
    when
        $p : Person(status == "gerade Aufgewacht")
    then
        System.out.println($p.getName() + " geht frühstücken.");
end
```

Die Spezifitätsstrategie wird in Drools durch das Regelattribut `salience` und durch das Bilden einer `activation-group` umgesetzt. Aus einer Menge von Regeln, die einer Activation Group angehören, kann immer nur eine Regel feuern, und mithilfe der Zuweisung von Prioritäten kann bestimmt werden, welche dieser Regeln feuern soll.

```
rule "Wenn das Kind eine gute Mathe- und Deutschnote hat, dann bekommt es..."
   activation-group "Spezifisch"
   salience 1
   when
      $k : Kind(matheNote == "gut" && deutschNote == "gut")
   then
      System.out.println("Kind bekommt Schwarzwalderkirsch Torte");
end

rule "Wenn das Kind eine gute Mathenote hat, dann bekommt es..."
   activation-group "Spezifisch"
   /*benötigt kein salience, da Defaultwert = 0 liegt
      die Priorität unter der der anderen Regel*/
   when
      $k : Kind(matheNote == "gut")
   then
      System.out.println($k.getName() + " bekommt Fisch mit Nudeln");
end
```

Die Aktualitätsstrategie wird von Drools ohne weiteres Zutun verwendet.

3.4 Der Rete Algorithmus

Der Rete[2] Algorithmus wurde 1979 von Charles Forgy im Rahmen seiner Doktorarbeit an der Carnegie Mellon University in den USA entwickelt[3]. Sein Ziel ist es, eine Effizienzsteigerung bei der Regelverarbeitung zu erlangen. Das wird erreicht, indem die Bedingungen von Regeln, die häufig gleiche Bedingungsteile enthalten, nur einmal überprüft werden und das Ergebnis gespeichert wird. Im weiteren Verlauf müssen nur die Änderungen an der Faktenbasis festgestellt werden, damit eine neue Konfliktmenge gebildet werden kann.

Es gibt für den Rete Algorithmus viele Optimierungen, auch eine abgeänderte Version wird in Drools verwendet. Zunächst wird der Rete Algorithmus in seiner einfachsten Fassung vorgestellt.
Der Rete Algorithmus erstellt aus allen Regelprämissen[3] ein Netzwerk aus Knoten. Dabei wird im wesentlichen zwischen vier Knoten unterschieden.[5, 4]

[2] „rete" aus dem lateinischen „Netz"
[3] Prämissen sind Bedingungen einer Regel.

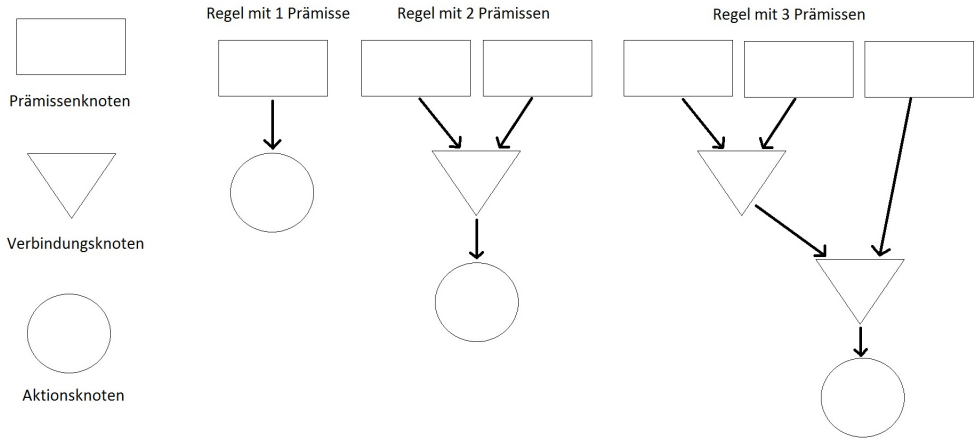

Abbildung 3.1: Knoten im Rete-Netzwerk

- Root/Rete Node: Eingangsknoten
- 1-Input/1-Output Node: Prämissenknoten
- 2-Input/1-Output Node: Verbindungsknoten
- Terminal Node: Aktionsknoten

Der Eingangsknoten dient als Eingang für alle Fakten.
Eine Bedingung wird durch einen Prämissenknoten dargestellt, wie zum Beispiel „Ist die Banane gelb?". Für jede Bedingung wird ein solcher Knoten erstellt.
Der Verbindungsknoten steht dafür, wenn eine Regel mehr als eine Bedingung hat. So werden immer jeweils zwei Prämissenknoten zu einem Verbindungsknoten zusammengeführt. Wenn beide Prämissen wahr sind, dann ist auch der Verbindungsknoten wahr. Bei drei Prämissenknoten würde der dritte mit dem Verbindungsknoten der ersten beiden verknüpft zu einem weiteren Verbindungsknoten. Es werden also weitere Verbindungsknoten erzeugt, bis alle Bedingungen in einem Knoten zusammengefasst sind.
Das Ergebnis des letzten Verbindungsknotens dient als Input für den Aktionsknoten, der die Aktionen der Regel darstellt.

Im ersten Durchgang werden alle Prämissenknoten überprüft, also mit der Faktenbasis abgeglichen, und der Wert, ob Wahr oder Falsch, wird abgespeichert. Ab dem zweiten Durchlauf muss nicht jede Prämisse wiederholt geprüft werden. Es werden nur noch die Prämissen überprüft, deren Fakten sich verändert haben.

3.4.1 Rete-OO

Rete-OO ist die Erweiterung des Rete Algorithmus, den Drools verwendet. „OO" steht hier für „object oriented". Hier kommen einige Knoten dazu, die Anhand von Regelbeispielen und ihrem dazugehörigen Rete-Netzwerk erklärt werden.[6]

```
rule "Regel 1"
   when
      $p : Person(name == "Joachim")
      $a : Adresse(strasse == "Spreeweg")
   then
   ...
end
```

Der erste Knoten ist der „Root Node". Er dient wieder als Eingang für alle Fakten (bzw. jetzt Objekte) in das Rete Netz. Der nächste Knoten ist ein „Default" „Entry Point Node" (Dunkelgrün). „Default" in diesem Fall, weil keine weiteren Entry Points definiert sind. Wären weitere definiert, so könnten die Fakten, die das Rete Netz betreten, in separate Netzwerke unterteilt werden.Die nächsten Knoten sind „Object Type Nodes" (Rot). Sie stellen die verwendeten Objekte im Bedingungsteil unserer Regel dar. Ein Knoten steht für *Person* und der andere für *Adresse*. Wenn wir jetzt mehrere Regeln hätten die auch Fakten vom Typ *Person* oder *Adresse* abfragen, so würden keine weiteren Object Type Nodes dazu kommen. Darauf folgen die „Alpha Nodes" (Blau). Jede der Alpha Nodes steht für eine literale Bedingung. In diesem Fall haben wir für das Objekt *Person* die Bedingung `name == "Joachim"` und für das Objekt *Adresse* die Bedingung `strasse == "Spreeweg"`. Würden dieselben Bedingungen auch in anderen Regeln stehen, so würden ähnlich wie beim Object Type Node auch hier keine weiteren Alpha Nodes dazukommen. Der darauf folgende Knoten ist ein „Left Input Adapter Node" (Gelb). Davon gibt es für jede

Abbildung 3.2: Legende ReteOO Knoten

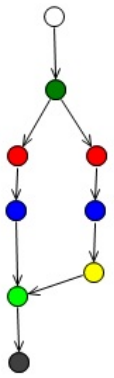

Abbildung 3.3: ReteNetz Regel 1

Regel einen, egal wie komplex oder simpel sie ist. Sie generiert aus dem Objekt, das über die Object Type Node und Alpha Node weitergegeben wird, ein Tupel. Dieses Tupel wird benötigt, um die „Terminal Node" (Grau) zu erreichen. Das Tupel wird an die „Join Node" (Hellgrün) weitergeleitet. Join Nodes haben zwei Eingänge (siehe Verbindungsknoten aus Abschnitt 3.4). Am linken Eingang liegt das Tupel an und am rechten Eingang das Objekt, mit dem das Tupel „sich vereinen soll". Liegen an den beiden Eingängen der Join Node jeweils „true" an, so wird das Tupel weiter zur Terminal Node geleitet, wodurch die Aktivierung der Regel ausgelöst wird.

Ein weiteres Beispiel soll darstellen, wie der ReteOO Algorithmus mit mehreren Regeln, die ähnliche Bedingungen haben, umgeht. Die erste Regel wird beibehalten und eine weitere wird der Regelbasis hinzugefügt.

