Regelbasierte Systeme mit JBoss Drools

Paul Weinhold

JUG Görlitz, 29.07.2015


I will follow the rules
I will follow the rules
I will follow the rules
I will follow the rules
I will fol
TheVirtualPresenter.com

Warum regelbasierte Systeme?

- Regeln gibt es überall
- Regeln sind ein Bestandteil der Fachlichkeit
- Menschen denken in Regeln
  - Regeln werden (meist) leicht verstanden
- Regeln obliegen Änderungen (Wartbarkeit)
  - Hinzufügen
  - Entfernen
  - Überarbeiten

Regeln

“Wenn die Ampel grün ist,
dann gehe ich über die Straße”

Regeln

“**Wenn** die Ampel grün ist,
dann gehe ich über die Straße”

Regeln

“WENN … DANN …“ - Form

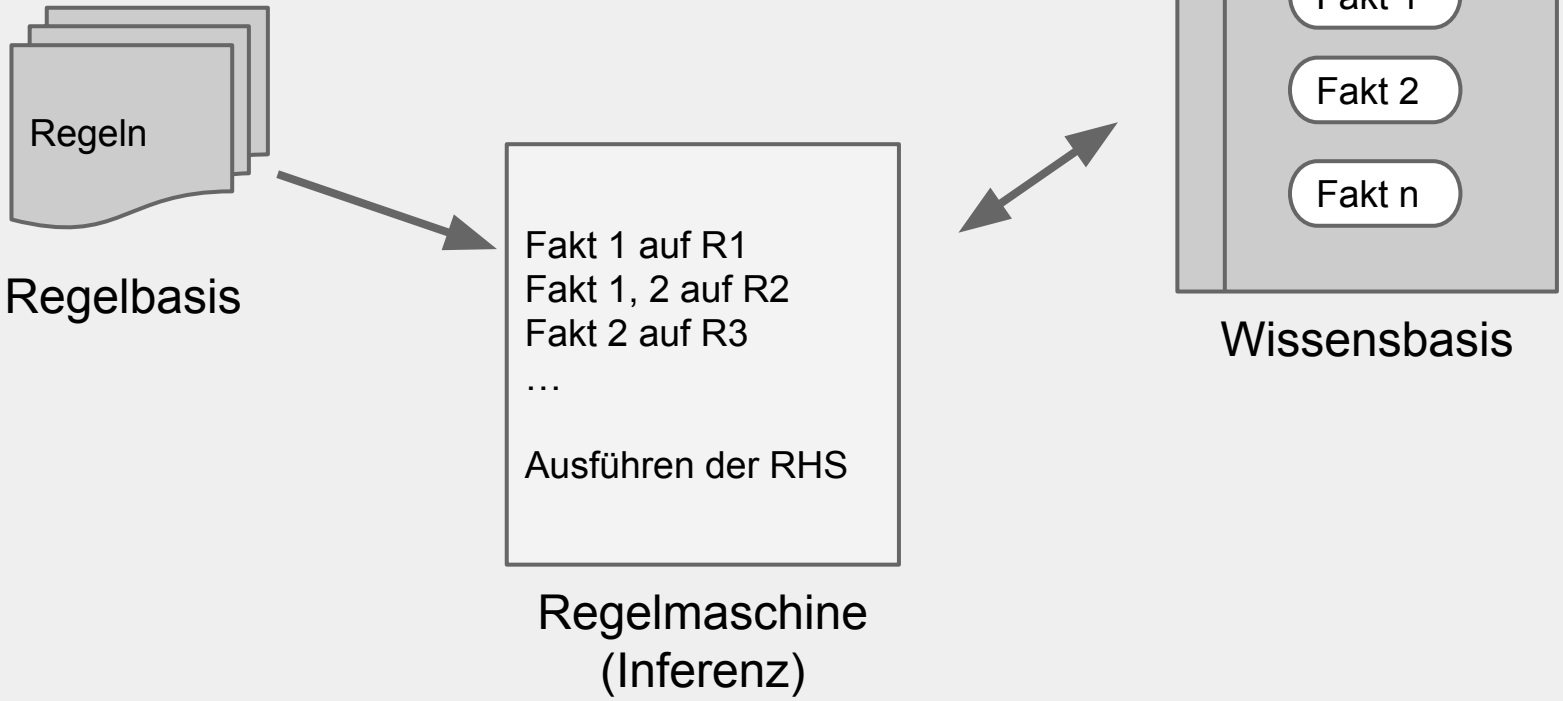
Bedingung
Prämise
Left-hand side (LHS)

