



camunda
The Business Process Company

**Geschäftsprozesse und Regeln mit
jBPM und Drools**

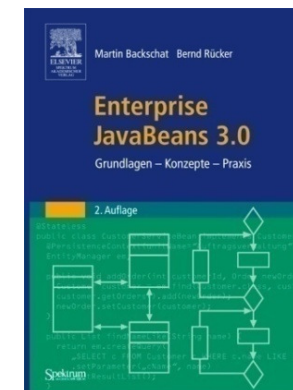
ein unschlagbares Team

Bernd Rücker

Wer bin ich?

- Berater, Trainer, Coach
- Softwareentwickler
- Committer im JBoss jBPM-Projekt
- Themen: BPM, SOA, Process Execution (jBPM, BPEL, XPD, ...), Java EE
- Eigene Trainings zu Process Execution, BPMN, BPM-Software, ...

camunda
The Business Process Company



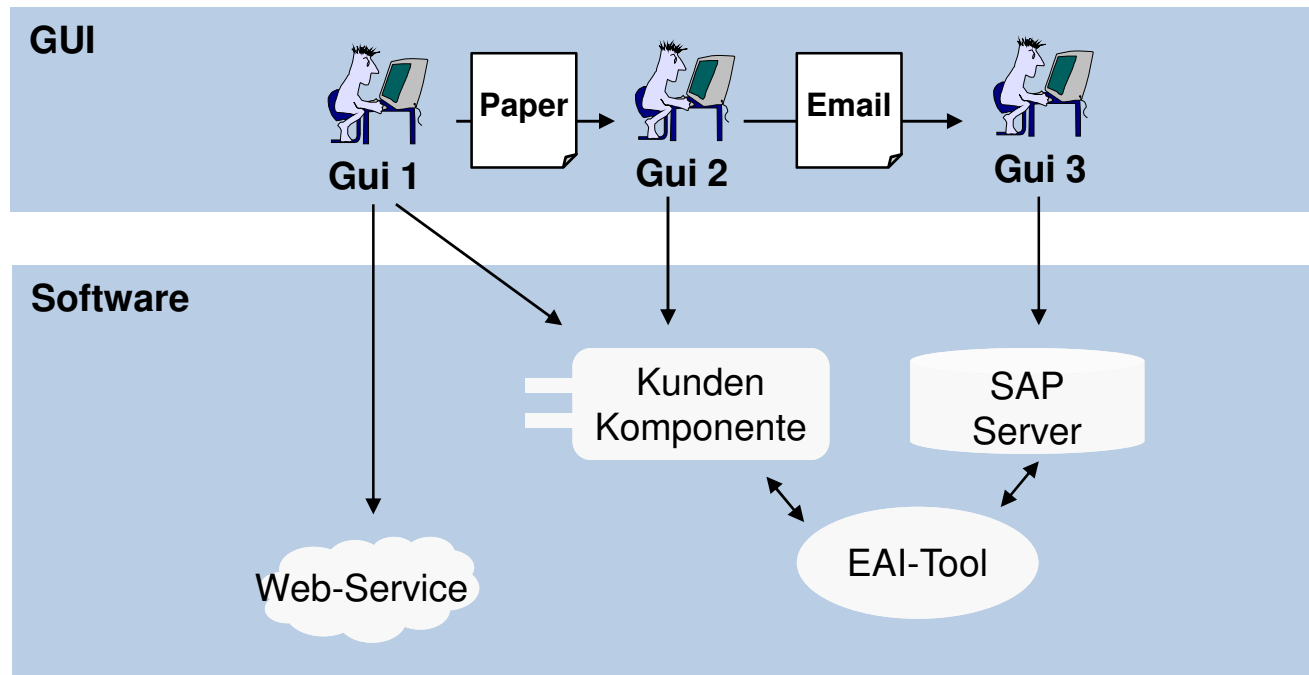
Agenda

Geschäftsprozess und Regeln

- Business Process Management (BPM)
- JBoss jBPM
- Business Rules Management (BRM)
- JBoss Drools
- Einsatzgebiete, Abgrenzung und Kombinationsmöglichkeiten