```
rule "Regel 2"
    when
        $p : Person(name == "Joachim" && hatHunger == false)
        $a : Adresse(strasse == "Spreeweg")
        $k : Kuchen(gehoertPerson == $p)
    then
    ...
end
```

Durch „Regel 2" ist eine neue Object Type Node dazugekommen, da mit Ausnahme vom *Kuchen* Objekt *Adresse* und *Person* schon in der ersten Regel vorhanden sind. Von den Alpha Nodes ist eine neue dazugekommen, obwohl zwei neue literale Bedingungen in „Regel 2" enthalten sind. Die neue Alpha Node steht für `hatHunger == false`, für die Bedingung `gehoertPerson == $p` wird keine Alpha Node erstellt, da diese abhängig von `$p` ist. Die Bedingung findet sich in der markierten Join Node wieder. Auf die Alpha Nodes folgen die jeweils für „Regel 1" und „Regel 2" zuständigen Left Input Adapter Nodes, die die Objekte in Tupeln weiterleiten. Es sind in diesem Beispiel zwei Terminal Nodes vorhanden, da wir auch zwei Regeln haben.

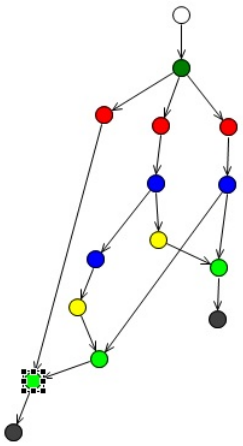

Abbildung 3.4: Rete Netz Regel 1+2

3.5 Domain Specific Language

Domain Specific Languages (DSL) werden benutzt, um Abstand von der Drools Regel Syntax zu bekommen. Bei der Verwendung einer DSL können normale Sätze in .drl Files verwendet werden. Dazu müssen diese Sätze in einer .dsl File vordefiniert werden. In dieser Definition wird dann auch das Konstrukt der entsprechenden Regel Syntax abgespeichert, das aus dem entsprechenden Satz entstehen soll. Damit in einer .drl File eine DSL verwendet werden kann, muss die entsprechende .dsl File angegeben werden. Das geschieht mit folgendem Code in einer .drl File: `expander dslTest.dsl`. Damit ist es möglich, die definierten Sätze zu verwenden, normale Konstrukte in der Drools Regel Syntax lassen sich mit einem Prefix „>" weiterhin einfügen. Zur Veranschaulichung ist in Abbildung 3.5 das Beispiel aus Abschnitt Rete-OO in eine DSL „übersetzt".

```
rule "Regel 1"
    when
        Es gibt eine Person mit dem Namen "Joachim"
        Es gibt eine Adresse mit der Strasse "Spreeweg"
    then
end

rule "Regel 2"
    when
        Es gibt eine Adresse mit der Strasse "Spreeweg"
        > $p : Person(name == "Joachim", hatHunger == false)
        Es gibt einen Kuchen, der der Person gehoert
    then
        Log : "Das ist ein System.out.println() Befehl."
end
```

Abbildung 3.5: DSL Regel Beispiel

Abbildung 3.6 zeigt die in der .dsl File definierten Sätze und ihre dazugehörigen Rule Language Konstrukte. In den geschweiften Klammern lassen sich „Variablen" festlegen, die in der .drl File mit einem beliebigen Wert belegt werden können.

Language Expression	Rule Language Mapping	Scope
Es gibt eine Person mit dem Namen "{value}"	$p : Person(name=="{value}")	[condition]
Es gibt eine Adresse mit der Strasse "{value}"	$a : Adresse(strasse == "{value}")	[condition]
Es gibt einen Kuchen, der der Person gehoert	$k : Kuchen(gehoert == $p)	[condition]
Log : "{message}"	System.out.println("{message}");	[consequence]

Abbildung 3.6: DSL Mapping

Im Drools Plugin für Eclipse ist ein Editor für .dsl Files enthalten, sowie ein spezieller Regel Editor für .drl Files, die eine DSL benutzen. In diesem ist es möglich, die Konstrukte aus der .dsl File einfach per Dropdown Menü auszuwählen (s. Abb. 3.7).

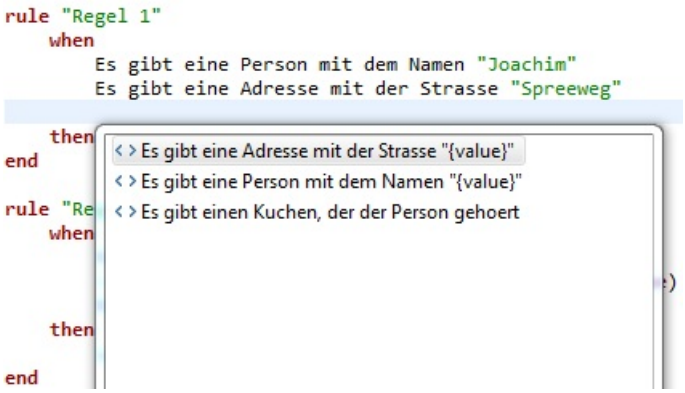

Abbildung 3.7: Regeleditor mit Dropdown für DSL

4 Modellierung der Adipositas Leitlinie

Die Adipositas Leitlinie wurde von der Deutschen Adipositas-Gesellschaft herausgegeben, um das Bewusstsein für das Gesundheitsproblem Adipositas zu stärken und um Empfehlungen zur Prävention und Therapie bereit zu stellen.[10]

4.1 Adipositas

Adipositas[1] ist eine chronische Krankheit, die sich in den letzten Jahrzehnten immer weiter ausgebreitet hat. In den Industriestaaten gehört sie bereits zu den Krankheiten mit der höchsten Mortalitätsrate[2][8]. Der Nationalen Verzehrsstudie II aus dem Jahr 2006 zufolge waren 58,2 % der Teilnehmer übergewichtig oder adipös[9]. Ob ein Mensch an Fettsucht erkrankt ist, wird über den Body Mass Index (BMI) berechnet. Dieser setzt sich aus dem Körpergewicht und der Körpergröße zum Quadrat zusammen: $BMI = \frac{Körpergewicht\,in\,Kg}{Körpergrösse\,in\,Meter^2}$ Übergewicht wird definiert durch einen BMI $\geq$ 25 kg/m² und Adipositas durch einen BMI $\geq$ 30 kg/m².

Kategorie	BMI
Untergewicht	$<$ 18,5
Normalgewicht	18,5 - 24,9
Übergewicht	$\geq$ 25
Präadipositas	25 - 29,9
Adipositas Grad I	30 - 24,9
Adipositas Grad II	35 - 39,9
Adipositas Grad III	$\geq$ 40

Tabelle 4.1: Gewichtsklassifikation bei Erwachsenen anhand des BMI [10]

Bei der Diagnose der Adipositas spielt nicht nur der BMI, sondern auch andere Messwerte eine Rolle, um das Risiko von Begleiterkrankungen einschätzen zu können. Bei Menschen mit einem BMI $\geq$ 25 kg/m² wird der Taillenumfang gemessen. Über den Taillenumfang lassen

[1] auch Fettleibigkeit oder Fettsucht genannt
[2] Sterberate

sich erhöhte Risiken für Stoffwechsel und kardiovaskuläre[3] Komplikationen feststellen. Das Risiko für kardiovaskuläre Komplikationen erhöht sich außerdem durch Kriterien aus Tabelle 4.2. Treffen drei dieser Kriterien zu, wird von einem Metabolischen Syndrom gesprochen.

erhöhter Taillenumfang	Männer $\geq$ 102 cm Frauen $\geq$ 88 cm
erhöhte Triglyzeride (nüchtern)	$\geq$ 150 mg/dl (1,7 mmol/L) oder Medikamenteneinnahme
niedriges Cholesterin (nüchtern)	Männer $<$ 40 mg/dl (1,0 mmol/L) Frauen $<$ 50 mg/dl (1,3 mmol/L) oder Medikamenteneinnahme
Bluthochdruck	$\geq$ 130 mm Hg systolischer Blutdruck oder $\geq$ 85 mm Hg diastolischer Blutdruck oder Medikamenteneinnahme
erhöhte Nüchternblutglukose	$\geq$ 100 mg/dl (5,6 mmol/L) oder Medikamenteneinnahme

Tabelle 4.2: Kriterien für die Diagnose des Metabolischen Syndroms [10]

Therapie

Der Leitlinie nach werden für jede BMI-Kategorie Ziele festgelegt und Maßnahmen, mit denen die Ziele erreicht werden sollen.

Bei einem BMI $<$ 30 kg/m^2 soll dem Patienten geraten werden sein Gewicht zu beobachten und als Ziel wird die Stabilisierung des Gewichts gesetzt.

Patienten mit einem BMI von $\geq$ 30 kg/m^2 oder $<$ 30 kg/m^2 und Begleiterkrankungen wie z.B. Diabetes oder erhöhtem Risiko für Begleiterkrankungen wird als Ziel eine dauerhafte Gewichtsreduktion gesetzt und eine Gewichtsstabilisierung auf lange Sicht. Als Maßnahmen werden die Begleiterkrankungen therapiert und das Basisprogramm soll durchgeführt werden.

Das Basisprogramm setzt sich aus Ernährungstherapie, Bewegungstherapie und Verhaltenstherapie zusammen.

Die Ernährungstherapie wird in vier Stufen unterteilt, wobei nicht jeder Patient alle Stufen durchgehen muss, sondern jede Stufe für ein weniger straffes bis sehr straffes Konzept steht. Während Stufe 1 bei der Nahrungsaufnahme ein tägliches Energiedefizit von etwa 500 kcal anstrebt und damit über einen Zeitraum von 6 Monaten einen durchschnittlichen Gewichtsverlust von 3,2 - 4,3 kg bewirkt, so wird bei Stufe 4 eine tägliche Gesamtenergiemenge von 800 bis 1200 kcal angestrebt, was einen Gewichtsverlust von 0,5 - 2 kg pro Woche

[3]Das Kardiovaskuläre System ist das Netz der Blutgefäße[11]

ermöglicht. Ein erhöhter Energieverbrauch lässt sich durch eine Bewegungstherapie erzielen. Sie trägt zur Gewichtsabnahme und Gewichtsstabilisierung bei. Die Verhaltenstherapie beinhaltet zum Beispiel das Dokumentieren des Ess-, Trink- und Bewegungsverhaltens in einem Ernährungstagebuch und Bewegungsprotokoll.
Wenn das Basisprogramm keine Wirkung zeigt, kann eine medikamentöse Therapie in Erwägung gezogen werden. Diese sollte aber nur dann fortgesetzt werden, wenn sich nach den ersten vier Wochen eine Gewichtsabnahme von mindestens 2 kg durchsetzt.
Zuletzt kann eine chirurgische Therapie nach Scheitern einer konservativen Therapie in Erwägung gezogen werden, allerdings nur bei Patienten mit Adipositas Grad III oder Grad II mit schweren Begleiterkrankungen wie Diabetes.

4.2 Wissensaquisition aus einem Dokument

Bei der Wissensgewinnung kann man zu Anfang den Anwendungsbereich in Teilbereiche einteilen, um einen „Ablauf" zu erstellen. Das macht Sinn, wenn ein Bereich auf das Ergebnis eines anderen Bereichs angewiesen ist. In einem medizinischen Bereich könnten das zum Beispiel Diagnose und Therapie sein, bei der die Therapie nicht beginnen kann, bevor eine Diagnose gestellt wurde. Sollten die Bereiche nicht voneinander abhängig sein, so können die Regeln der Übersicht wegen getrennt modelliert werden, jedoch sollte kein künstlicher Ablauf modelliert werden.
Daraufhin sollte jeder Teilbereich weiter in „Unterziele" zerkleinert werden. Unterziele können am Beispiel der Adipositas die Berechnung des BMI oder die Feststellung des Risikos für Begleiterkrankungen sein. Diese Unterziele bestehen dann aus einer oder mehreren Regeln, die alle für das System relevanten Fälle abdecken. Ein Beispiel einer Regel, die irrelevant für das Regelsystem ist, wäre bei einem Blitzer: Wenn das Auto unter 60 km/h fährt, dann blitzt es nicht.

4.3 Modellierung der Adipositas Regeln und des Datenmodells

4.3.1 Datenmodell

Das Datenmodell besteht aus den Klassen (oder auch „Fakt Typen"), die für das System bei der Verarbeitung von Regeln relevant sind. Das sind Klassen, von denen Objekte instanziiert werden und als Fakten ins System eingegeben, modifiziert und gelöscht werden sollen.
Zum einen wird die Klasse *Patient* erstellt. Sie hat als Attribute *name* und *geschlecht*. Beide

werden als Schlüsselattribute deklariert, ohne die Regelverarbeitung nicht stattfinden kann.