Bedingungen prüfen
Fakten

Konsequenz
Konklusion
Right-hand side (RHS)

Aktion die bei Erfüllung ausgeführt wird

Regelsystem



Wie implementiere ich Regeln?

Wie implementiere ich Regeln?

Imperativer Stil

```
if (customer.getLevel() == Level.GOLD) {
  //do something for Gold customer
} else if (customer.getLevel() == Level.SILVER) {
  if (customer.getAccounts().isEmpty()) {
    //do something for Silver customer with no accounts
  } else {
    for (Account account : customers.getAccounts()) {
      if (account.getBalance() < 0) {
        //do something for Silver customer that
        //has account with negative balance
      }
    }
  }
}
```

Imperativer Stil

- Nicht immer einfach zu schreiben
- mutiert schnell zu Spaghetti-Code
- Fachlicher Code vermengt mit Infrastruktur
- Schwer zu ändern
- Von Domainexperten nicht überprüfbar und nicht änderbar

Deklarativer Stil

```
if Customer(level == Level.GOLD)
//then do something for Gold customer
```

```
if Customer(level == Level.SILVER, accounts.empty)
//then do something for Silver customer with no accounts
```

```
if Customer(level == Level.SILVER),
   Account(balance < 0, customer.accounts contains this)
//then do something for Silver customer that
//has account with negative balance
```

Deklarativer Stil

- Je Anforderung eine Regel
  - Einfacher zu lesen
  - Geschäftsregeln können natürlicher abgebildet werden
- Im Gegensatz zum Imperativen Stil muss der Code nicht explizit für jeden Kunden evaluiert werden
  - Die Regelmaschine nimmt die Kunden-Instanzen aus der Wissensbasis (in Drools: working memory)

Vorteile einer Regelmaschine

- Einfach zu Verstehen und zu modifizieren für ...
  - neue Entwickler im Team
  - Business Analysten
- Einfache Wartung
  - Regeln sind voneinander unabhängig
- Steigende Komplexität ist beherrschbar
- Abstraktion der Ausführung für eine bessere Performance
  - effiziente Algorithmen (Rete)
  - Parallel Execution

Vorteile eine Regelmaschine

- Trennung des fachlichen Codes vom Rest
- Einfache Wiederverwendbarkeit
- Änderung der Regeln ohne Redeploying

Nachteile einer Regelmaschine

- Kein Wundermittel: Die zu lösenden Probleme bleiben gleich schwer
- Entwickler müssen geschult werden
  - Mangelndes Hintergrundwissen kann ineffiziente Regeln zur Folge haben
- Entwickler muss anders Denken
- Komplexeres Debugging
- Höherer Speicherverbrauch

Wann sollte keine Regelmaschine verwendet werden?

- Kleine Projekte mit weniger als 20 Regeln
- Wenn die Geschäftslogik “in Stein gemeißelt” ist
- Wenn es nur einfache Regel sind
  - nur über EIN Fakt
  - mit nicht mehr als 2 if-then Ausdrücken beherrschbar
- Wenn Performance das Hauptziel ist (Videocodec)
- Wenn das Projekt nicht weitergepflegt wird

Wann einsetzen?