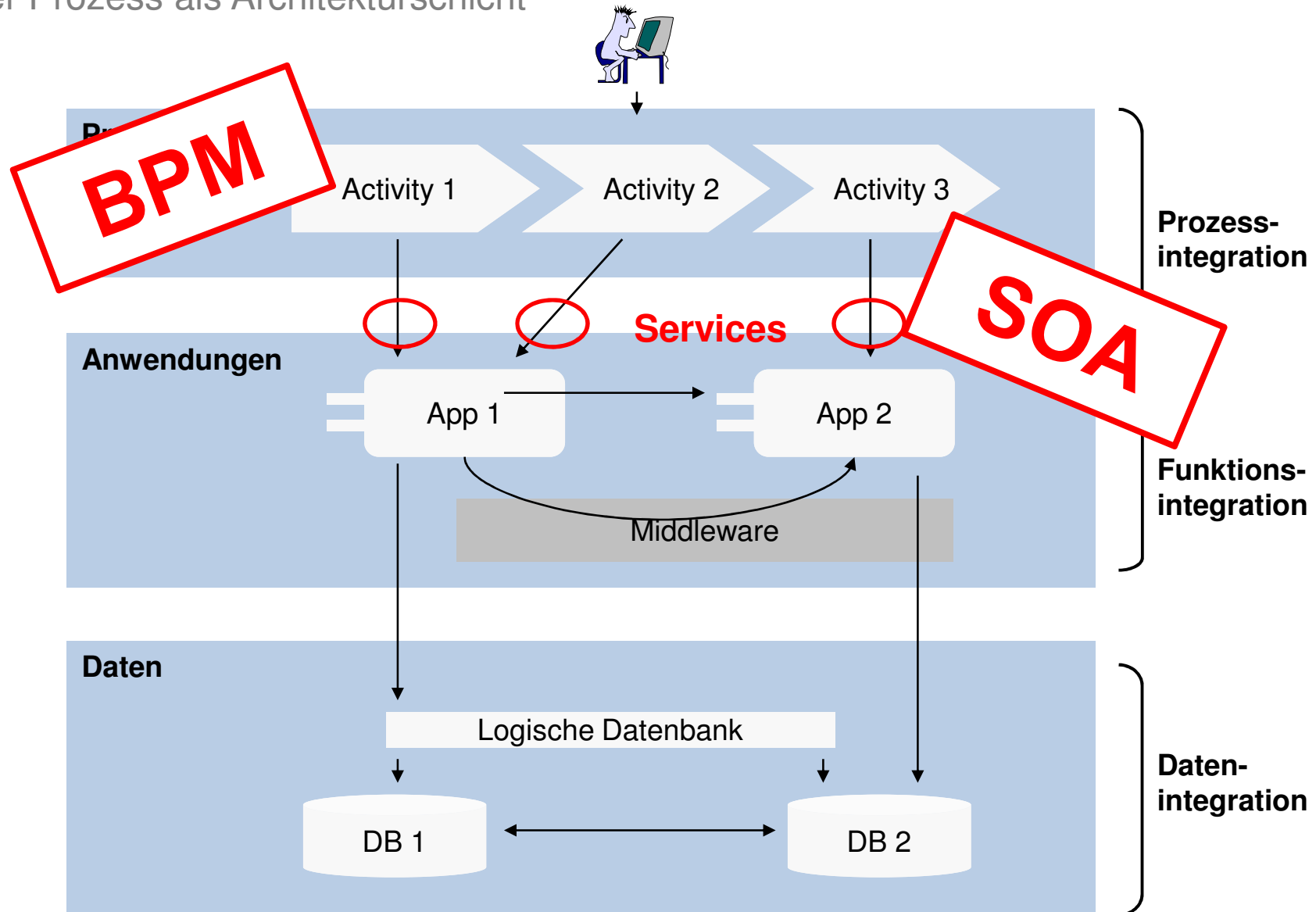
Geschäftsprozesse & Software

Der Status Quo in manchen Unternehmen



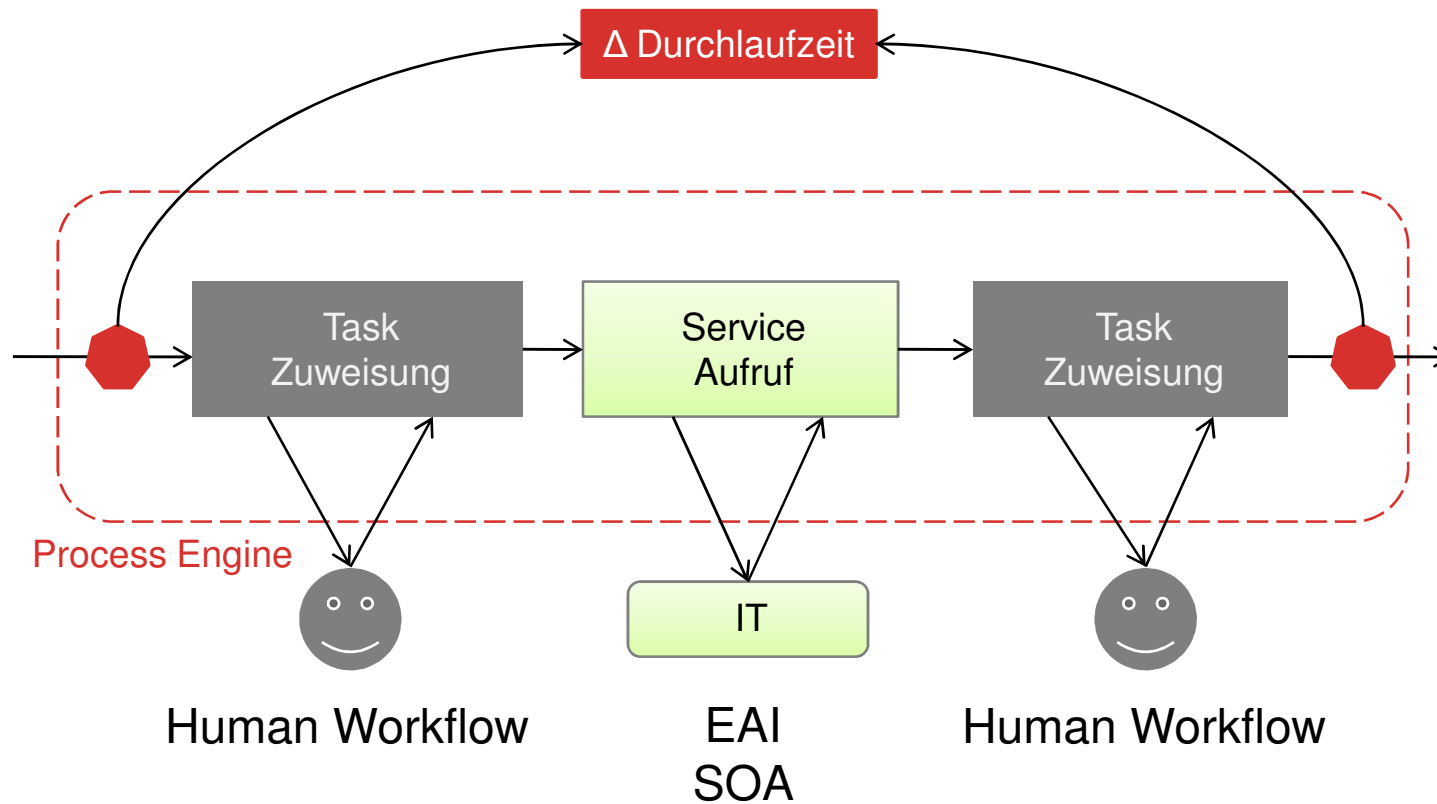
Softwareintegration

Der Prozess als Architekturschicht



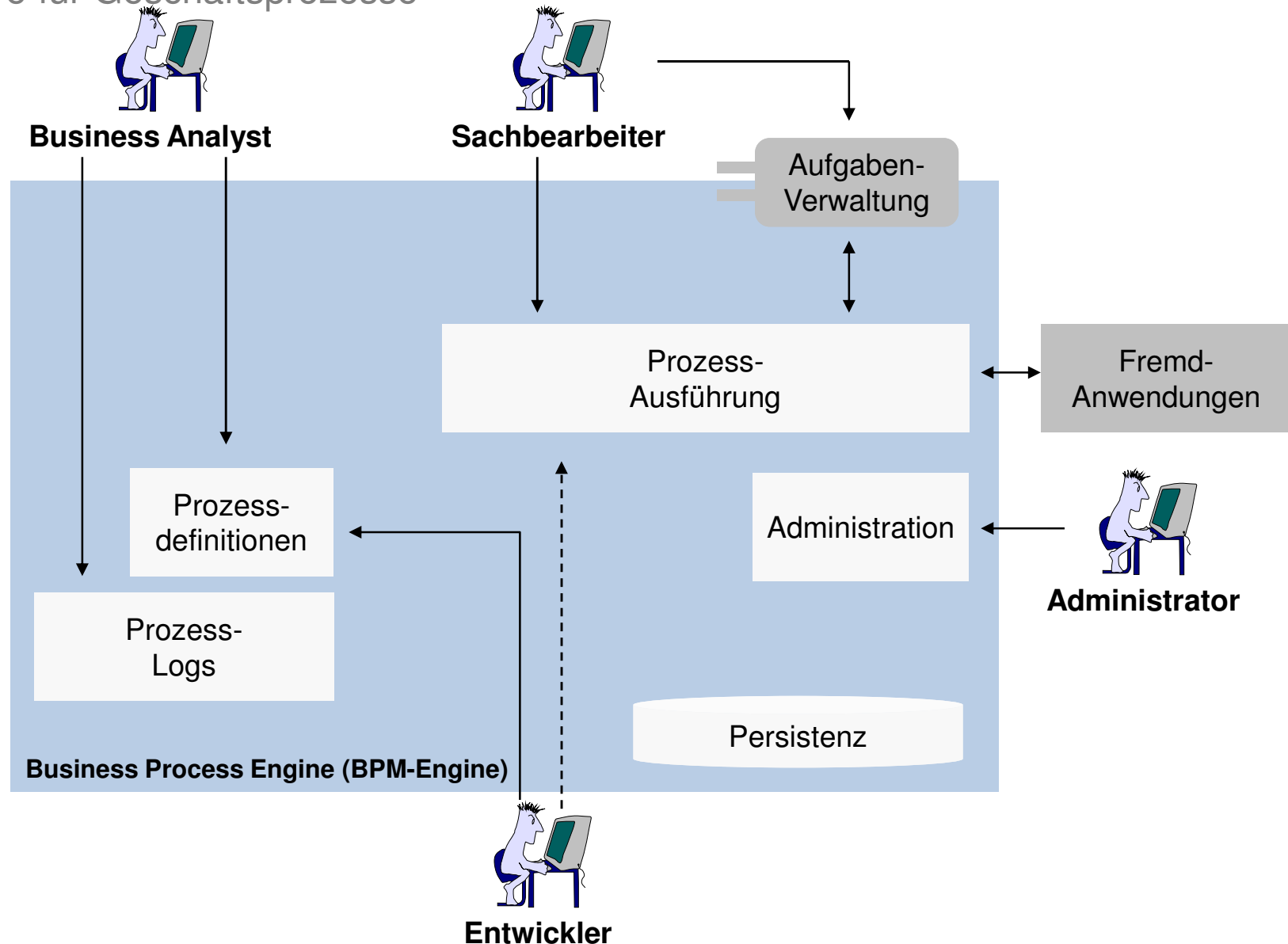
Ein „digitaler“ Prozess

mit Business Process Engine



Business Process Engine

Middleware für Geschäftsprozesse



Was leistet die Business Process Engine

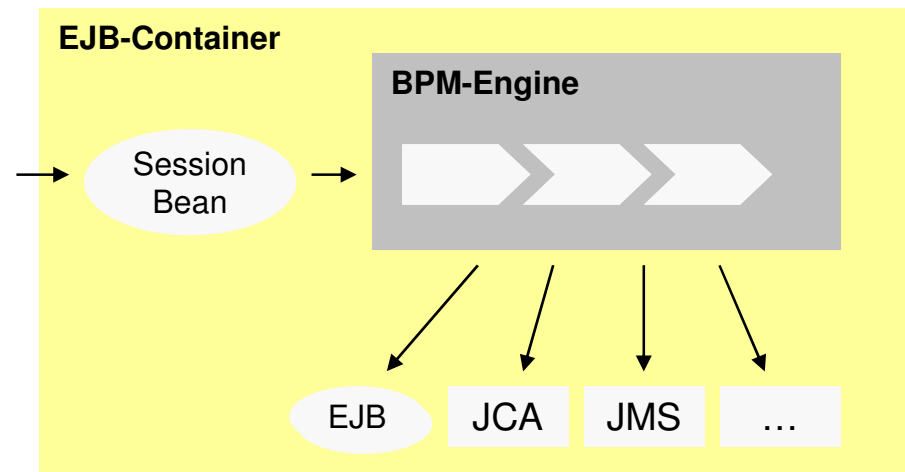
Features

- Versionierung, Persistenz & Interpretation von Prozessmodellen
- Steuerung & Persistenz von Prozessinstanzen
- Task-Management & Wait-States
- Prozesskontext (Variablen zu Prozess speichern)
- Einbindung externer Services
- Verwalten von Ereignissen (wie Timeouts, ...)