```
declare Patient
   name : String @key
   geschlecht : String @key
end
```

Die Klasse *Messdaten* trägt die Attribute mit sich, die nötig sind, um eine komplette Diagnose zu erstellen. Auch in diesem Fall sind alles Attribute als Schlüssel deklariert mit Ausnahme des *BMI* Attributs, welches sich aus den Attributen *gewichtInKg* und *groesseInMeter* errechnet.

```
declare Messdaten
   gewichtInKg : double @key
   groesseInMeter : double @key
   taillenUmfang : double @key
   triglyzeride : int @key
   hdl_cholesterin : int @key
   nuechternblutglukose : int @key
   blutdruck : int @key    //systolischer Blutdruck / oberer Blutdruck
   bmi : double
end
```

Die Klasse *Krankheitsbild* trägt Attribute, die die Ausprägung der Krankheit beschreiben und damit die Diagnose stellen.

```
declare Krankheitsbild
   abdominaleAdipositas : boolean
   adipositasGrad : String
   risikoBegleiterkrankungGrad : int
   risikoMetabolischeKardiovaskulaereKomplikationen : String
   metabolischesSyndrom : boolean
end
```

Um die Messdaten und das Krankheitsbild einem Patienten zuordnen zu können, werden zwei „Helferklassen" benötigt. Zum einen die Klasse *KrankheitsbildVonPatient* und *MessdatenVonPatient*. Beide Klassen tragen jeweils ein Patient Attribut mit sich, sowie ein *Messdaten* bzw. *Krankheitsbild* Attribut.

```
declare MessdatenVonPatient
   patient : Patient
   messdaten : Messdaten
```

```
end

declare KrankheitsbildVonPatient
   patient : Patient
   krankheitsbild : Krankheitsbild
end
```

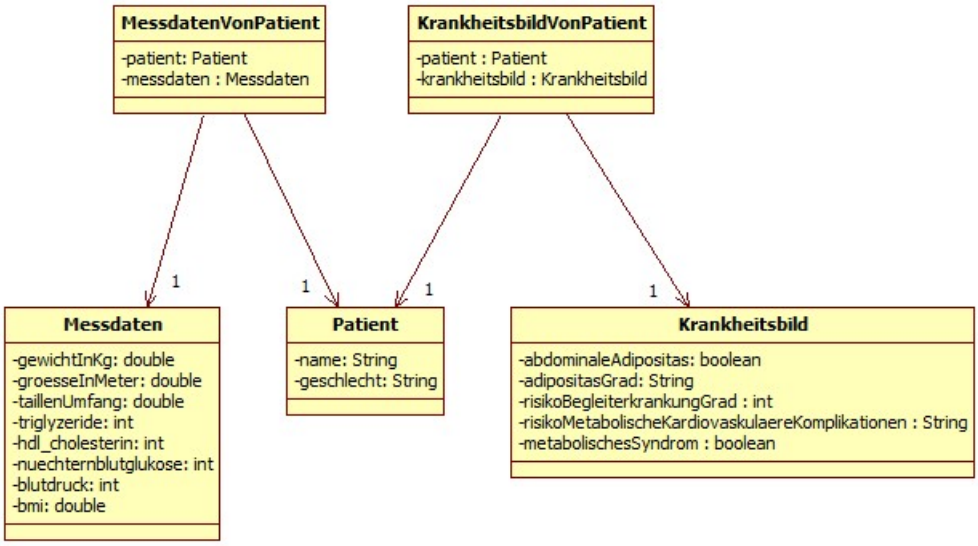

Abbildung 4.1: UML Klassendiagramm

Abbildung 4.1 zeigt das Datenmodell in einem UML Klassendiagramm. Die Pfeile stellen eine gerichtete Assoziation dar. Assoziationen stehen dafür, dass Objekte kommunizieren können. Eine gerichtete Assoziation lässt die Kommunikation nur in einer Richtung durch. Die Zahl am Ende des Pfeils steht für die Multiplizität. Sie gibt die Anzahl der Objekte an, mit denen das Ausgangsobjekt in Beziehung stehen kann.

4.3.2 Diagnose-Regeln

Zur Modellierung der Diagnose-Regeln wurde zu Anfang festgestellt, welche „Unter- bzw. Nebenziele" im Diagnoseteil von Bedeutung sind. Daraus haben sich Ziele, die unbedingt erreicht werden müssen, und optionale Ziele ergeben.
Zu den optionalen Zielen gehört zum Beispiel die Diagnose des metabolischen Syndroms, das bei Patienten auftreten kann, aber nicht muss. Die Berechnung des BMI dagegen muss durchgeführt werden, da ohne sie dem Patienten kein *adipositasGrad*-Attribut zugewiesen werden kann, was Vorraussetzung für die Regeln im Therapiebereich ist.

```
function double calcBMI(double gewichtInKg, double groesseInMeter){
    double bmi = gewichtInKg / (groesseInMeter * groesseInMeter);
    return bmi;
}

rule "Berechne den BMI des Patienten"
    agenda-group "group diagnose"
    when
        $patient : Patient()
        $messdatenVonPatient : MessdatenVonPatient($patient == patient && messdaten.bmi
            == 0.0 && messdaten.gewichtInKg > 0.0 && messdaten.groesseInMeter > 0.0)
    then
        modify($messdatenVonPatient){getMessdaten().setBmi(calcBMI($messdatenVonPatient
            .getMessdaten().getGewichtInKg(),$messdatenVonPatient.getMessdaten().
            getGroesseInMeter()))}
end
```

Auf die BMI Berechnung folgt die Zuweisung des *adipositasGrad*-Attributs. Eine weitere optionale Diagnose ist die abdominale Adipositas, mit der gleichzeitig ein erhöhtes Risiko auf metabolische und kardiovaskuläre Komplikationen diagnostiziert wird.
Alle Diagnose-Regeln werden einer Agenda Gruppe zugeordnet, sodass die Therapie-Regeln erst feuern können, wenn die Diagnose vollständig abgeschlossen ist. Dazu wird allen optionalen Regeln eine höhere Priorität gegeben, als die der Regel, die das *adipositasGrad*-Attribut setzt, da diese Regeln den Fokus der Agenda Gruppe auf die Therapie-Regel-Gruppe legen. Die höhere Priorität für die optionalen Regeln sind daher wichtig, da die vorgeschlagene Therapie durch sie beeinflusst werden kann.

4.3.3 Therapie Regeln

Die vorgeschlagenen Maßnahmen und gegebenen Ziele zur Therapie der Adipositas sind in der Leitlinie eindeutig dargestellt. So können die Regeln hauptsächlich anhand des *adipositasGrad*-Attributs und *risikoMetabolischeKardiovaskulaereKomplikationen*-Attributs erstellt werden. Dabei kommt es vor, dass es verschiedene Therapien für einen Krankheitsgrad gibt. Wenn ein Patient zum Beispiel *adipositasGrad = „Präadipositas”* hat, gibt es drei mögliche Therapien. Diese werden dann noch durch andere Bedingungen spezifiziert. In diesem Fall muss die Spezifitätsstrategie (s. Abschnitt 3.3)angewandt werden, sodass am Ende die eine „richtige” Therapie vom System vorgeschlagen werden kann und nicht eine der weniger speziellen Regeln ausgelöst wird, obwohl eine ganz andere Therapie angebracht ist.

