The SkyNet funding bill is passed.
The system goes online on August 4th, 1997.
Human decisions are removed from strategic defense.
SkyNet begins to learn at a geometric rate.
It becomes self-aware at 2:14am Eastern time, August 29th.
In a panic, they try to pull the plug.
And, Skynet fights back.
Artificial Intelligence

Making computers think like people

Some Areas of Artificial Intelligence

- Vision
- Natural Language
- Understanding
- Expert Systems
- Genetic Algorithms
- Artificial Neural Systems
- Speech
- Robotics
Expert Systems - Knowledge Representation and Reasoning

- The study of Knowledge is Epistemology
- Nature, Structure and Origins of Knowledge

- Expert Systems use Knowledge representation to facilitate the codification of knowledge into a knowledge base which can be used for reasoning
  - we can process data with this knowledge base to infer conclusions
• Turing Complete
  – Propositional Logic
  – First Order Logic

• The Brain is the Inference Engine
  – scale to a large number of rules and facts
  – matches facts, the data, against Production Rules, also called Productions or just Rules, to infer conclusions which result in actions
  – A Production Rule is a two-part structure using First Order Logic for knowledge representation.

  \textit{when <conditions> then <actions>}
  – The process of matching the new or existing facts against Production Rules is called Pattern Matching
What is a Production Rule System

Production Memory (rules)

Inference Engine

Pattern Matcher

Agenda

Working Memory (facts)

Codification of the business knowledge

Repository of asserted Java instances

insert update retract
What is a Rule

Quotes on Rule names are optional if the rule name has no spaces.

- rule "<name>"
  <attribute> <value>
  when
  <LHS>
  then
  <RHS>
  end

Quotes on Rule names are optional if the rule name has no spaces.

- salience <int>
- agenda-group <string>
- no-loop <boolean>
- auto-focus <boolean>
- duration <long>

RHS can be any valid java. Future versions will support other languages, i.e Groovy
What is a Rule

- public void helloMark(Person person) {
  if (person.getName().equals("mark")) {
    System.out.println("Hello Mark");
  }
}

- Rule "Hello Mark"
  when
  Person( name == "mark" )
  then
    System.out.println("Hello Mark");
  end

Rules can never be called directly

Methods that must be called directly

Specific instances cannot be passed.

Specific passing of instances
package com.sample

import java.util.Map
import com.sample.Cheese

global Cheese cheese

function void exampleFunction(Cheese cheese) {
    System.out.println( cheese );
}

rule “A Cheesy Rule”
    when
        ...
    then
        ...
    end
Simple Pattern

Pattern

Object Type

Field Constraint

Field Name

Restriction

Evaluator

Value

Shower( temperature == “hot” )
Anatomy of a Pattern
Two Phase System

Working Memory Action
- insert
- modify
- retract

Determine possible rules to fire

Agenda Evaluation
- Select Rule to Fire
  - Rule Found
  - No Rule Found

Fire Rule

exit
Features

- **Engine**
  - Full Rete Implementation -- with high performance indexing
  - Dynamic RuleBases
  - Stateful and Stateless Execution Modes
  - Async operations
  - Rete and Sequential Rete
  - Rule Agent
  - Optional Data Shadowing
  - Pluggeable Dialects

- **Propositional Logic**
  - Literal Restriction
  - Variable Restriction
  - Return Value Restriction
  - Jointed and dis-jointed Connectives allowed - '&&' '||'
  - inline-Eval
Features

- First Order Logic (Quantifiers)
  - And
  - Or
  - Exists
  - Not
  - Accumulate
  - Collect
  - From
  - Forall
  - Nesting of any CE inside of 'and' and 'or'
  - Support for both infix and prefix 'and'/or' CEs
  - Nesting and Chaining of 'from', 'accumulate', 'collect'
Features

- Execution Control
  - Conflict Resolution (salience) Now pluggeable
  - Agenda Filters
  - Agenda Groups
  - Activation Groups
  - Rule Flow
  - Attributes (no-loop, lock-on-active)

- Temporal Rules
  - Scheduler for rule duration will fire when a rule is true for X duration

- Truth maintenance
  - Logical Insertions

- Event Model
  - Working Memory, Agenda, Rule Flow and Rule Base
Features

- **Configurable**
  - All Rete optimizations and execution behavior can be configured