Business Process Engine in Java

Architektur

- Process Engine ist eigene Architekturschicht
- Domänenobjekte oder Referenzen als Prozessvariablen
- Ansteuerung ext. Services



JBoss jBPM

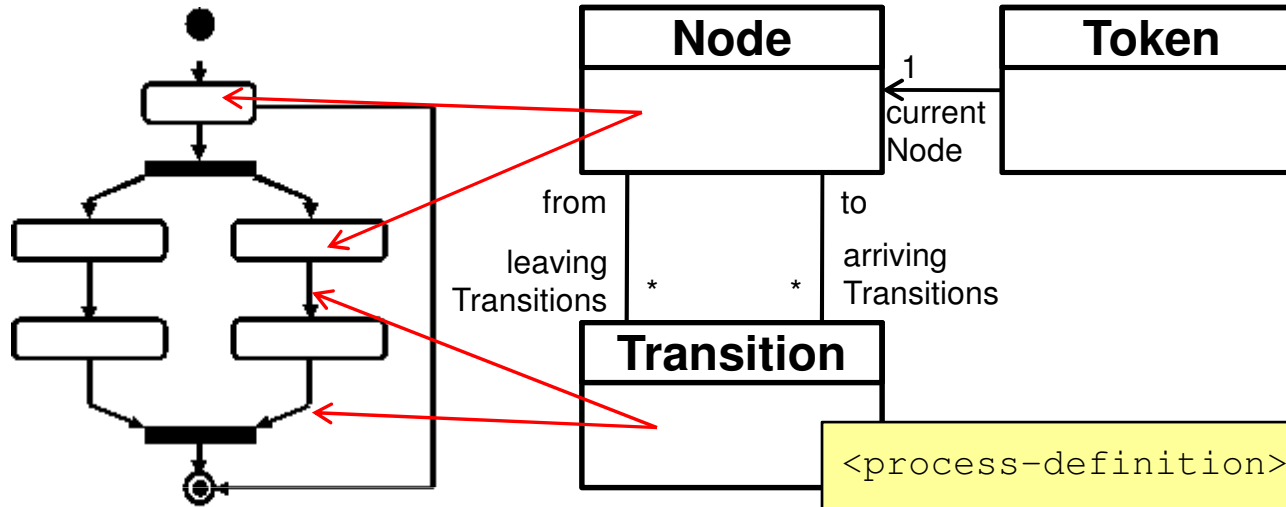
Open Source Process Execution



- Business Process Engine
- POJO-Kern: Interne Prozessrepräsentation durch Java-Modelle
- Persistenz über Hibernate (DB-Unabhängigkeit)
- Lauffähig mit oder ohne Application-Server
- „Library“
- Klein und flexibel, leicht erweiterbar
- Aktuell Version 3.2, Version 4 in der Entwicklung
- Open Source (LGPL)

„Graph oriented programming“

jBPM in a nutshell



```
<process-definition>
...
<node name="serve client">
  <transition name="ok" to="order" />
  <transition name="nok" to="joke" />
</node>
<node name="order" />
<node name="joke" />
...
</process-definition>
```

Process Execution

Token

The diagram shows a yellow background representing the execution phase. A red dot labeled "Token" is positioned on the left. Four red arrows point from this token to the four main stages of the process definition shown in the adjacent block: "Submit article", "Check for dirty words", "Add to pdf", and "end".

Process Definition

The diagram shows a blue background representing the process definition phase. It contains a vertical flow of process elements:

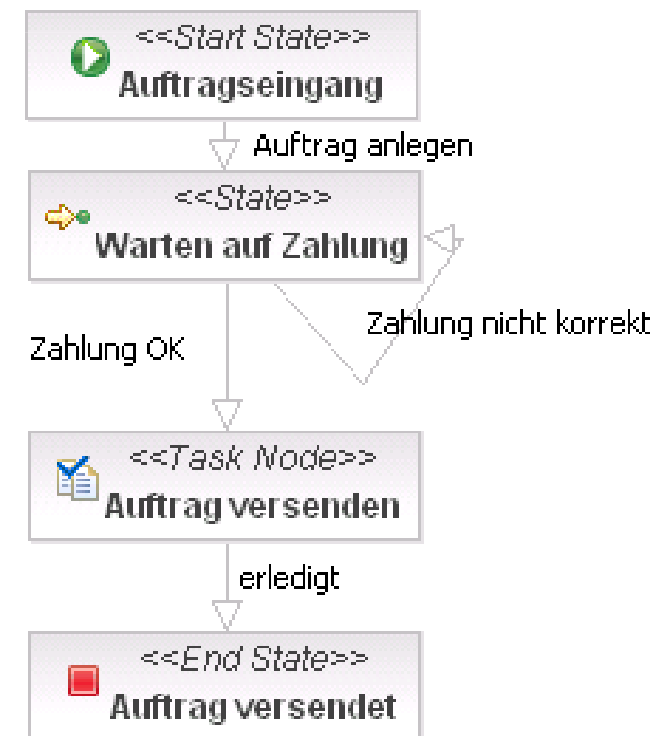
- Start State**: "Submit article" (with a green play button icon).
- Task Node**: "Check for dirty words" (with a document icon).
- Node**: "Add to pdf" (with a gear icon).
- End State**: "end" (with a red stop sign icon).

Arrows indicate the flow from "Submit article" to "Check for dirty words", then to "Add to pdf" (labeled "OK"), and finally to "end".

Verschiedene Node-Typen

jBPM in a nutshell

- Task-Node: Human Tasks / Aufgaben
- State: Wait-States
- Fork / Join
- Decision: Automatische Entscheidung
- Start-State / End-State
- ...
- Eigene Node-Typen mit Verhalten können implementiert werden



- Einfache Java-API zur Steuerung der Engine
 - Prozessstart
 - Aufgabenliste
 - ...
- Aufrufen von „User-Code“
 - definierte Stellen im Prozess
 - Interface & Java-Klassen

jBPM in a nutshell

```
JbpmConfiguration conf = JbpmConfiguration.getInstance();
JbpmContext context = conf.createJbpmContext();

ProcessInstance pi = context.getGraphSession().
    findLatestProcessDefinition("Ticket").createProcessInstance();
pi.getRootToken().signal();

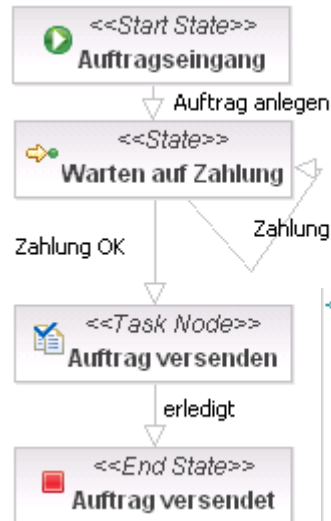
List<TaskInstance> tasks = context.getTaskMgmtSession().
    findTaskInstances("Vertrieb");
tasks.get(0).end("Ticket schliessen");

context.close();
```

```
public class MyAction implements ActionHandler {
    public void execute(ExecutionContext ctx) {
        Object var = ctx.getVariable("var");
        result = service.doSomething(var);
        ctx.setVariable("result", result);
    }
}
```