```
rule "Wenn Patient der Kategorie 'Präadipositas' angehoert, dann werden folgende
    Maßnahmen vorgeschlagen"   agenda-group "group therapie"
   activation-group "Präadipositas"
   when
      $patient : Patient()
      KrankheitsbildVonPatient(patient == $patient && krankheitsbild.adipositasGrad
          == "Präadipositas")
      MessdatenVonPatient(patient == $patient)
   then
      System.out.println("Patient " + $patient.getName() + " ist Präadipös. " + "Ziel
          ist es...");
end
```

4.4 Versuche und Tests

Um den modellierten Anwendungsbereich zu testen, wird dem System eine Reihe von Testpatienten mit verschiedenen Messdaten eingegeben, sodass möglichst alle definierten Regeln mindestens einmal feuern können. Um festzustellen welche Regeln ausgelöst werden, hat jede Regel im Konsequenzteil einen Methodenaufruf, der einen passenden Text zur Regel in die Standardausgabe ausgibt.

Im ersten Test hat ein männlicher Patient folgende Messdaten:

gewichtInKg:	140 kg
groesseInMeter:	1,80 m
taillenUmfang:	104 cm
triglyzeride:	140 mg/dl
hdl_cholesterin:	35 mg/dl
nuechternblutglukose:	90 mg/dl
blutdruck:	135 mm Hg (systolischer Blutdruck)

Davon ausgehend müsste eine abdominale Adipositas diagnostiziert werden, aufgrund des Taillenumfangs. Außerdem ein metabolisches Syndrom, da mindestens 3 der 5 ausschlaggebenden Kriterien zutreffend sind. Dazu kommt die Berechnung des BMI, die einen Wert von über 40 ergibt. Daraus folgt, dass der Patient Adipös Grad III ist. Für Patienten mit Adipositas Grad III gibt es nur eine Therapie, die laut der Leitlinie in Frage kommt, welche das volle Programm umfasst.

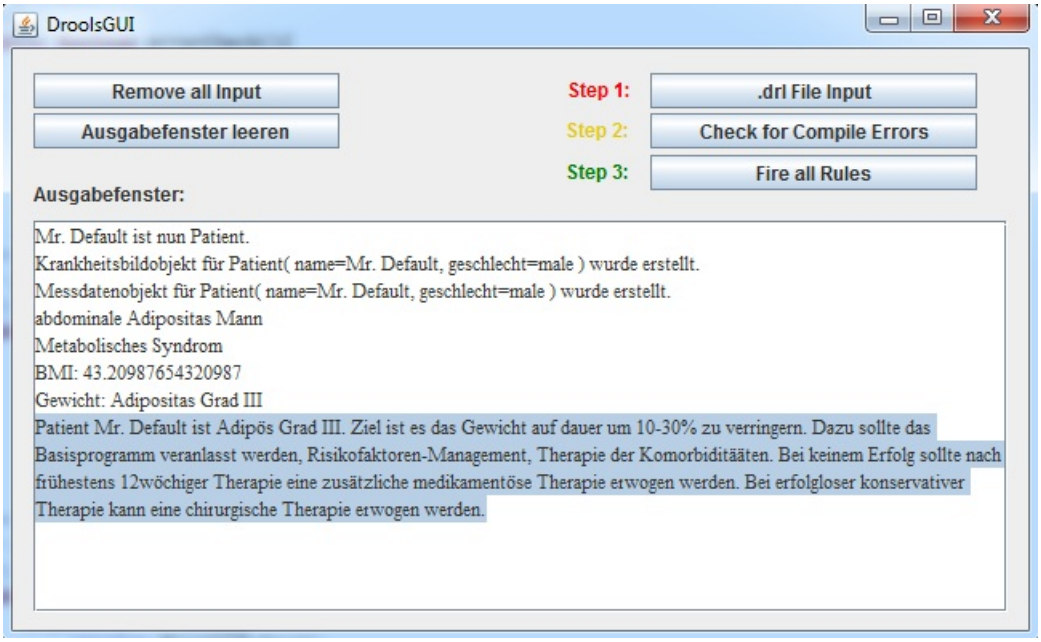

Abbildung 4.2: Ergebnis Testpatient 1

Abbildung 4.2 zeigt jede Regel, die gefeuert wurde und bestätigt damit die Annahmen. Der markierte Teil ist die vorgeschlagene Therapie.

Im zweiten Test hat eine weibliche Patientin folgende Messdaten:

gewichtInKg:	90 kg
groesseInMeter:	1,75 m
taillenUmfang:	84 cm
triglyzeride:	130 mg/dl
hdl_cholesterin:	45 mg/dl
nuechternblutglukose:	90 mg/dl
blutdruck:	125 mm Hg (systolischer Blutdruck)

Die Berechnung des BMI ergibt einen Wert von ≈ 29, der die Patientin der Kategorie „Präadipositas" zuordnet. Des Weiteren hat sie einen Taillenumfang ≥ 80 cm, was bei einer Frau ein erhöhtes Risiko für metabolische und kardiovaskuläre Komplikationen bedeutet. Ihre restlichen Messwerte liegen alle im Rahmen des Ungefährlichen. Die vorgeschlagenen Therapiemaßnahmen in der Kategorie „Präadipositas" variieren der Leitlinie zufolge. So wird bei einem Taillenumfang einer Frau ≥ 80 cm, oder vorhandenem Risikofaktor und Begleiterkrankungen, eine Therapie mit Basisprogramm und Therapie der Begleiterkrankungen „verordnet". Noch spezifischer wird der Fall, wenn der Patient einen BMI von über 27 hat, dann kann nach zwölfwöchiger Therapie eine zusätzliche medikamentöse Behandlung erfolgen.

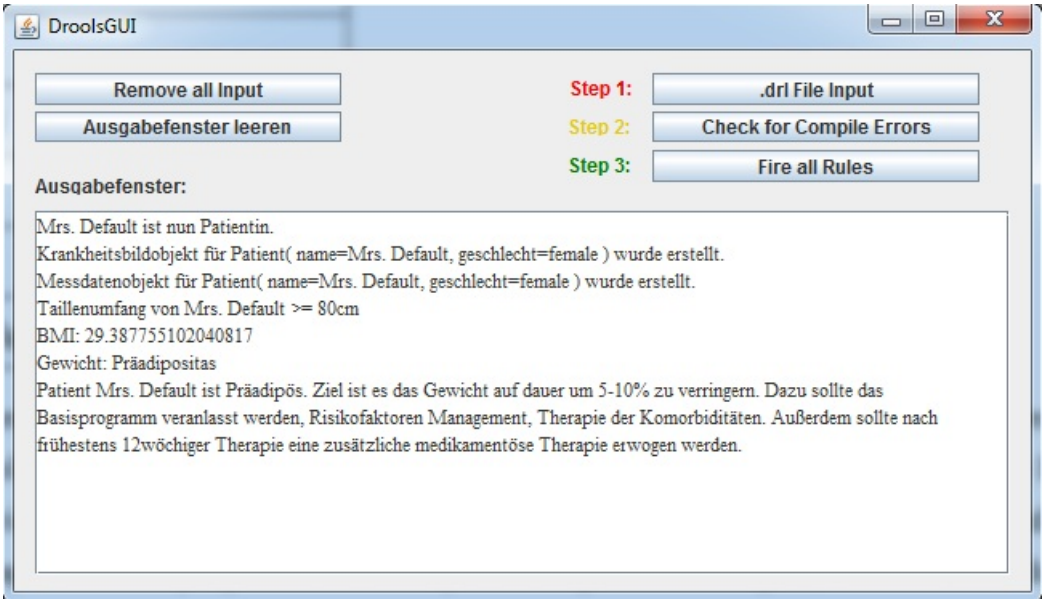

Abbildung 4.3: Ergebnis Testpatient 2

Abbildung 4.3 zeigt die ausgelösten Regeln und als Resultat die daraus gefolgerte Therapie.

5 Implementierung

In diesem Kapitel wird die Implementation von Drools in die im Rahmen der Arbeit entwickelte GUI[1] vorgestellt, sowie die GUI selbst. Dazu wurde die Eclipse IDE[2] verwendet, und das Drools Plugin (Version 5.4.0) für Eclipse.

5.1 Drools Plugin

Das Drools Plugin kann über die JBoss Webseite[3] bezogen werden, oder direkt über den Eclipse Update Service[4]. Es bietet Funktionen, die bei der Erstellung von Regeln und Drools Projekten helfen. Es beinhaltet einen Regeleditor, der den Benutzer auf Syntax- und Compilerfehler aufmerksam macht, sowie einen Domain Specific Language Editor (s. Abb. 3.6) und einen speziellen Regeleditor für Regel Dateien die DSLs benutzen. Dieser kennt die definierten DSL Sprachkonstrukte, sodass die Regeln einfach zusammengefügt werden können. Eine benötigte Komponente ist das Eclipse Graphical Editing Framework (GEF). Es wird unter anderem für die Erstellung der Rete View benötigt (s. Abb. 3.4). Manche Eclipse Distributionen haben es bereits vorinstalliert. Sollte es fehlen, kann es auch über den Eclipse Update Service installiert werden[5].

5.2 Implementierung von Drools

Für ein neues Drools Projekt benötigt es eine definierte Drools Runtime, sie beinhaltet zum Beispiel die RETE Engine und funktioniert als Compiler für Regeln. Sie kann in den Optionen des Drools Plugin erstellt werden (s. Abb. 5.1).

Der folgende Code zeigt die Implementation von Drools in Java.

[1] GUI steht für Graphical User Interface
[2] Integrated Development Environment
[3] http://www.jboss.org/drools/downloads
[4] Help -> Install New Software -> http://download.jboss.org/drools/release/5.4.0.Final/org.drools.updatesite/
[5] Help -> Install New Software -> http://download.eclipse.org/tools/gef/updates/releases

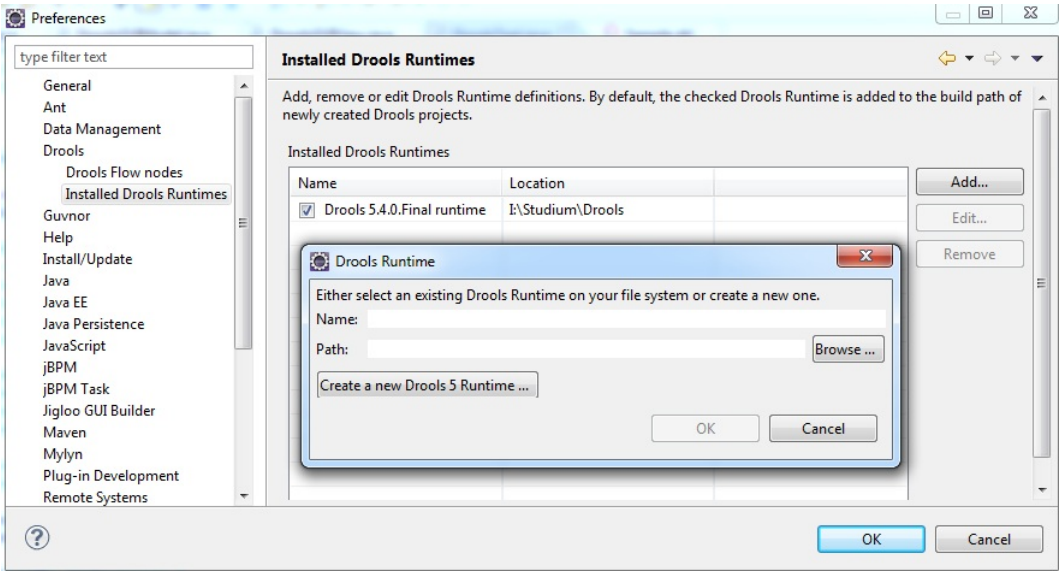

Abbildung 5.1: Erstellung einer Drools Runtime

