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

- Vision
- Natural Language
- Understanding
- Expert Systems
- Genetic Algorithms
- Artificial Neural Systems
- Speech
- Robotics

Some Areas of Artificial Intelligence
Branches of AI

**Biological**
- Artificial Neural Networks
- Evolutionary
  - Artificial Life
  - Genetic Algorithm
  - Wet DNA Computing

**Symbolic**
- Logic
- Frames & Scripts
- Rule Based Expert Systems
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
Production Rule System

• Turing Complete
  – Propositional Logic
  – First Order Logic
  – Declarative

• 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.

  when <conditions> then <actions>

  – The process of matching the new or existing facts against Production Rules is called Pattern Matching
Declarative Behavioural Modelling

Rule Engine
- Decision Trees
- Decision Tables
- Structured Natural Language
- Score Cards
- GUI Driven Rule Authoring
- Rule Flow

Technical Rule Language (DRL)
- Event Management/Stream Processing
- Forward Chaining
- Backward Chaining
- Constraint Programming
- Fuzzy Logic
- Uncertainty

Workflow/BPM
- Genetic Algorithms
- Neural Networks
- Geo-Spatial
- Semantic Web

Knowledge Asset Management and Configuration/Deployment System

Agent

p1 + p

p1 MultPoint

ESB

Agent

Agent
# The A-Team go Shopping

<table>
<thead>
<tr>
<th>Team</th>
<th>name</th>
<th>role</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Hannibal</td>
<td>Leader</td>
<td>Colonel</td>
</tr>
<tr>
<td></td>
<td>Faceman</td>
<td>Treasurer</td>
<td>Lieutenant</td>
</tr>
<tr>
<td></td>
<td>B.A.</td>
<td>Mechanic</td>
<td>Sergeant</td>
</tr>
<tr>
<td></td>
<td>Murdoch</td>
<td>Pilot</td>
<td>Captain</td>
</tr>
</tbody>
</table>
What do they Buy?
<table>
<thead>
<tr>
<th>Team</th>
<th>Item</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Barrack Babe</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Fur Love Cuffs</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Love Swing</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>Bondage Bear</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Bondage Starter Kit</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Nymphette basque</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Stress Balls</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Chocolate Body Paint</td>
<td>5</td>
</tr>
</tbody>
</table>
Relationships

<table>
<thead>
<tr>
<th>Team</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>role</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>price</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cart</th>
<th>0..n</th>
</tr>
</thead>
<tbody>
<tr>
<td>itemName</td>
<td></td>
</tr>
<tr>
<td>ownerId</td>
<td></td>
</tr>
</tbody>
</table>
What is a Production Rule System

Codification of the business knowledge

Production Memory

(rules)

Inference Engine

Pattern Matcher

Agenda

Working Memory

(facts)

Repository of asserted Java instances

insert update retract

Codification of the business knowledge
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

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

Methods that must be called directly

specific passing of instances

Rules can never be called directly

Specific instances cannot be passed.
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
Expressiveness

• Turing Complete
  Propositional Logic
  First Order Logic

• Propositional Logic
  – Cheese.name == “stilton”

• First Order Logic (Quantifiers)
  – Exists
  – Not
  – Accumulate
  – Collect
  – From
  – Forall

• Execution Control
  – Conflict Resolution (salience)
  – Agenda Groups
  – Activation Groups
  – Rule Flow

• Temporal Rules
  – Scheduler
Expressiveness

- **Truth Maintenance**
  - Logical Objects
  - Compensating Actions/Rollbacks (todo)
- **Nesting of conditional elements inside quantifiers**
- **Backward chaining**
- **Uncertainty**
  - Bayesian Logic
  - Fuzzy Logic
- **Event Stream/Management Processing**
- **Constraint Programming (solver)**
Authoring API

Parser

(xml, drl)

Interim AST

Rule Builder

Package Builder

Code Generator

Compiler

package
Runtime API

- Rule Base
- Working Memory
  - Working Memory Event Support
  - Truth Maintenance System
- Agenda
  - Agenda Event Support
Object Insertion and Pattern Matching

• LHS
  – One or more Patterns
  – Patterns are the conditions that must be satisfied for the rule to be legible for firing

• Object assertion
  – Patterns within the Rule Base are matched. Resulting in partial and full matches for Rules.
  – Fully matched Rules result in the creation of an Activation
  – No rules fire at this stage
Object Modification