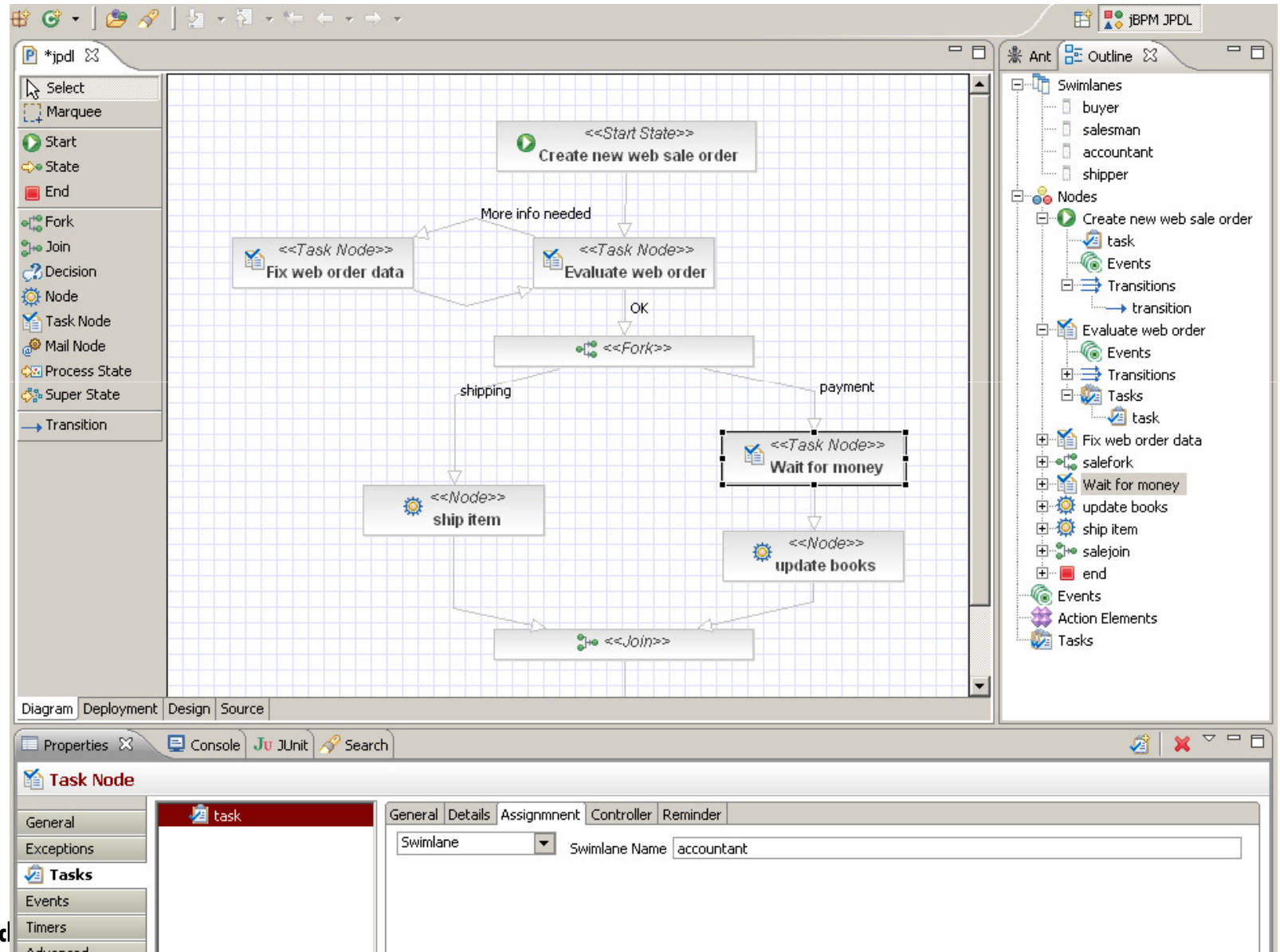
Gemeinsame Sprache, Beispiel jBPM



```
<process-definition name="SimpleOrder">
  <swimlane name="Lager">
    <assignment pooled-actors="Lager" />
  </swimlane>
  <start-state name="Auftragseingang">
    <event type="node-leave">
      <action name="create order in business logic" class="com.camunda.jmgui.actions:
    </event>
    <transition name="Auftrag anlegen" to="Warten auf Zahlung"></transition>
  </start-state>
  <state name="Warten auf Zahlung">
    <transition name="Zahlung OK" to="Auftrag versenden"></transition>
    <transition name="Zahlung nicht korrekt" to="Warten auf Zahlung"></transition>
  </state>
  <task-node name="Auftrag versenden">
    <task name="Auftrag versenden" swimlane="Lager" />
    <event type="node-leave">
      <action name="set order shipped status in business logic" class="com.camunda.:
    </event>
    <transition name="erledigt" to="Auftrag versendet"></transition>
  </task-node>
  <end-state name="Auftrag versendet" />
</process-definition>
```

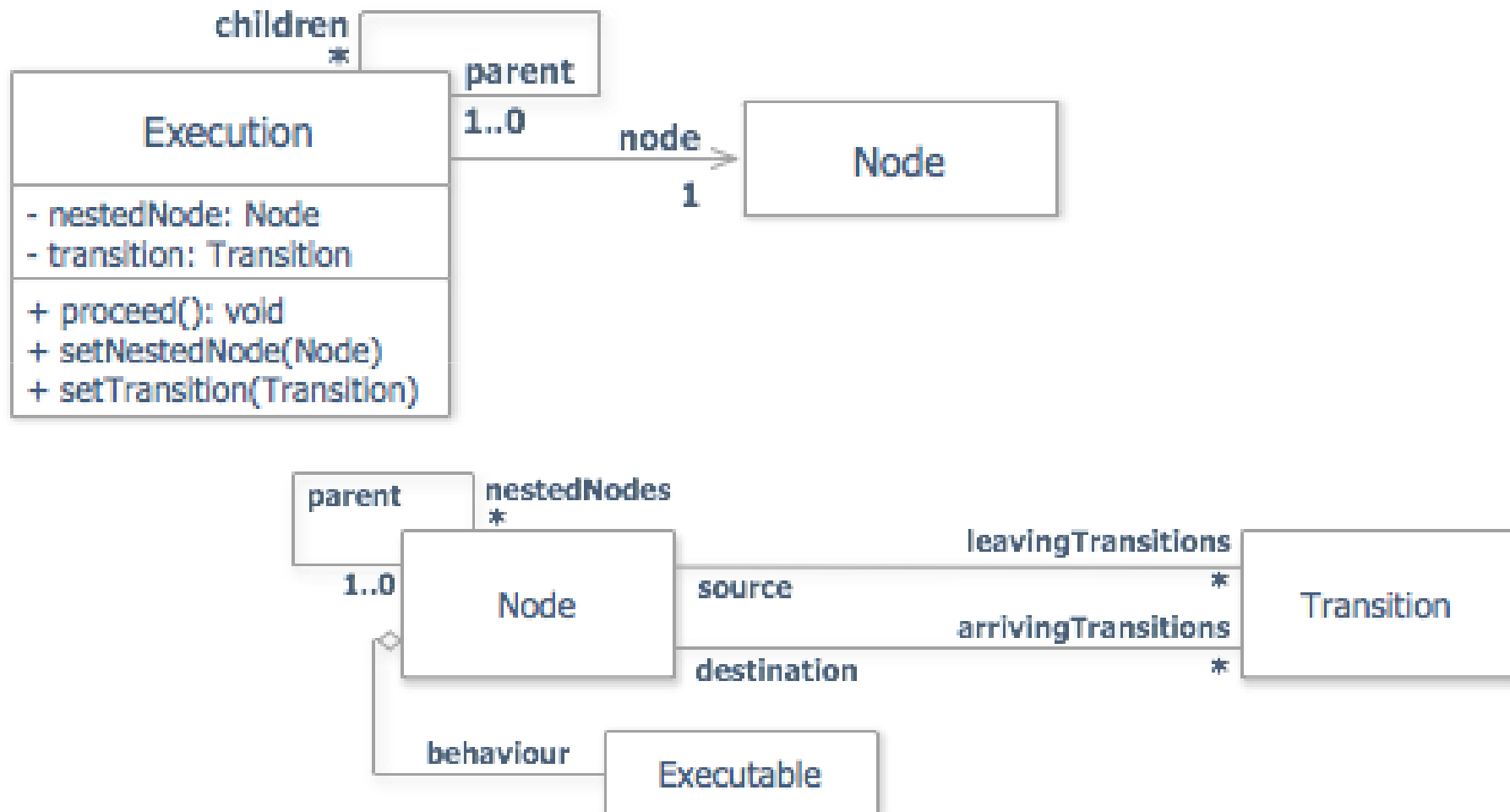
Tooling: Eclipse

jBPM in a nutshell



Die Zukunft: Process Virtual Machine (PVM)

Sprachen werden aufbauend auf der PVM entwickelt



Was sind Regeln?

- „Wenn ich müde bin, dann gehe ich ins Bett!“
- „WENN .. DANN ..“-Struktur
- Bedingung und Konsequenz (Prämisse und Konklusion; Left-Hand-Side LHS und Right-Hand-Side RHS)
- Konsequenz wird häufig als „Aktion“ bezeichnet
- Bedingungen prüfen „Fakten“
- Regeln „feuern“, wenn deren Bedingung eintrifft

Wie werden Regeln umgesetzt?

Alternativen

- Direkte Programmierung im Quellcode:

```
if ( person.istMuede() == true ) {  
    person.putzeZaehne();  
    person.geheInsBett();  
}
```

- Spezifische Lösungen (Codegenerierung, DSL, Speziallösungen, ...)
- Regelmaschine / Rule Engine

Probleme programmierter Regeln

Wie werden Regeln umgesetzt?

- Wartbarkeit und Validierbarkeit nicht gegeben
- Regeln müssen durch Entwickler in Quellcode übersetzt werden
- Fachliche Regeln werden über verschiedene Klassen verteilt
- Keine Lesbarkeit der Regeln für den Fachbereich
- Konflikt-Lösung muss realisiert werden

Vorteile der Rule-Engine

Wie werden Regeln umgesetzt?