```
//imports
public class DroolsTest {
    public static void main(String[] args){
        File drlFile;
        KnowledgeBuilder builder = KnowledgeBuilderFactory.newKnowledgeBuilder();
        drlFile = new File("C:\\myRules.drl");
        builder.add(ResourceFactory.newFileResource(drlFile), ResourceType.DRL);

        if (builder.hasErrors()) {
            throw new RuntimeException(builder.getErrors().toString());
        }

        KnowledgeBase knowledgeBase = builder.newKnowledgeBase();

        StatefulKnowledgeSession session = knowledgeBase.newStatefulKnowledgeSession();
        //hier können Fakten der Session hinzugefügt werden
        session.fireAllRules();
        session.dispose();
    }
}
```

Als erstes wird ein `KnowledgeBuilder` erstellt. Ihm werden alle Drools Rule Files, mit denen gearbeitet werden soll, zugewiesen. Er wandelt die Dateien in `KnowledgePackage` Objekte um, die von der `KnowledgeBase` genutzt werden.

```
File drlFile;
KnowledgeBuilder builder = KnowledgeBuilderFactory.newKnowledgeBuilder();
drlFile = new File("C:\\myRules.drl");
builder.add(ResourceFactory.newFileResource(drlFile), ResourceType.DRL);
```

Im nächsten Schritt werden die eingelesenen Ressourcen auf Compilerfehler überprüft und falls vorhanden werden Exceptions geworfen. Dieser Schritt ist optional.

```
if (builder.hasErrors()) {
   throw new RuntimeException(builder.getErrors().toString());
}
```

Daraufhin wird die `KnowledgeBase` aus den Informationen, die dem `KnowledgeBuilder` hinzugefügt wurden, erstellt.

```
KnowledgeBase knowledgeBase = builder.newKnowledgeBase();
```

Zuletzt wird eine `StatefulKnowledgeSession` instanziiert. In die Session werden die Fakten eingegeben, sie dient somit als Faktenbasis. *Stateful* bedeutet, dass die Fakten, solange die Session läuft, gespeichert sind. Eine *Stateful* Session wird erst mit der *dispose*-Methode beendet. Während die Session läuft, kann auch öfters die *fireAllRules*-Methode genutzt werden, während bei einer *Stateless* Knowledge Session zu Anfang nur einmal eine *execute*-Methode aufgerufen wird, und die Regeln mit den vorhandenen Fakten gefeuert werden bis keine Regelaktivierungen mehr vorhanden sind. Woraufhin sich die *Stateless* Session von selbst auflöst.

```
StatefulKnowledgeSession session = knowledgeBase.newStatefulKnowledgeSession();
//hier können Fakten der Session hinzugefügt werden
session.fireAllRules();
session.dispose();
```

5.3 GUI

Die GUI wurde entwickelt, damit Droolsanfänger sich nur um das Erstellen von Regeln kümmern müssen, um erste Erfolge zu erzielen, ohne sich mit Eclipse, dem Drools Plugin und dem Java Code auseinander setzen zu müssen. Die GUI ist einfach gehalten und setzt nur die offensichtlichsten Funktionen um. Sie bietet keine Funktionen, die in den Ablauf von Regeln eingreifen, Fakten modifizieren, eingeben oder löschen. Daher muss alles was dem Anwendungsbereich angehört über .drl Files der GUI bzw. der Drools Engine „gespeist" werden. Die Funktionen sind das Einlesen von .drl Dateien, Überprüfung der syntaktischen Korrektheit dieser Dateien und das Feuern der feuerbereiten Regeln. Ein genereller Ablauf bei der Nutzung der GUI sieht folgendermaßen aus:

1. Es werden zu Anfang eine oder mehrere .drl Dateien angegeben über den Button „.drl File Input"

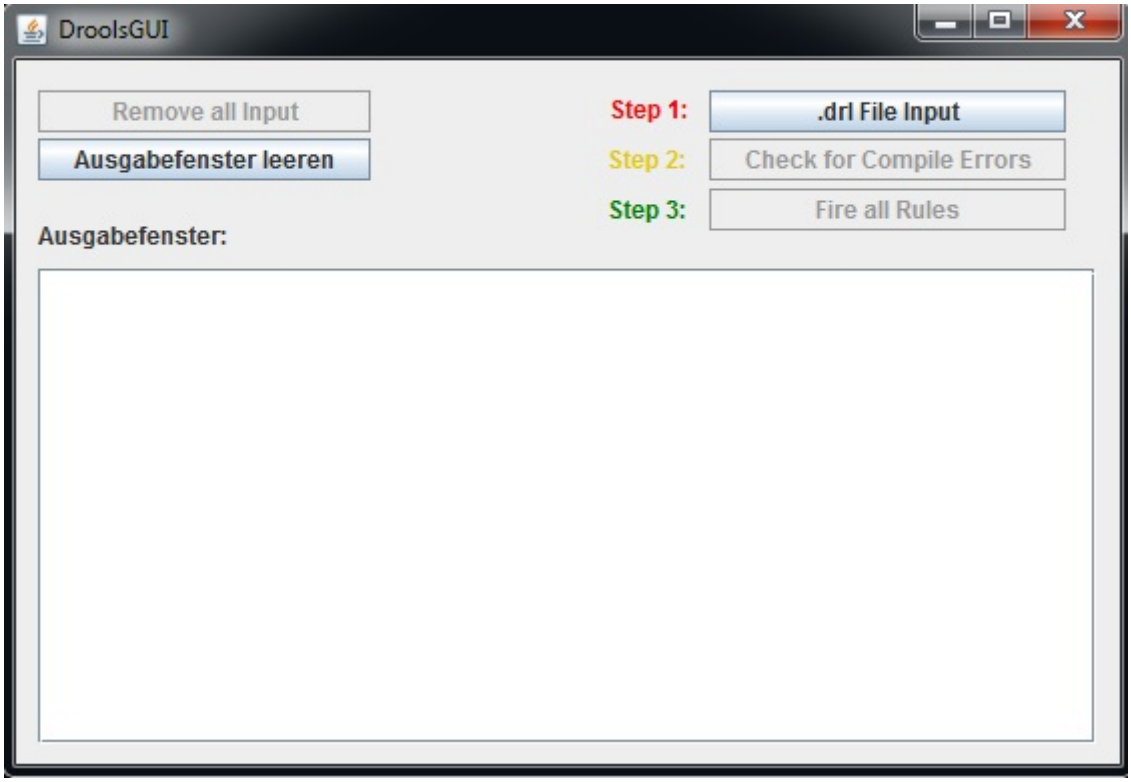

Abbildung 5.2: Drools GUI

2. Wenn alle Dateien angegeben sind, werden eventuelle Compiler-Fehler über den Button „Check for Compile Errors" im Ausgabefenster angezeigt
3. Sollten keine Fehler auftreten, lassen sich die aktivierten Regeln mit dem Button „Fire all Rules" abfeuern.
4. Um die aktuellen Daten zu löschen, kann der „Remove all Input" Button gedrückt werden. Wenn eine bereits eingelesene Datei verändert wurde, kann diese einfach neu eingelesen werden.

6 Benutzerhandbuch

Dieses Kapitel ist für Benutzer gedacht, denen grundsätzliche Begriffe aus regelbasierten Systemen bekannt sind, mit Drools und seiner Syntax aber noch nicht vertraut sind. Des Weiteren sollten dem Benutzer Java Grundkenntnisse zur Verfügung stehen, da im Folgerungsteil der Regeln die Syntax von Java angewendet wird. Es folgen Schritt für Schritt Anweisungen, die dem Benutzer die grundlegende Drools Syntax an einem Beispiel zeigt, sodass es ihm möglich ist, selber einen Anwendungsbereich bis zu einem gewissen Level zu modellieren.

Step 1:

Erstelle eine neue Textdatei und nenne sie *„BilderbuchWelt.drl"*. Eine .drl Datei ist nichts anderes als eine Textdatei. Die Endung besagt nur, dass es sich um eine Drools Rule File handelt.

Step 2:

Wir definieren so genannte Fakt Typen in der .drl Datei.