- Problem für imperative Algorithmen ungeeignet
  - zu komplex
  - nicht vollständig verstanden / lösbar
- Fachlogik im ständigen Wandel
- Domainexperten sollen Wissen einpflegen, können aber nicht entwickeln
- Beispielanwendung:
  - Geschäftsregeln
  - Fehlererkennung
  - Betrugserkennung bei Geldtransfers

JBoss Drools

a Business Rules Management System (BRMS)

- 100% Java
- Als Application Server oder als Bibliothek
- Aktuelle Version: 6.2.0
- Apache 2.0 Lizenz
- von Red Hat JBoss
- Enterprise Edition mit Support
- Seit 2001

JBoss Drools

Bestandteile:

- Drools Expert - Core Business Rules Engine (BRE)
- Drools Workbench - Web UI for authoring and management
- Drools Fusion - Complex Event Processing (CEP)
- jBPM Integration
- Eclipse Plugin

JBoss Drools

```
<dependency>
    <groupId>org.drools</groupId>
    <artifactId>drools-core</artifactId>
    <version>6.2.0.Final</version>
</dependency>
```

JBoss Drools

- Implementation des Rete-Alogrithmus
- Regeln werden mit einer DSL definiert (.drl)
  - Alternativ als XML (Relikt aus alten Versionen)
- Entscheidungstabellen Support (Excel-Tabellen)
- Definition eigener DSLs für Regeln
- Fakten können beliebige Objekte sein (POJO)
- Fakten im Working Memory
  - Stateful
  - Stateless

Auf gehts … Regel Syntax - LHS

- Jede Regel besteht aus 1 oder mehreren Bedingungen

```
Account ( balance == 200)
Customer ( name == "John")
```

Accounts × Customer

Rechenbeispiel:

3 Accounts mit balance = 200 und 2 mit <> 200

5 Customers die “John” heißen und 3 nicht

Auf gehts … Regel Syntax - LHS

- Jede Regel besteht aus 1 oder mehreren Bedingungen

```
Account ( balance == 200)
Customer ( name == "John")
```

Accounts × Customer

Rechenbeispiel:

3 Accounts mit balance = 200 und 2 mit <> 200

5 Customers die “John” heißen und 3 nicht

-> Regel wird 15mal gefeuert

Regel Syntax - LHS

- And/Or Conditions

```
Customer ( name == "John", age < 26)


Customer (name == "John" || age < 26)
```

- Variablen

```
$a: Account ()
$c: Customer ( accounts contains $a)
```

Regel Syntax - LHS

- mvel - Support (https://github.com/mvel/mvel)

```
$customer.name //getName()

$customer.address.postalcode

//Null-safe bean navigation
$customer.?address.postalCode
```

Regel Syntax - LHS

- not element

```
not Account( type == Account.Type.SAVINGS)
```

- exists element

```
exists Account( type == Account.Type.SAVINGS)
```

- eval element

Regel Syntax - LHS

- Object Identities/equalities

Equals:

```
$customer1: Customer()
$customer2: Customer( this != $customer1)
```

Identity:

```
$customer1: Customer()
$customer2: Customer()
eval($customer2 != $customer1)
```

Regel Syntax - RHS

- Wird aus ausgeführt wenn LHS matched
- Beliebiger Java oder mvel Code
  - Ziel so wenig wie möglich Code
  - Komplexe Konsequenzen auslagern
- Imperativer-Stil
- Bad Practice: (if statements)
- Aufgabe:
  - Fakten modizieren, hinzufügen und entfernen

Regel Syntax - RHS

- modify

```
rule "reset accounts"

 when
  $account: Account( balance > 0)

 then
  modify($account) {
    setBalance(0)
  }
end
```

Regel Syntax - RHS

- insert

```
rule "create accounts"

 when
  $customer: Customer()
  not Account( customer = $customer)
 then
  Account account = new Account();
  account.setBalance(10);
  account.setCustomer($customer);
  insert(account);
end
```

Regel Syntax - RHS

- retract

```
rule "retract accounts"

 when
  $account: Account( balance > 1000)

 then
  retract($account);
end
```