- **Authoring**
  - Technical rules: DRL and XML formats supported
  - Template based DSLs
  - Guided Editor
  - BRMS and Eclipse
  - Decision Tables (Excel, Open Office)

- **BRMS (Knowledge Asset Management System)**
  - Authoring and Storage DRLs, RuleFlows, Decision Tables, Business Rules (Guided Editor)
  - Multi level Versioning, at both the rule and package level
  - Classification
  - Deployment
  - Built on Standards and Open APIs - JCR (JSR 170) Dublin Core, GWT
More Expression Flexibility

- 3.0.x only allows comma separated field constraints. 'or' could be used at the CE level, but resulted in subrule generation.
  - Can now use && and || inside the pattern for multiple values on the same field and across files - no subrule generation.
  - Can be as deeply nested as you like...

- Old
  - Person(age > 30, age < 40 ) OR Person (hair == ‘black’)

- New
  - Person(age > 30 && < 40 || hair == ‘black’
More Expression Flexibility

- Auto vivification of variables in dialect
- Old
  - Cheese(oldPrice : oldPrice, new Price == (oldPrice * 1.10))
- New
  - Cheese(newPrice == (oldPrice * 1.10))
- Drools 3.0 only supported infix 'and'/'or' constraints, which is idea for some situations but more complicated to read for others
- Old
  - Person(name == "matthew") or Person(name == "john")
- New
  - (or Person(name == "matthew") Person(name == "john"))
3.0.x had to always declare the variable, causing clutter, can now access direct properties of pattern variables.

Old

- p: Person (personId : id)
  i: Item (id == personId, value > 100)

New

- p: Person()
  i: Item(id == p.id, value > 100)

Eval rewrite for complex expressions, engine works out best way to do it - of course, flat models work best for performance.

Old

- Person($pets : pets ) eval
  ($pets.get(’rover’).getType().equals(“dog”) )

New

- Person(pets[’rover’].type == “dog”)

Eval rewrite for complex expressions, engine works out best way to do it - of course, flat models work best for performance.
More Powerful CEs

3.0:
- 'and'
- 'or'
- 'not' Could only nest a single Pattern
- 'exists' Could only nest a single Pattern

4.0:
- 'forall'
- 'from'
- 'collect'
- 'accumulate'
- 'not' - Now allows any nested CE
- 'exists' - Now allows any nested CE
More Powerful CEs

- 'forall', true when the pattern is true for all facts
  - `forall(Bus (color=="red"))`

- 'from', Pulls and unifies against non-working memory data (can call Hibernate queries)
  - `Restaurant ( rating == "fivestar")
    from hbSession
    .getNamedQuery ("restaurant query")
    .setProperties ( [key1 : value1, key2 : value2])
    .list()`
More Powerful CEs

- Collect - Allows you to use cardinality eg: when there are more than 6 red buses:
  - List(size > 6) from collect (Bus(color == “red”))

- 'from' can be chained and nested. Following is true if all items in a cart have a price greater than 10
  - List(size == ($list.size)) from collect(Item(price > 10)) from $cart.items Accumulate

- More powerful 'accumulate' allows you to execute actions on each matched fact in the set
  - $total : Integer()
    - from accumulate( $item : Item() )
      - init(count = 0; total=0)
      - action(count++;total += $item.price)
      - result( return total/count )

- Accumulate functions, built in and user defineable
  - $total : Integer()
    - from accumulate( Item($p : price) )
      - average($p)
Questions?

- **Dave Bowman**: All right, HAL; I'll go in through the emergency airlock.
- **HAL**: Without your space helmet, Dave, you're going to find that rather difficult.
- **Dave Bowman**: HAL, I won't argue with you anymore! Open the doors!
- **HAL**: Dave, this conversation can serve no purpose anymore. Goodbye.

Joshua: **Greetings, Professor Falken.**
Stephen Falken: **Hello, Joshua.**
Joshua: **A strange game. The only winning move is not to play. How about a nice game of chess?**
Overview

- Managing large rule sets

- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future

- An integrated approach to rules and processes
Example: Clinical DS

- Use rules to define clinical knowledge

  - Validation rules
  - Safety
  - Diagnosis assistance
  - Patient treatment
  - Authorization rules
  - ...