```
declare Sonne
   strahlt : boolean
end

declare Blume
   istGluecklich : boolean
end
```

Fakt Typen werden zu eigenständigen Java Klassen durch den Drools Compiler übersetzt. Zum Beispiel würde für den Fakt Typ *„Sonne"* eine Klasse mit dem booleschen Instanzattribut *„strahlt"* erstellt. Dazu wird automatisch ein Konstruktor mit allen Attributen, einem default Konstruktor ohne Attribute, sowie Setter und Getter erstellt. Den Fakt Typen können keine Funktionen in der .drl gegeben werden. Vererbung wird wie in Java durch das Schlüsselwort *„extends"* realisiert.

Step 3:

Wir definieren unsere Regeln:

```
rule "Regel für Testdaten"
   when
      not Sonne()
   then
      insert(new Sonne(true));
      insert(new Blume());
end

rule "Wenn eine Sonne strahlt, dann sind alle Blumen glücklich."
   when
      exists Sonne(strahlt == true)
      $b : Blume()
   then
      modify($b){setIstGluecklich(true)};
end
```

Im „*Wenn*"-Teil der ersten Regel wird die Bedingung „Es existiert keine Sonne" gestellt. Damit der Aktionsteil (then) ausgeführt werden kann, darf der Faktenbasis also kein „*Sonne*" Objekt bekannt sein. Mit der *insert()* Methode werden der Faktenbasis die Fakten bekannt gemacht. In diesem Fall ein Faktum vom Typ „*Sonne*" und dem Attribut „*strahlt = true*", sowie ein Fakt des Typs „*Blume*". In der zweiten Regel wird das Blumen Faktum einer Variabel zugewiesen, sodass das Faktum im Aktionsteil modifiziert oder gelöscht werden kann (mit der retract(Object o) Methode können Fakten gelöscht werden). Ein einfacher Setteraufruf würde keinen Effekt mit sich bringen, da der Rete Tree nur mit den entsprechenden Methoden aktualisiert wird. Wichtig ist zu beachten, dass NUR Objekte die sich in der Faktenbasis befinden mit diesen Methoden „gehandlet" werden können. Jedes Objekt wird also mit *insert(Object o)* der Faktenbasis bekannt gemacht, das kann entweder in einer Regel passieren oder in der Java Applikation (wird hier nicht behandelt).

Step 4:

Wir fügen unseren Regeln *System.out.println()* Statements hinzu um beim Feuern der Regeln verfolgen zu können, was passiert.

```
rule "Regel für Testdaten"
   when
      not Sonne()
   then
      insert(new Sonne(true));
      insert(new Blume());
      System.out.println("Testfakten wurden erzeugt.");
end
```

```
rule "Wenn eine Sonne strahlt, dann sind alle Blumen glücklich."
    when
        exists Sonne(strahlt == true)
        $b : Blume()
    then
        modify($b){setIstGluecklich};
        System.out.println("Die Sonnenstrahlen lassen die Blume frohlocken!");
end
```

In diesem Zustand aktiviert sich die zweite Regel immer wieder von neuem selbst, daher muss der Regel das no-loop Attribut hinzugefügt werden.

```
rule "Wenn eine Sonne strahlt, dann sind alle Blumen glücklich."
    no-loop
    when
        exists Sonne(strahlt == true)
        $b : Blume()
    then
        modify($b){setIstGluecklich};
        System.out.println("Die Sonnenstrahlen lassen die Blume frohlocken!");
end
```

So können die Regeln ausgeführt werden.

7 Ausblick und Fazit

Um aus der modellierten Adipositas Leitlinie eine in der Praxis verwendbare Applikation zu bekommen, muss eine Benutzerschnittstelle programmiert werden, mit der Patienten- und Messdaten in das System eingefügt werden können. Dazu wird eine Datenbank benötigt, die als Backup der eingegebenen und von den Regeln erzeugten Daten dient. Als Backup daher, weil diese Daten bereits im Working Memory als Fakten gelagert sind und bei einem Systemausfall verloren gingen.

Ein Aspekt von Drools, der in einer weiteren Arbeit untersucht werden könnte, ist die Möglichkeit der Regelerstellung mithilfe eines grafischen Interfaces zu beleuchten. Dazu bietet Drools eine eigene Komponente: Drools Guvnor. Guvnor bringt weitere Funktionen mit sich, u.a. ein Repository zur Versionsverwaltung von Regelbasen und die Möglichkeit Regeln zu testen.

Auch wenn das im Rahmen der Arbeit entwickelte wissensbasierte Beispielsystem in diesem Zustand noch nicht in der Praxis einsetzbar ist, so kann man doch feststellen, dass Drools in medizinischen Anwendungen Verwendung finden kann. Neue medizinische Erkenntnisse erfordern ständige Modifikationen der Wissensbasis, die durch die Flexibilität von Regel Systemen mit wenig Aufwand angepasst werden können. Nicht zuletzt lieferten die in dem Beispielsystem getesteten Regeln durchaus korrekte Ergebnisse. Der in Drools implementierte Rete Algorithmus ermöglicht eine sehr schnelle Verarbeitung von Wissen, vor allem zu spüren, wenn Hunderte bis Tausende verschiedene Regeln geprüft werden müssen. Diese Anforderung kann besonders in komplexen medizinischen Anwendungen von Vorteil sein, besonders gegenüber einer Umsetzung ohne regelbasiertes System.

A Anhang

A.1 Drools Rule Files

dataModel.drl

```
package de.DroolsGUI.adipositasRules

declare Patient
   name : String @key
   geschlecht : String @key
end

declare KrankheitsbildVonPatient
   patient : Patient
   krankheitsbild : Krankheitsbild
end

declare Krankheitsbild
   abdominaleAdipositas : boolean
   adipositasGrad : String
   risikoBegleiterkrankungGrad : int
   risikoMetabolischeKardiovaskulaereKomplikationen : String
   metabolischesSyndrom : boolean
end

declare MessdatenVonPatient
   patient : Patient
   messdaten : Messdaten
end

declare Messdaten
   gewichtInKg : double @key
   groesseInMeter : double @key
   taillenUmfang : double @key
   triglyzeride : int @key
   hdl_cholesterin : int @key
   nuechternblutglukose : int @key
   blutdruck : int   @key  //systolischer Blutdruck / oberer Blutdruck
   bmi : double
end
```

diagnose_rules.drl

```
package de.DroolsGUI.adipositasRules


/*-------------------------------------
   Metabolisches Syndrom
-------------------------------------*/
function boolean calcMetabolischesSyndrom(String geschlecht, double taillenUmfang, int
     triglyzeride, int hdl_cholesterin, int nuechternblutglukose, int blutdruck){
   int i = 0;
```