Regelmaschine aufsetzen

```
StatelessKieSession ksession = KieServices.Factory.get()
                .getKieClasspathContainer()
                .newStatelessKieSession("TestKS");

        Customer customer1 = new Customer("John", 35);
        Customer customer2 = new Customer("Tom", 23);

        ksession.execute(Arrays.asList(customer1, customer2));
```

Stateless vs Stateful Session

- Stateless
  - Alle Fakten werden vor der Ausführung der Regeln in die Wissensbasis geschrieben
  - Ein erneutes starten der Regelmaschine führt zu einer neuen Session
- Stateful
  - Fakten verbleiben in der Wissensbasis (auch die in RHS hinzugefügten)
  - Erneutes Ausführen auf Basis des letzten Standes

Live coding

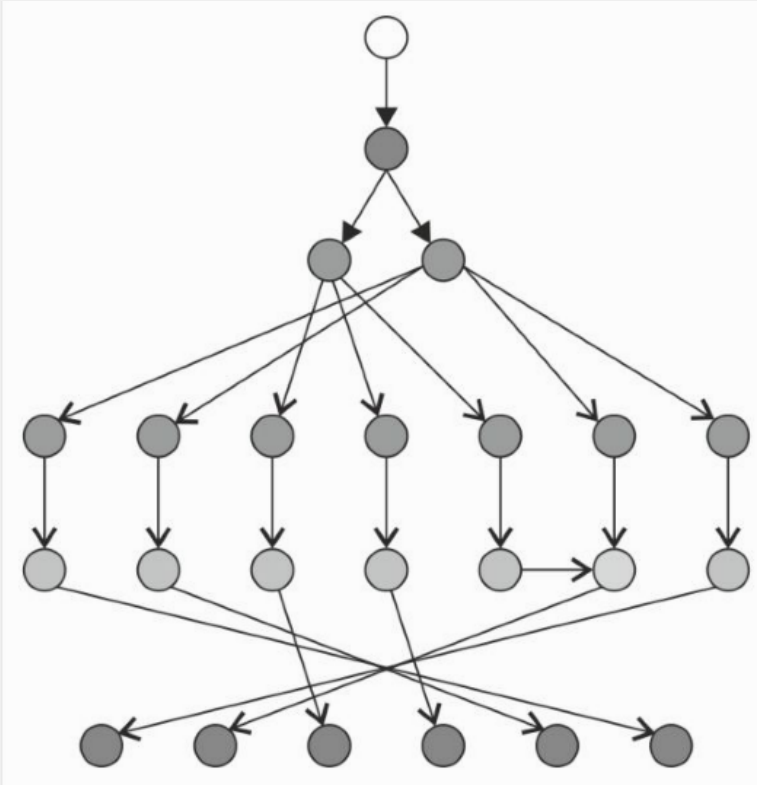
Rete - Algorithmus

- rete - lat. Netz, Netzwerk
- Wikipedia:

  > “ein Algorithmus und Expertensystem zur Mustererkennung und zur Abbildung von Systemprozessen über Regeln”