- Explizite Formulierung der Regeln als Regeln
- Deklarativ: Welche Regeln wann wie ausgeführt werden entscheidet die Regelmaschine
- Regeln für Fachbereich verständlich

Bedingung:

```
Person.muede = true
```

Konsequenz:

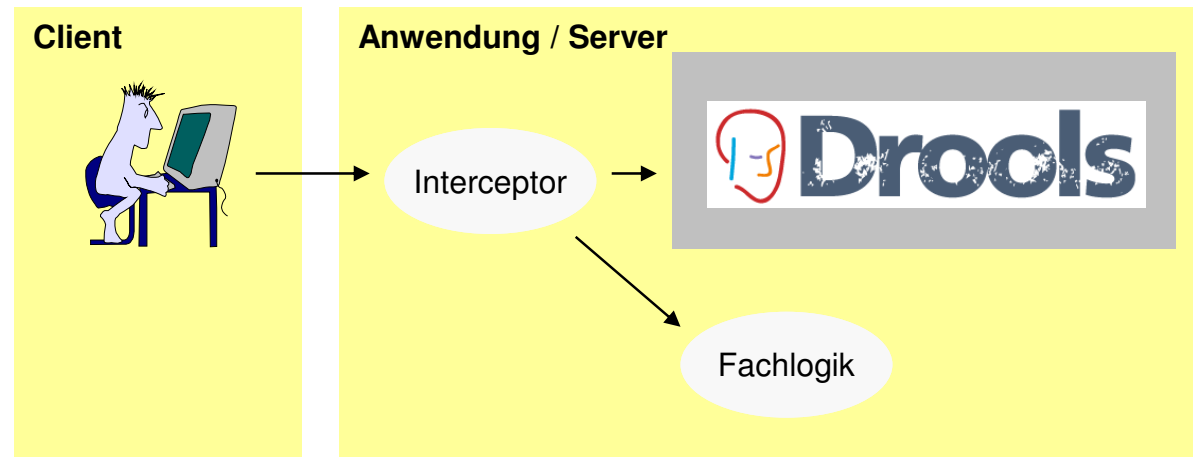
```
person.putzeZaehne();
```

```
person.geheInsBett();
```

Rule Engines in Java

Architektur

- Fakten (Wissen) = Domänenobjekte (POJOs)
- Rule Engine wird generisch in die Architektur integriert (Interceptoren, ...)



JBoss Drools

Die Open Source Rule Engine

- Java Rule Engine (RETE-Implementierung)
- „JBoss Drools“ / „JBoss Rules“
- Lauffähig mit oder ohne Application-Server
- „Library“
- Business Rules Management System (BRMS) in der Entwicklung
- Aktuell Version 4.0
- Open Source (ASL)



Drools-Regeln

Beispiel

```
package com.camunda.demo

import demo.business.*;
import demo.infrastructure.ErrorList;

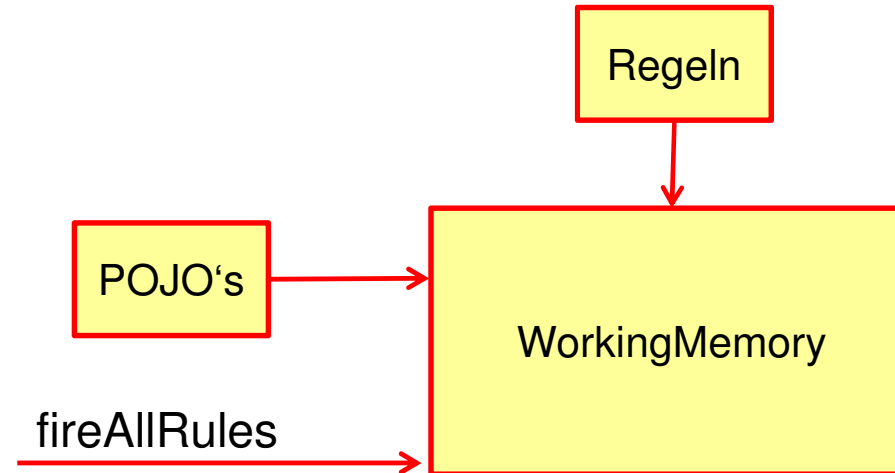
global ErrorList errors;

rule "Auftragsrabatt bei hohem Bestellwert"
when
    o: Order( value>5000 )
then
    o.setDiscount(0.05);
end

rule "Nachnahme nur bis 2500 € möglich"
when
    o: Order( value>2500, shippingType="COD" )
then
    errors.addError("Nachname nicht möglich bei Auftragswert " + o.getValue()
);
end
```


Drools im Einsatz

API



```
RuleBaseLoader loader = RuleBaseLoader.getInstance();
RuleBase ruleBase = loader.loadFromReader(
    new InputStreamReader(this.getClass().getResourceAsStream(
        "/demo.drl")));

WorkingMemory wm = ruleBase.newStatefulSession();

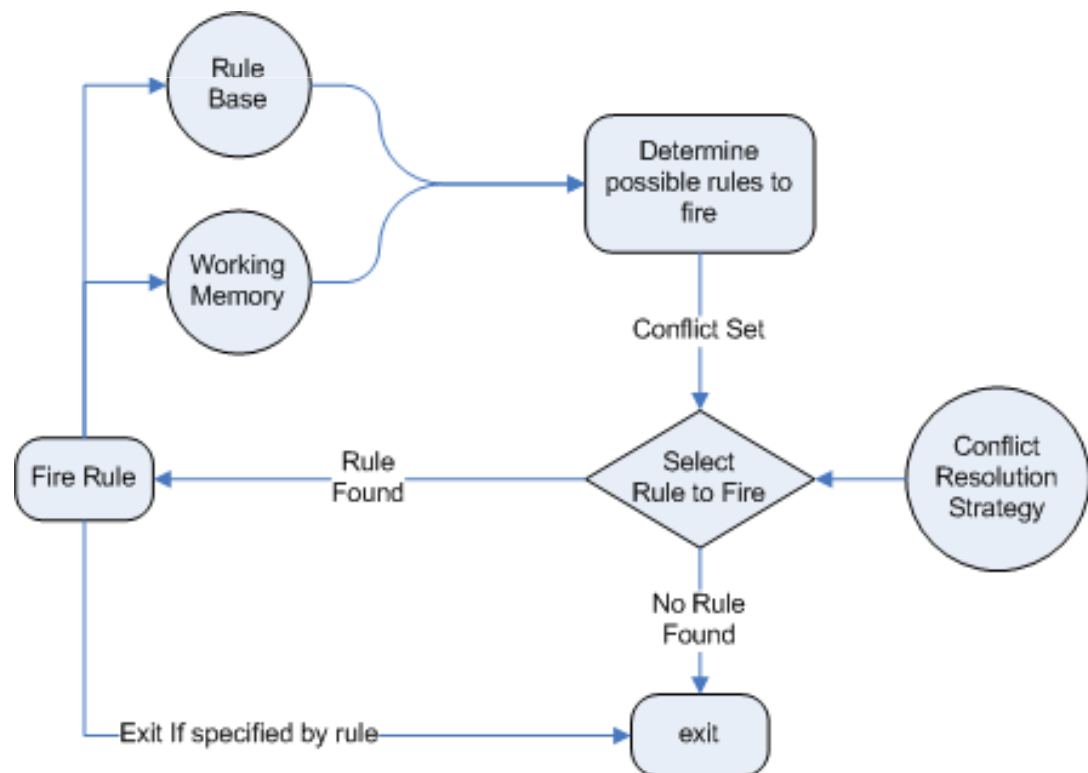
wm.insert(meldung);

wm.fireAllRules();
```

Drools im Einsatz

Wie funktioniert es intern?

- In-memory Knowledge-Repository
- Basiert auf Rete Algorithm
- Forward chaining
- Backward chaining ist geplant (Drools 5)



Für den Fachbereich: Decision Tables

The screenshot shows an OpenOffice.org Calc spreadsheet titled 'IntegrationExampleTest'. The spreadsheet contains a decision table with the following data:

	B	C	D	E
7				
8				
9		RuleSet Some business rules		
10		Import org.drools.decisiontable.Cheese, org.drools.dec		
11		Sequential true		
12				
13		RuleTable Cheese fans		
14		CONDITION	CONDITION	ACTION
15		Person	Cheese	list
16	(descriptions)	age	type	add(" \$param")
17	Case	Persons age	Cheese type	Log
18	Old guy	42	stilton	Old man stilton
19	Young guy	21	cheddar	Young man cheddar
20				
21		Variables java.util.List list		
22				

```
[when]Versicherter unter {MindestAlter} Jahre alt=  
    Person( age >= {MindestAlter} )  
[when]Mehr als {Anzahl} Unfälle gebaut=  
    Person( accidentCount > {Anzahl} )  
  
...  
  
[then]Police nicht möglich=  
    errors.addError("Police kann nicht ausgestellt werden");
```

```
rule "Abmeldung - Check mit DSL"  
when  
    Versicherter unter 25 Jahre alt  
    and  
    Mehr als 3 Unfälle gebaut  
then  
    Police nicht möglich  
end
```