```
    if((taillenUmfang >= 102 && geschlecht == "male") || (taillenUmfang >= 88 && geschlecht == "
        female")){
        i++;
    }
    if(triglyzeride >= 150){
        i++;
    }
    if((hdl_cholesterin < 40 && geschlecht == "male") || (hdl_cholesterin < 50 && geschlecht == "
        female")){
        i++;
    }
    if(nuechternblutglukose >= 100){
        i++;
    }
    if(blutdruck >= 130){
        i++;
    }
    if(i >= 3) return true;
    else return false;
}

rule "Wenn 3 von 5 Messwerten zu hoch bzw. zu niedrig liegen, dann liegt ein Metabolisches Syndrom
     vor"
    salience 1
    agenda-group "group diagnose"
    when
        $patient : Patient($geschlecht : geschlecht)
        MessdatenVonPatient(patient == $patient && $taillenUmfang : messdaten.taillenUmfang &&
            $triglyzeride : messdaten.triglyzeride && $hdl_cholesterin : messdaten.hdl_cholesterin
            && $nuechternblutglukose : messdaten.nuechternblutglukose && $blutdruck : messdaten.
            blutdruck)
        $kbp : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild.metabolischesSyndrom
            == true))
        eval(calcMetabolischesSyndrom($geschlecht, $taillenUmfang, $triglyzeride, $hdl_cholesterin,
            $nuechternblutglukose, $blutdruck))
    then
        modify($kbp){getKrankheitsbild().setMetabolischesSyndrom(true)};

        System.out.println("Metabolisches Syndrom");
end



/*----------------------------
    abdominale Adipositas
----------------------------*/
rule "Wenn der Taillenumfang von einem Mann >= 102cm beträgt liegt eine abdominale Adipositas vor
     und das Risiko auf Metabolische Kardiovaskuläre Komplikationen ist 'deutlich erhoeht'"
    salience 1
    agenda-group "group diagnose"
    when
        $patient : Patient(geschlecht == "male")
        MessdatenVonPatient(patient == $patient && (messdaten.taillenUmfang >= 102))
        $kbp : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild.
            risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich erhoeht"))
    then
        modify($kbp){getKrankheitsbild().setAbdominaleAdipositas(true),
                getKrankheitsbild().setRisikoMetabolischeKardiovaskulaereKomplikationen("deutlich
                    erhoeht")};

        System.out.println("abdominale Adipositas Mann");
end
rule "Wenn der Taillenumfang von einer Frau >= 88cm beträgt liegt eine abdominale Adipositas vor
     und das Risiko auf Metabolische Kardiovaskuläre Komplikationen ist 'deutlich erhoeht'"
    salience 1
    agenda-group "group diagnose"
    when
        $patient : Patient(geschlecht == "female")
```

```
        MessdatenVonPatient(patient == $patient && (messdaten.taillenUmfang >= 88))
        $kbp : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild.
            risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich erhoeht"))
    then
        modify($kbp){getKrankheitsbild().setAbdominaleAdipositas(true),
                getKrankheitsbild().setRisikoMetabolischeKardiovaskulaereKomplikationen("deutlich
                    erhoeht")};

        System.out.println("abdominale Adipositas Frau");
end

rule "Wenn der Taillenumfang von einem Mann >= 94cm beträgt so ist das Risiko auf Metabolische
     Kardiovaskuläre Komplikationen 'erhoeht'"
    salience 1
    agenda-group "group diagnose"
    when
        $patient : Patient(geschlecht == "male")
        MessdatenVonPatient(patient == $patient && (messdaten.taillenUmfang >= 94 && messdaten.
            taillenUmfang < 102))
        $kbp : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild.
            risikoMetabolischeKardiovaskulaereKomplikationen == "erhoeht"))
    then
        modify($kbp){getKrankheitsbild().setRisikoMetabolischeKardiovaskulaereKomplikationen("
            erhoeht")};
        System.out.println("Taillenumfang von " + $patient.getName() + " >= 94cm");
end

rule "Wenn der Taillenumfang von einer Frau >= 80cm beträgt so ist das Risiko auf Metabolische
     Kardiovaskuläre Komplikationen 'erhoeht'"
    salience 1
    agenda-group "group diagnose"
    when
        $patient : Patient(geschlecht == "female")
        $kbp : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild.
            risikoMetabolischeKardiovaskulaereKomplikationen == "erhoeht"))
        MessdatenVonPatient(patient == $patient && (messdaten.taillenUmfang >= 80 && messdaten.
            taillenUmfang < 88))

    then
        modify($kbp){getKrankheitsbild().setRisikoMetabolischeKardiovaskulaereKomplikationen("
            erhoeht")};
        System.out.println("Taillenumfang von " + $patient.getName() + " >= 80cm ");
end

/*-------------------------
    BMI berechnen
-------------------------*/
function double calcBMI(double gewichtInKg, double groesseInMeter){
    double bmi = gewichtInKg / (groesseInMeter * groesseInMeter);
    return bmi;
}

rule "Berechne den BMI des Patienten"

    agenda-group "group diagnose"
    when
        $patient : Patient()
        $messdatenVonPatient : MessdatenVonPatient($patient == patient && messdaten.bmi == 0.0 &&
            messdaten.gewichtInKg > 0.0 && messdaten.groesseInMeter > 0.0)
    then


        modify($messdatenVonPatient){getMessdaten().setBmi(calcBMI($messdatenVonPatient.getMessdaten
            ().getGewichtInKg(),$messdatenVonPatient.getMessdaten().getGroesseInMeter()))}

        System.out.println("BMI: " + $messdatenVonPatient.getMessdaten().getBmi());
end
```

```
/*-------------------------------------------------------------------
    Regeln zum festellen des Adipositas Grades, sowie des Risikos für Begleiterkrankungen
-------------------------------------------------------------------*/

rule "Wenn der BMI des Patienten kleiner als 18.5 ist, dann gehört der Patient der Kategorie '
     Untergewicht' an und hat ein 'niedriges' Risiko für Begleiterkrankungen des Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Untergewicht"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi < 18.5 && messdaten.bmi > 0.0)

   then
      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Untergewicht"),
                                 getKrankheitsbild().setRisikoBegleiterkrankungGrad(0)}

      System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
           getAdipositasGrad());
      drools.setFocus( "group therapie" );
end


rule "Wenn der BMI des Patienten >= 18.5 und < 25 ist, dann gehört der Patient der Kategorie '
     Normalgewicht' an und hat ein 'durchschnittliches' Risiko für Begleiterkrankungen des
     Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Normalgewicht"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 18.5 && messdaten.bmi < 25)

   then
      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Normalgewicht"),
                                 getKrankheitsbild().setRisikoBegleiterkrankungGrad(1)}



      System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
           getAdipositasGrad());
      drools.setFocus( "group therapie" );
end


rule "Wenn der BMI des Patienten >= 25 und < 30 ist, dann gehört der Patient der Kategorie '
     Präadipositas' an und hat ein 'gering erhöhtes' Risiko für Begleiterkrankungen des
     Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Präadipositas"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 25 && messdaten.bmi < 30)

   then
      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Präadipositas"),
                                 getKrankheitsbild().setRisikoBegleiterkrankungGrad(2)}



      System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
           getAdipositasGrad());
      drools.setFocus( "group therapie" );
```

```
end


rule "Wenn der BMI des Patienten >= 30 und < 35 ist, dann gehört der Patient der Kategorie '
     Adipositas Grad I' an und hat ein 'erhöhtes' Risiko für Begleiterkrankungen des Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Adipositas Grad I"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 30 && messdaten.bmi < 35)

   then


      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Adipositas Grad I")
           ,
                                getKrankheitsbild().setRisikoBegleiterkrankungGrad(3)}



      System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
           getAdipositasGrad());
      drools.setFocus( "group therapie" );
end


rule "Wenn der BMI des Patienten >= 35 und < 40 ist, dann gehört der Patient der Kategorie '
     Adipositas Grad II' an und hat ein 'hohes' Risiko für Begleiterkrankungen des Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Adipositas Grad II"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 35 && messdaten.bmi < 40)

   then

      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Adipositas Grad II
           "),
                                getKrankheitsbild().setRisikoBegleiterkrankungGrad(4)}



      System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
           getAdipositasGrad());
      drools.setFocus( "group therapie" );
end


rule "Wenn der BMI des Patienten >= 40 ist, dann gehört der Patient der Kategorie 'Adipositas Grad
      III' an und hat ein 'sehr hohes' Risiko für Begleiterkrankungen des Übergewichts"

   agenda-group "group diagnose"
   when
      $patient : Patient()
      $krankheitsbildVonPatient : KrankheitsbildVonPatient(patient == $patient && !(krankheitsbild
           .adipositasGrad == "Adipositas Grad III"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 40)

   then
      Krankheitsbild kb = $krankheitsbildVonPatient.getKrankheitsbild();
      modify($krankheitsbildVonPatient){getKrankheitsbild().setAdipositasGrad("Adipositas Grad III
           "),
                                getKrankheitsbild().setRisikoBegleiterkrankungGrad(5)}
```

```
        System.out.println("Gewicht: " + $krankheitsbildVonPatient.getKrankheitsbild().
            getAdipositasGrad());
        drools.setFocus( "group therapie" );
end
```

therapie_rules.drl

```
package de.DroolsGUI.adipositasRules


rule "Wenn Patient der Kategorie 'Normalgewicht' angehoert, dann werden folgende Maßnahmen
     vorgeschlagen"
    agenda-group "group therapie"
    when
        $patient : Patient()
        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "
            Normalgewicht")
    then