- Abbildung der Regeln in einem DAG

Rete - Algorithmus

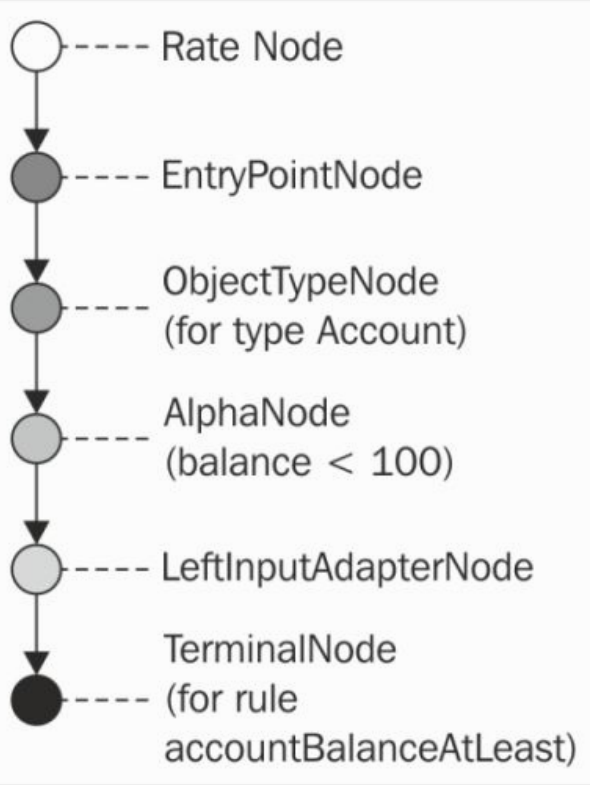


Rete - Algorithmus

```
rule "accountBalanceAtLeast"

 when
  $account: Account( balance < 100)
 then
  //do something
end
```

Rete - Algorithmus

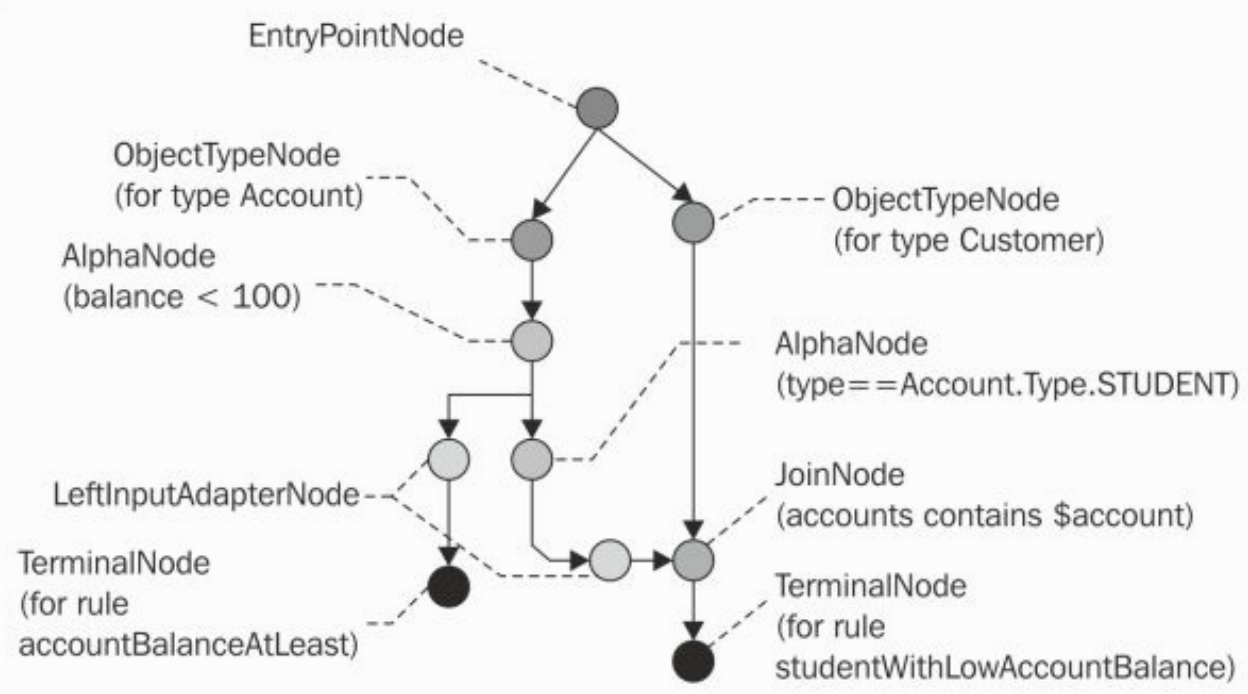
```
rule "accountBalanceAtLeast"

 when
  $account: Account( balance < 100)
 then
  //do something
end
```



Rete - Algorithmus

```
rule "studentWithLowAccountBalance"
 when
  $account: Account( balance < 100, type == Account.Type.STUDENT)
  $customer: Customer( accounts contains $account)
 then
  //do something
end
```

Rete - Algorithmus



Vielen Dank

Fragen?

Literatur Hinweise