DSL-Support

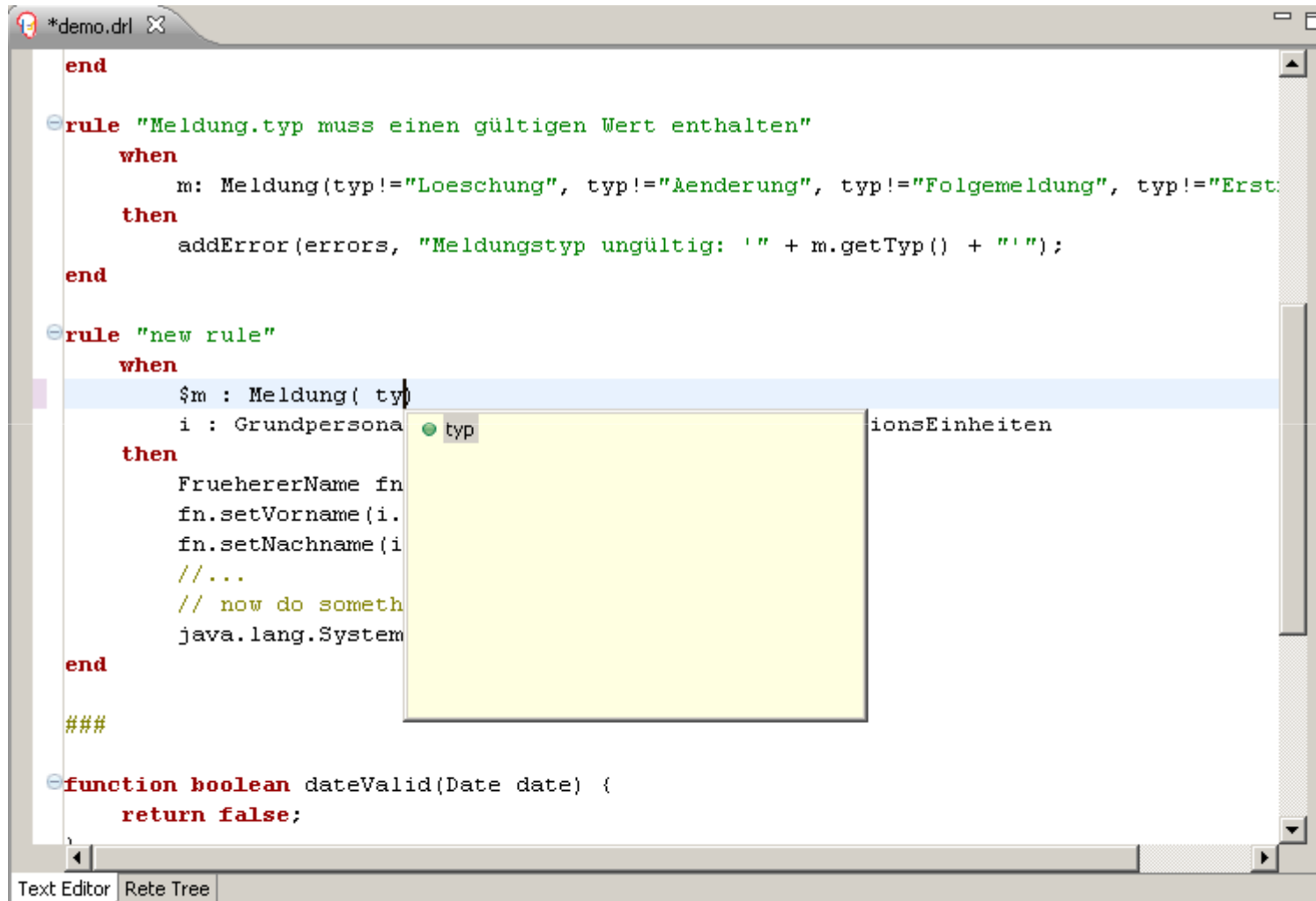
```
*Basic-rules.drl  hr-lang.dsl
#created on: 7/03/2006
package YourRulePackage

expander hr-lang.dsl

rule "Your First Rule"
  when
    #conditions
    There exists a Person with name of {name}
  then
    There exists a Person with name of {name}
    Person is at least {age} years old and lives in {locati
    then
      message {Message}
    end
  end
end

rule "Yo
  #inc
  when
  then
    #actions
  end
end
```

Regeleditor ohne DSL



```
end

rule "Meldung.typ muss einen gültigen Wert enthalten"
  when
    m: Meldung(typ!="Loeschung", typ!="Aenderung", typ!="Folgemeldung", typ!="Erst:
  then
    addError(errors, "Meldungstyp ungültig: '" + m.getTyp() + "'");
  end

rule "new rule"
  when
    $m : Meldung( typ
    i : Grundpersona
  then
    fruehererName fn
    fn.setVorname(i.
    fn.setNachname(i
    //...
    // now do someth
    java.lang.System
  end

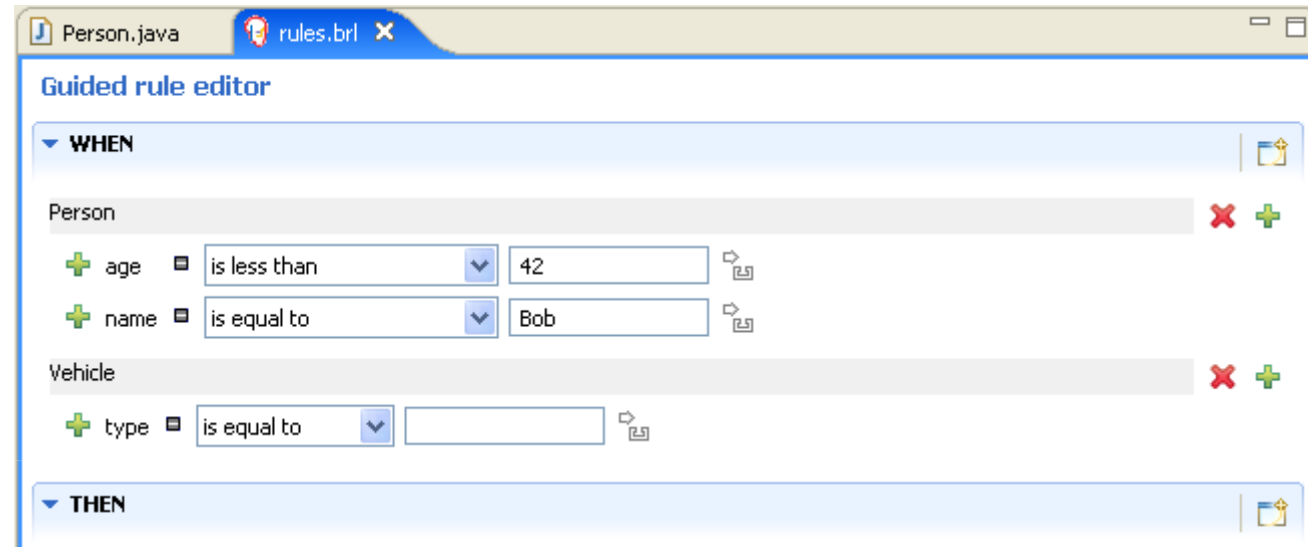
###

function boolean dateValid(Date date) {
  return false;
}
```

The screenshot shows a text editor window titled '*demo.drl'. The code is a Drools rule file. A code completion popup is visible over the line '\$m : Meldung(typ', showing a list of suggestions: 'typ' (selected), 'ionsEinheiten', and 'ionsEinheiten'. The editor has tabs for 'Text Editor' and 'Rete Tree' at the bottom.

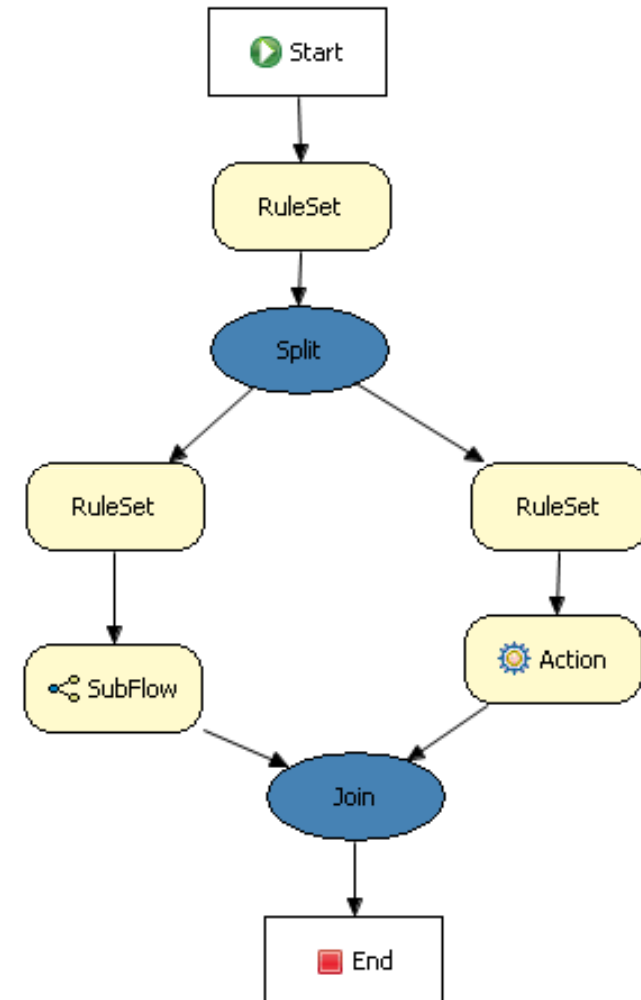
Guided Rule Editor

Verfügbar in
Eclipse &
BRMS



Rule Flow

- Regel-Reihenfolge
- Parallelität
- Bedingungen ob Regeln ausgeführt werden



Tooling: Eclipse

The screenshot displays the Eclipse IDE in a debug configuration for a Drools application. The main components are:

- Debug Console:** Shows the execution flow, including thread suspension at a breakpoint in `Rule_A_to_B_0`.
- Code Editor:** Contains the following rule definitions:

```
import org.drools.examples.State;

rule Bootstrap
  when
    a : State(name == "A", state == State.NOTRUN )
  then
    System.out.println(a.getName() + " finished");
    a.setState( State.FINISHED );
  end

rule "A to B"
  when
    State(name == "A", state == State.FINISHED )
    b : State(name == "B", state == State.NOTRUN )
  then
    b.setState( State.FINISHED );
    System.out.println(b.getName() + " finished");
  end

rule "B to C"
  salience 10
  when
    State(name == "B", state == State.FINISHED )
    c : State(name == "C", state == State.NOTRUN )
  then
    System.out.println(c.getName() + " finished");
  end
```
- State Transition Diagram:** A graph showing the progression of states from A to B to C, with nodes representing different states and edges representing transitions.
- Variables View:** Lists variables like `b`, `changes`, `name`, and `state` with their current values.
- Audit View:** Provides a detailed log of rule activations and object modifications, such as "Activation created: Rule Bootstrap a=A[NOTRUN](1)".
- Agenda View:** Shows the current state of the agenda, including active items and their salience.
- Working Memory View:** Displays the current state of the working memory, including objects like `State` and `PropertyChangeSupport`.

BPM vs. BRM | BPM + BRM

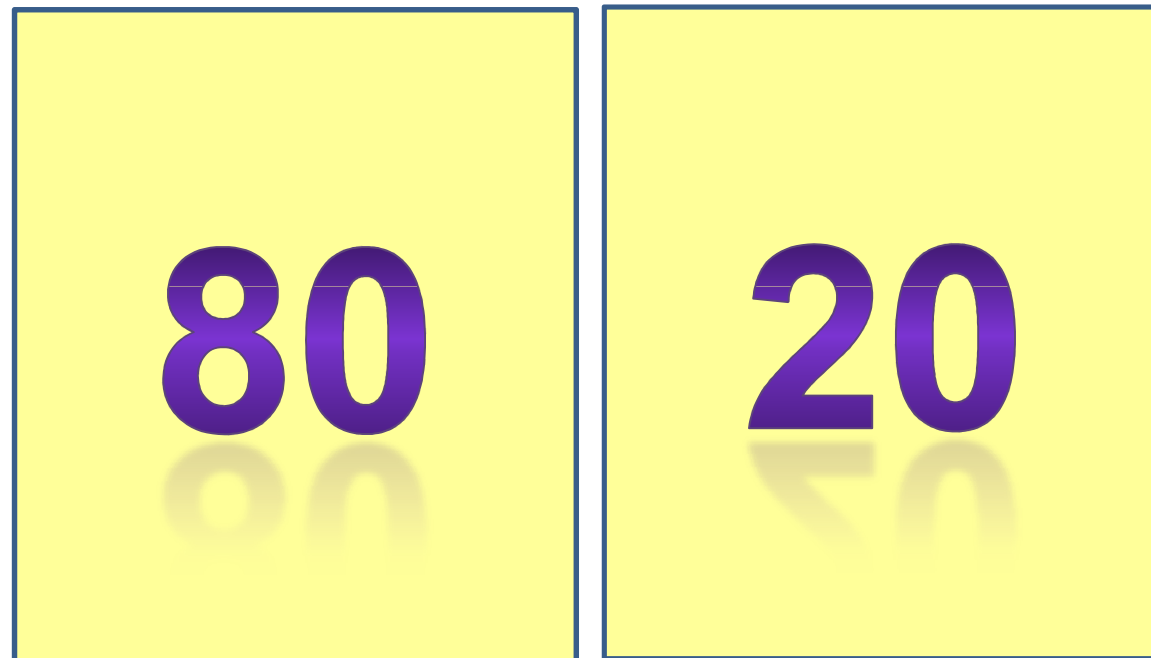


Geschäftsprozesse mit Regeln umsetzen?

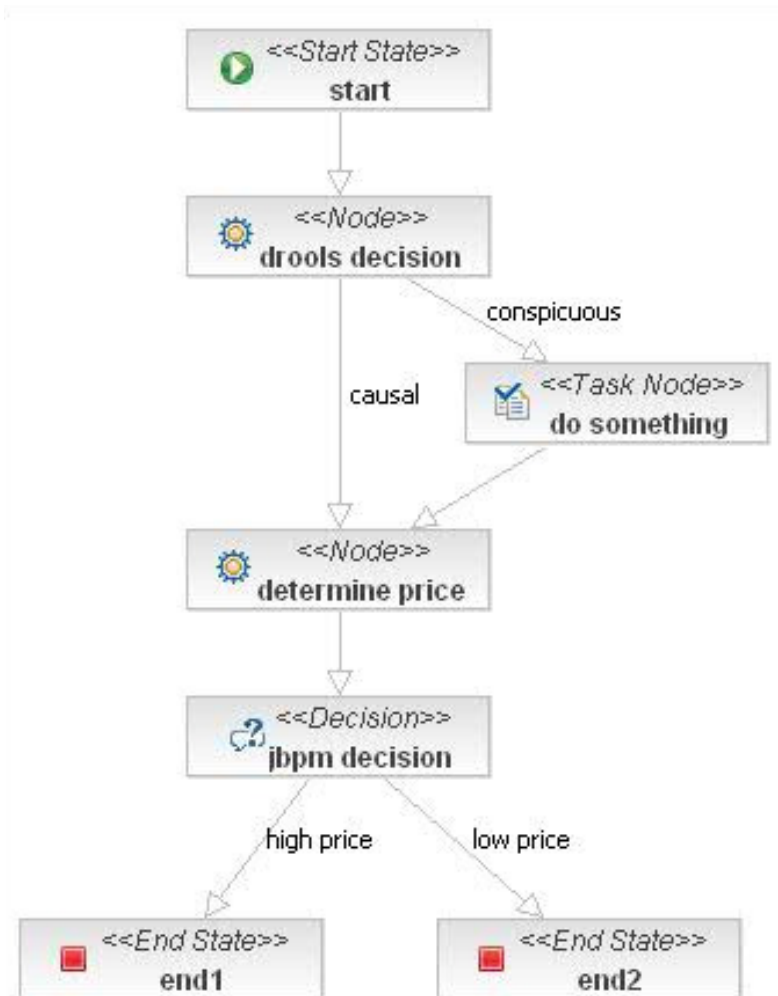
- Möglich!
- Vorteil: Maximale Flexibilität
- Nachteile
 - Keine Visualisierung / Modellierung des Prozessflusses
 - Nicht immer offensichtlich, warum was passiert
 - Keine „Grenzen“ durch Prozessstruktur
 - Evtl. Performance

- BPM (Prozesse)
 - Geschäftsprozesse / Workflows
 - Fachliche Modellierung
 - Hoher Standardisierungs-/Wiederholungsgrad
- BRM (Regeln)
 - Prozessunabhängige Regeln
 - Punktuelle Integration in Prozesse
 - Beeinflussung des Prozessablaufs (80/20 Regel)

Wann benutze ich was?