        System.out.println("Patient " + $patient.getName() + " hat ein Normales Gewicht. " + "Ziel
            ist es das Gewicht zu stabilisieren. "
                    + "Dazu sollte auf Gewichtsveränderungen geachtet werden um ggf. darauf
                        reagieren zu können. ");
end

rule "Wenn Patient der Kategorie 'Präadipositas' angehoert, dann werden folgende Maßnahmen
     vorgeschlagen"
    agenda-group "group therapie"
    activation-group "Präadipositas"
    when
        $patient : Patient()

        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "
            Präadipositas")
        MessdatenVonPatient(patient == $patient)
    then

        System.out.println("Patient " + $patient.getName() + " ist Präadipös. " + "Ziel ist es eine
            Gewichtszunahme zu verhindern. "
                    + "Dazu sollte auf Gewichtsveränderungen geachtet werden um ggf. darauf
                        reagieren zu können, "
                    + "sowie eine Beratung über gesundheitsförderlichen Lebensstil gegeben werden
                        . ");
end

rule "Wenn Patient der Kategorie 'Präadipositas' angehoert und einen Taillenumfang von w: >80 m:
     >94, dann werden folgende Maßnahmen vorgeschlagen"
    agenda-group "group therapie"
    activation-group "Präadipositas"
    salience 1
    when
        $patient : Patient()

        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "
            Präadipositas" && (krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "
            erhoeht" || krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich
             erhoeht"))
        MessdatenVonPatient(patient == $patient)
    then

        System.out.println("Patient " + $patient.getName() + " ist Präadipös. " + "Ziel ist es das
            Gewicht auf dauer um 5-10% zu verringern. "
                    + "Dazu sollte das Basisprogramm veranlasst werden, Risikofaktoren Management
                        , Therapie der Komorbiditäten. ");
end
```

```
rule "Wenn Patient der Kategorie 'Präadipositas' angehoert und einen Taillenumfang von w: >80 m:
     >94 und einen BMI > 27 , dann werden folgende Maßnahmen vorgeschlagen"
   agenda-group "group therapie"
   activation-group "Präadipositas"
   salience 2
   when
      $patient : Patient()

      KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "
          Präadipositas" && (krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "
          erhoeht" || krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich
           erhoeht"))
      MessdatenVonPatient(patient == $patient && messdaten.bmi >= 27)
   then

      System.out.println("Patient " + $patient.getName() + " ist Präadipös. " + "Ziel ist es das
          Gewicht auf dauer um 5-10% zu verringern. "
                  + "Dazu sollte das Basisprogramm veranlasst werden, Risikofaktoren Management
                      , Therapie der Komorbiditäten. "
                  + "Außerdem sollte nach frühestens 12wöchiger Therapie eine zusätzliche
                      medikamentöse Therapie erwogen werden. ");
end

rule "Wenn Patient der Kategorie 'Adipositas Grad I' angehoert, dann werden folgende Maßnahmen
     vorgeschlagen"
   agenda-group "group therapie"
   activation-group "Adipositas Grad I"
   when
      $patient : Patient()

      KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "Adipositas
          Grad I")
   then

      System.out.println("Patient " + $patient.getName() + " ist Adipös Grad I. " + "Ziel ist es
          das Gewicht auf dauer um 5-10% zu verringern. "
                  + "Dazu sollte das Basisprogramm veranlasst werden, sowie eine Beratung über
                      gesundheitsförderlichen Lebensstil gegeben werden. ");
end

rule "Wenn Patient der Kategorie 'Adipositas Grad I' angehoert und einen Taillenumfang von w: >88
     m: >102, dann werden folgende Maßnahmen vorgeschlagen"
   agenda-group "group therapie"
   activation-group "Adipositas Grad I"
   salience 1
   when
      $patient : Patient()

      KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "Adipositas
          Grad I" && krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich
          erhoeht")
   then

      System.out.println("Patient " + $patient.getName() + " ist Adipös Grad I. " + "Ziel ist es
          das Gewicht auf dauer um 5-10% zu verringern. "
                  + "Dazu sollte das Basisprogramm veranlasst werden, Risikofaktoren Management
                      , Therapie der Komorbiditäten. "
                  + "Bei keinem Erfolg sollte nach frühestens 12wöchiger Therapie eine
                      zusätzliche medikamentöse Therapie erwogen werden.");
end

rule "Wenn Patient der Kategorie 'Adipositas Grad II' angehoert, dann werden folgende Maßnahmen
     vorgeschlagen"
   agenda-group "group therapie"
   activation-group "Adipositas Grad II"
   when
      $patient : Patient()
```

```
        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "Adipositas
            Grad II")
    then

        System.out.println("Patient " + $patient.getName() + " ist Adipös Grad II. " + "Ziel ist es
            das Gewicht auf dauer um >=10% zu verringern. "
                    + "Dazu sollte das Basisprogramm veranlasst werden, sowie eine Beratung über
                        gesundheitsförderlichen Lebensstil gegeben werden.");
end

rule "Wenn Patient der Kategorie 'Adipositas Grad II' angehoert und einen erhöhtes Risiko für
     Begleiterkrankungen hat, dann werden folgende Maßnahmen vorgeschlagen"
    agenda-group "group therapie"
    activation-group "Adipositas Grad II"
    salience 1
    when
        $patient : Patient()

        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "Adipositas
            Grad II" && krankheitsbild.risikoMetabolischeKardiovaskulaereKomplikationen == "deutlich
             erhoeht")
    then

        System.out.println("Patient " + $patient.getName() + " ist Adipös Grad II. " + "Ziel ist es
            das Gewicht auf dauer um 10-20% zu verringern. "
                    + "Dazu sollte das Basisprogramm veranlasst werden, Risikofaktoren Management
                        , Therapie der Komorbiditäten. "
                    + "Bei keinem Erfolg sollte nach frühestens 12wöchiger Therapie eine
                        zusätzliche medikamentöse Therapie erwogen werden."
                    + "Bei erfolgloser konservativer Therapie, können chirurgische Maßnahmen
                        erwogen werden.");
end

rule "Wenn Patient der Kategorie 'Adipositas Grad III' angehoert, dann werden folgende Maßnahmen
     vorgeschlagen"
    agenda-group "group therapie"
    when
        $patient : Patient()

        KrankheitsbildVonPatient(patient == $patient, krankheitsbild.adipositasGrad == "Adipositas
            Grad III")
    then

        System.out.println("Patient " + $patient.getName() + " ist Adipös Grad III. " + "Ziel ist es
             das Gewicht auf dauer um 10-30% zu verringern. "
                    + "Dazu sollte das Basisprogramm veranlasst werden, Risikofaktoren-Management
                        , Therapie der Komorbiditääten. "
                    + "Bei keinem Erfolg sollte nach frühestens 12wöchiger Therapie eine
                        zusätzliche medikamentöse Therapie erwogen werden. "
                    + "Bei erfolgloser konservativer Therapie kann eine chirurgische Therapie
                        erwogen werden.");
end
```

Literaturverzeichnis

[1] Becker, P.: *Grundlagen von Decision Support und Expertensystemen.* http://www2.inf.fh-rhein-sieg.de/~pbecke2m/ki-master/ruleengine.pdf, Besucht am 9.9.2012

[2] I. Boersch, J. Heinsohn, R. Socher: *Wissensverarbeitung - Eine Einführung in die Künstliche Intelligenz*, Elsevier/Spektrum Akademischer Verlag, 2. Auflage 2007

[3] *Drools Expert User Guide* Version 5.4.0.CR1 http://docs.jboss.org/drools/release/5.4.0.CR1/drools-expert-docs/html/, Besucht am 9.9.2012

[4] Zuleger, C.: Bachelor Thesis: *Konzeption und Implementation eines Expertensystems zur Überwachung der Sensorfertigung bei der Firma epro GmbH*

[5] Rete Algorithm: http://legacy.drools.codehaus.org/Rete, Besucht am 9.9.2012

[6] *Drools RETEOO:* salaboy.com/2011/06/06/drools-reteoo-for-dummies-1-intro/, Besucht am 20.9.2012

[7] *Drools Wikipedia:* http://en.wikipedia.org/wiki/Drools, Besucht am 12.9.2012

[8] *Fettsucht-Welle rollt mindestens bis 2050*: http://www.spiegel.de/wissenschaft/medizin/usa-fettsucht-welle-rollt-mindestens-bis-2050-a-727431.html, Artikel vom 5.11.2010

[9] *Nationale Verzehrsstudie II*: Herausgeber: Max Rubner-Institut, Bundesforschungsinstitut für Ernährung und Lebensmittel

[10] *Leitlinie zur Prävention und Therapie der Adipositas:* Herausgeber: Deutsche Adipositas Gesellschaft, Deutsche Diabetes-Gesellschaft, Deutsche Gesellschaft für Ernährung, Deutsche Gesellschaft für Ernährungsmedizin, 2007

[11] *Kardiovaskuläres System*: http://de.wikipedia.org/wiki/Blutkreislauf

[12] *Paul Browne*: JBoss Drools Business Rules, Packt Publishing Ltd., 2009

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Abschlussarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Brandenburg an der Havel, den 20.09.2012

Jonas Preckwinkel