---

General Treatment Guidelines

Hospital Policies

Clinical Pathways

Different Stages

Law

Administrative Rules
Managing large rule sets

- Ideally, multiple rule sets can be combined into one knowledge base
  - Additive knowledge

- But rules are contextual
  - Explicitly specifying this context as part of the rule conditions makes the rules
    - More complex
    - Harder to reuse
Influencing execution order

Specifying the order in which rules should be executed:

- Purely declaratively
- Using salience (priorities)
- Using a control fact
- Using agenda-groups
- Dynamically adding/removing rules
- Manually loading rule sets and firing rules
- ...

Drools
Influencing execution order

- Taking into account:
  - Complexity
    - Keep rules simple
  - Understandability
    - Do not lose overview
  - Scalability
    - Lots of different rule sets
  - Performance
    - Without sacrificing performance
Overview

- Managing large rule sets

- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future

- An integrated approach to rules and processes
A graphical flow chart that defines the order in which rule sets should be evaluated

- Rules are grouped into rule sets
- Flow chart allows you to express
  - Sequence
  - Parallelism (split / join)
  - Choice
  - ...
Ruleflow Group

Grouping rules into ruleflow groups by using a special
ruleflow-group rule attribute

rule 'YourRule'
    ruleflow-group 'group1'
    when
        ...
    then
        ...
    end
Ruleflow

A domain-specific workflow language

- Workflow concepts in the context of rule evaluation
  - Supporting relevant workflow patterns

- Execution tightly integrated into rules engine
  - Avoid overhead by having to integrate with external engine
  - Allow usage of all rule features in combination with ruleflow

- Use the power of rules inside the ruleflow itself
  - Rules can used as a powerful condition and actions language
“When processing orders, make sure to first validate the order before processing it.”
Ruleflow features

- **Rule set nodes**

- **Control flow**
  - Sequence
  - Parallelism (split / join)
  - Choice

- **Nodes**
  - Actions
  - Milestone (= state)
  - Subflows
  - Looping
Using rules inside ruleflow

- as part of ruleflow groups

- as constraint expression language, e.g.
  - Split constraints
  - Milestone constraints

- as action language, e.g.
  - Action of an action node
Clinical pathways
- Describes the treatment of patients having a particular disease

Use rules to model the recommended treatment of patients
- Different rule sets for different stages of the treatment
- Use ruleflow to describe the overall flow

Multiple ruleflow instances can coexist and influence each other
- by inserting, updating or removing facts
Start

Examinations

Wait for Diagnose

Additional Examinations

Diagnose complete?

Notify GP

Select Treatment

Treatment X

Treatment Y

End

Schedule Follow-up

End

Start

examinations

Wait

treatment

End
Ruleflow designer (Eclipse)
Overview

- Managing large rule sets
- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future
- An integrated approach to rules and processes
Executing ruleflows

- Ruleflow instance
  - Whenever a ruleflow process is executed, a new ruleflow process instance is created that represents that specific execution of the ruleflow process
  - A ruleflow can be executed more than once
  - Multiple process instances (even of the same process) can coexist

- No limitations
  - Can be used in combination with normal rules
  - Can use all rule features like agenda groups, salience, etc.
Ruleflows are part of the rule base

```java
PackageBuilder builder =
    new PackageBuilder();
packageBuilder.addPackageFromDrl(...);
packageBuilder.addRuleFlow(...);
Package pkg = builder.getPackage();
RuleBase ruleBase =
RuleBaseFactory.newRuleBase();
ruleBase.addPackage( pkg );
```
Ruleflow processes should be started whenever necessary
  • Programmatically
    • `workingMemory.startProcess(id)`
  • From inside rules
    • `drools.getWorkingMemory().startProcess(id)`

Why manually?
  • Engine cannot (currently) automatically determining when a ruleflow should be started
Overview

- Managing large rule sets

- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future

- An integrated approach to rules and processes
How does it work?