Kombinationsmöglichkeiten



Regeln treffen Entscheidungen

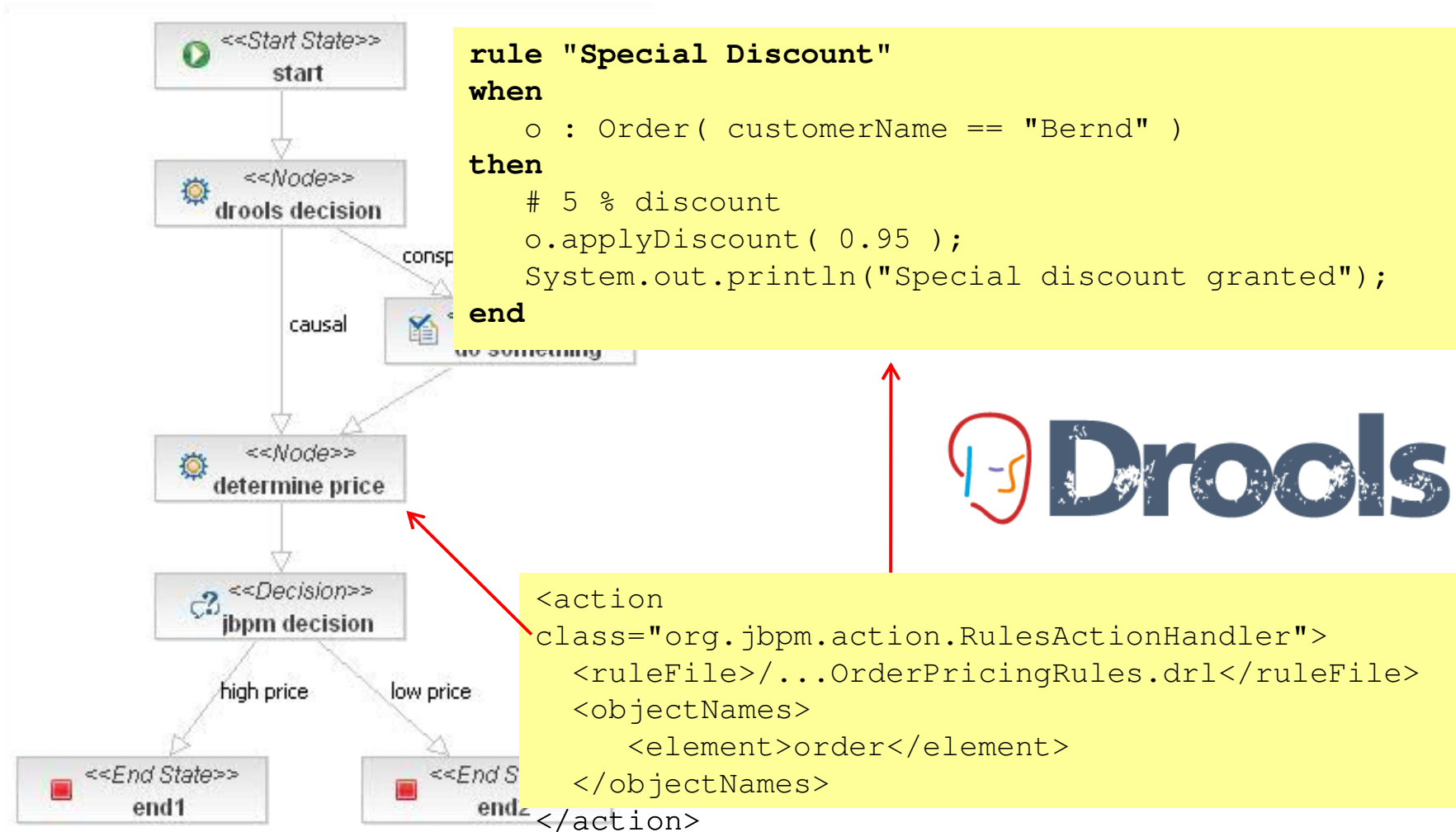
Regeln steuern Zuweisung von Aufgaben

Regeln schließen neues Wissen

Und: Beeinflussung des Prozessablaufs in Sonderfällen

Schließen von "neuem Wissen"

jBPM + Drools



Entscheidungen

jBPM + Drools

```
<node name="drools decision">
  <action class="org.jbpm.action.RulesActionHandler">
    <ruleFile>/com/.../OrderDecisionRules.drl</ruleFile>
    <objectNames>
      <element>order</element>
    </objectNames>
    <signalToken>>false</signalToken>
  </action>
  <transition name="conspicuous" to="do something" />
  <transition name="casual" to="determine discount" />
</node>
```

Besseres Design:
Regeln schreiben
Ergebnis in
Prozessvariable
und jBPM
„Decision“ wertet
diese aus

```
rule "Conspicuous Order"
when
  Order( price > 500)
then
  System.out.println("signal conspicuous order");
  executionContext.getNode().
    leave(executionContext, "conspicuous");
end
```


Beeinflussung des Prozessablaufs

Event getrieben reagieren / Ausnahmen



Actor-Assignment

Wer ist zuständig?

```
<assignment class="org.jbpm.assignment.RulesAssignmentHandler">
  <group>underwriting</group>
  <ruleFile>/Assignment.drl</ruleFile>
  <objectNames><element>policy</element></objectNames>
</assignment>
```



```
rule "Determine Junior Role"
when
  Policy( basePrice < 500)
then
  insert(new Role("junior"));
end
```

```
rule "Determine Actor"
salience -100
when
  Role($roleName : roleName)
  $a : Assignable()
  $group : Group()
  Membership( group == $group, role == $roleName,
              $user : user )
then
  $a.setActorId($user.getName());
end
```

Fazit

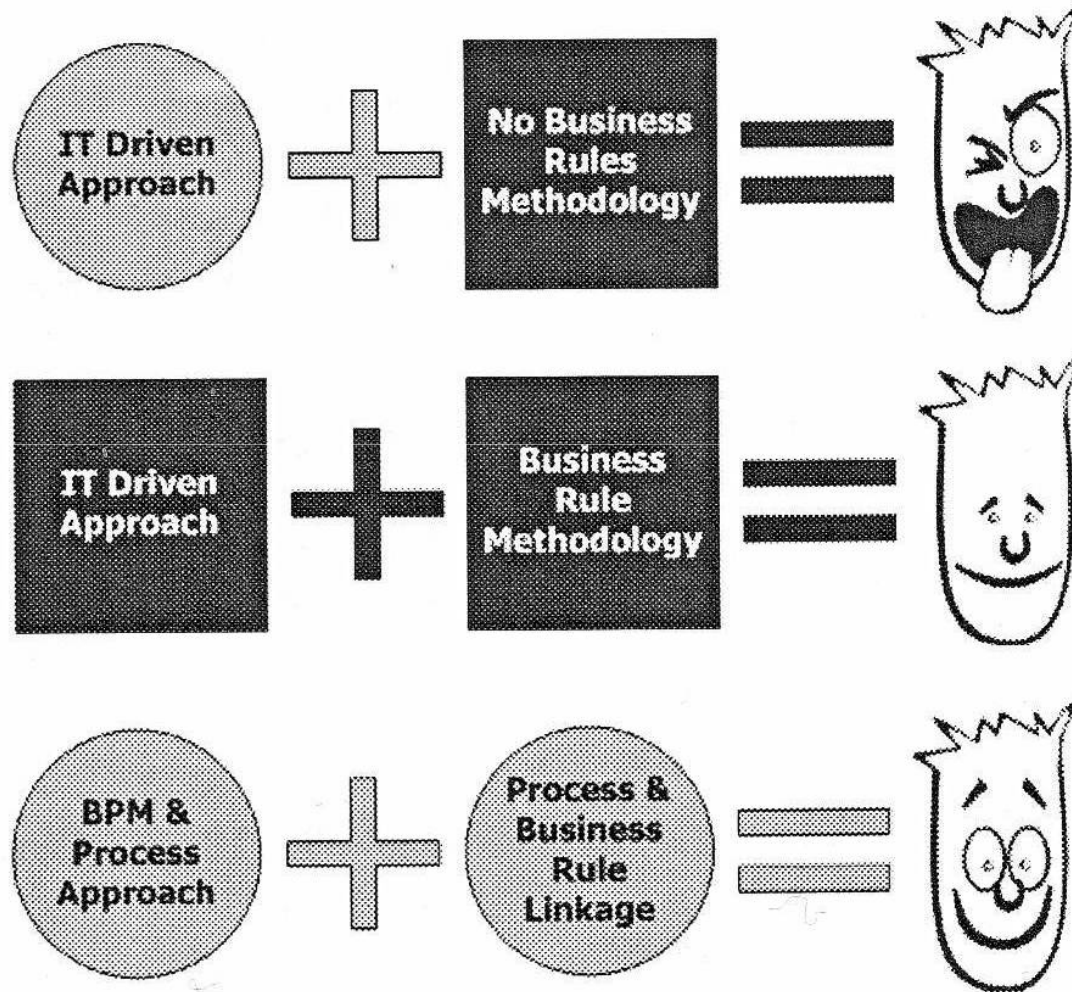
Geschäftsprozesse und Regeln mit jBPM und Drools

- jBPM & Drools integrieren sehr gut
- Benutzen Sie das richtige Tool für den Job
- Kombinieren Sie!
- jBPM ist eine kleine, flexible Process Engine. Bewährt auch in großen Projekten
- Drools ist cool 😊
- Drools steht teuren Rule Engines in nichts nach
- BPM & BRM sind Themen der Zukunft!

- Ausblick: Integration in den JBoss ESB

Fazit

Geschäftsprozesse und Regeln mit jBPM und Drools



Quelle:
Lary Ward &
Jordan Masanga

Fragen & Antworten



Bernd Rücker
Geschäftsführer
Berater, Trainer & Coach
bernd.ruecker@camunda.com
+49 711 3278645

Unsere Themen

- Ganzheitliches BPM
- Prozessautomatisierung
- SOA, BPEL, XPD, jBPM, Drools, ESB
- BPMN
- BPM-Toolauswahl

Unsere Leistungen

- Beratung
- Seminare
- Process as a Service
(Hosting)