- **Ruleflow-groups**
  - Activations for rules that are part of a ruleflow are not automatically put on the agenda
    - Ruleflow-group acts as a bucket
    - Once a ruleflow-group is activated, its activations are allowed to continue
    - Ruleflow-group deactivates if the bucket is empty

- **Constraints**
  - Constraints are translated into rules and become part of the Rete network
    - Automatic evaluation of constraints by the rules engine
    - Take advantage of optimizations in rules engine

- **Actions**
  - Interpreted at runtime
Overview

- Managing large rule sets

- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future

- An integrated approach to rules and processes
Future extensions to ruleflow

- Extend control flow capabilities
  - Workflow patterns

- Extend data capabilities
  - Variables (different scopes)

- Extend node types

- Extend execution engine
  - Transaction support
  - Persistence
  - Distributed execution
Overview

- Managing large rule sets
- Ruleflow
  - Language
  - Execution
  - How does it work?
  - Future
- An integrated approach to rules and processes
A knowledge base can contain knowledge in different formats
- Rules
- Decision tables / trees
- Domain-specific (rule) languages
- Ruleflow
- ...

Business expert can choose most appropriate “language” to model its business knowledge
Choose, integrate and unify

- Seamless *integration* between rules and processes
  - Processes can more easily embed rules for decision making
    - No complex integration
  - Rules can more easily interact with processes
    - Invoking processes from inside rules
    - Shared data
Choose, integrate and unify

- **Unified** approach to manage rules and different types of processes
  - Authoring environment
    - Plug in custom editors
  - Repository
  - Packaging + deployment
  - Execution (API)
  - Management, audit
  - Analysis
  - ...

Drools
### Unified Repository

#### Technical rule assets

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance extra liens percent</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Insurance Calcule</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver is underage</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>New licenced Driver</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver Single Young Male Driver factor</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver Mature Married With Young Child factor</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Priory Claimed Driver</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Day Vehicle Place</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Night Vehicle Place</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants an extra Car</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants glass coverage</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Driver wants non related expenses coverage</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>insuranceProcess</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>approve</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>rejection</td>
<td>Aug 28, 2007</td>
<td>Production</td>
</tr>
</tbody>
</table>
Unified auditing

- **Activation executed**: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown (2)
  - **Object removed (2)**: Diagnose: Diagnose disease X: Type unknown
    - Activation cancelled: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
    - Activation cancelled: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown (2)
  - **RuleFlowGroup activated**: Examinations [size=2]
  - **RuleFlow started**: ClinicalPathwayX[org.drools.examples.cdss.ClinicalPathwayX]
    - **Activation executed**: Rule Examination1
    - **Activation executed**: Rule Examination2
    - RuleFlowGroup deactivated: Examinations [size=0]
    - **RuleFlowGroup activated**: AdditionalExaminations [size=2]
  - **Object inserted (2)**: Diagnose: Diagnose disease X: Type unknown
    - Activation created: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown (2)
    - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
    - Activation created: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown (2)
    - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-12
  - **Activation executed**: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown (2)
  - **Object removed (2)**: Diagnose: Diagnose disease X: Type unknown
    - Activation cancelled: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown (2)
    - Activation cancelled: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
    - **Activation executed**: Rule Examination3
    - RuleFlowGroup deactivated: AdditionalExaminations [size=0]
  - **RuleFlow completed**: TreatmentY[org.drools.examples.cdss.TreatmentY]
  - **RuleFlow started**: TreatmentY[org.drools.examples.cdss.TreatmentY]
  - **RuleFlow completed**: ClinicalPathwayX[org.drools.examples.cdss.ClinicalPathwayX]
  - **Object inserted (2)**: Diagnose: Diagnose disease X: Type 2
Drools as a platform

Choose, integrate, unify!

- Supports different process models
- Integrated approach

Plugging in new
- Domain-specific extensions
  - Translation
- Extend existing models
  - E.g. new node type
- Plug in your own model
Joshua: Greetings, Professor Falken.
Stephen Falken: Hello, Joshua.
Joshua: A strange game. The only winning move is not to play. How about a nice game of chess?

• Dave Bowman: All right, HAL; I'll go in through the emergency airlock.
• HAL: Without your space helmet, Dave, you're going to find that rather difficult.
• Dave Bowman: HAL, I won't argue with you anymore! Open the doors!
• HAL: Dave, this conversation can serve no purpose anymore. Goodbye.