• **How to modify a object in the Working Memory**
  – From Java Code
    ```java
    workingMemory.update( factHandle, modifiedFact )
    ```
  – From a Consequence
    ```java
    update( modifiedFact )
    ```

• **JavaBeans PropertyChangeListeners can provide automatic notification.**

• **Modifications result in**
  – Activation Cancellations
  – Activation Creations
  – Internally this is similar to a retract and assert
Two Phase System

- **Working Memory Actions**
  - Occurs in Java code and during the execution of a Consequence
  - Assertion
  - Deletion
  - Modification

- **Agenda Evaluation**
  - Triggered by Calling `workingMemory.fireAllRules()`
  - Executes the first Rule’s Consequence and enters Working Memory Action phase. At the end of the Consequence it returns to evaluating the Agenda.
  - When the Agenda is empty it returns back to the main Java code.
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
Customer customer = new Customer( "Fred Flinstone" );
customer.addItem( new Item( "brie" ) )
customer.addItem( new Item( "cheddar" ) )
customer.addItem( new Item( "feta" ) )
workingMemory.insert( customer )

rule "Message the customers who have not bought any brie"
   when
       $customer : Customer( $cart : cart -> ( ! $cart.includes( new Item( "brie" ) ) ) )
   then
       $customer.sendMessage( "Brie is your best friend" );
end
More Expression

• 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.
  – Person( age > 30 && < 40 || hair =="black" )

• 3.0.x auto-have autovivification of variables in dialect expressions
  – Before: Cheese( oldPrice : oldPrice, newPrice ==
                       ( oldPrice * 1.10 ) )
  – Now: Cheese( newPrice == ( oldPrice * 1.10 ) )
More Expression

• 3.0.x had to always declare the variable, causing clutter, can now access direct properties of pattern variables.
  
  – Before: p : Person(personId : id)
    i : Item(id == personId, value > 100 )
  
  – Now: p : Person()
    i : Item(id == p.id, value > 100 )

• Eval rewrite for complex expressions
  
  – Before: Person($pets:pets
                    eval($pets['rover'].type == "dog")
  
  – Now: Person( pets['rover'].type == "dog" )
Rule Engines are Relational

Customer
   int id

Item
   int customerId
   String name

Customer customer = new Customer( "Fred Flinstone" );
workingMemory.insert( customer );
workingMemory.insert( new Item( "brie", customer.getId() )
workingMemory.insert( new Item( "cheddar", customer.getId() )
workingMemory.insert( new Item( "feta", customer.getId() )

rule "Message the customers who have not bought any brie"
   when
       Customer( $customerId : id )
           not ( Item( customerId == $customerId, name == "brie" ) )
   then
       $customer.sendMessage( "Brie is your best
Exploiting Relational Data in 3.2

• 'forall'
• ‘from’
• ‘collect’
• ‘accumulate’
• **Forall**
  – True when the pattern is true for all facts
  – Forall( Bus(color == “red”) )

• **From**
  – Pulls and unifies against none working memory data

  Can call hibernate queries

  Sub fields

  Restaurant( rating == “five star” )
  from hbSession.getNamedQuery( “restaurant query” ).
  setProperties( key1 : value1, key2 : value2).list()
rule "Message the customers who have not bought any brie, and haven't bought brie in previous shopping trips"
when
    Customer( $customerId : id )
    not ( Item( customerId == $customerId, name == "brie" ) )
    not ( Item() from hibernateSession.getNamedQuery( "How much cheese?" ).setProperties( { customerId = $customerId, type => "brie" } ) )
then
    $customer.sendMessage( "You really haven't had enough Brie recently, remember Brie is your best friend" );
end
Collect

- Allows you to use cardinality
- When there are more than 6 red buses
- List(size > 6) from collect (Bus(color == “red”))
- 'from' can be chained. 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
rule "If we continuously have less than 10 brie items, then do a discount"
  duration 60000 //1 minute
  when
    $context : Context( count < 10 )
    cheeseList : ArrayList(size < 10)
     from collect Item( name == "brie" )
  then
    $context.setCount( $context.getCount() + 1 );
end

rule "If we continuously have less than 10 brie items, then do a discount"
  duration 60000 //1 minute
  when
    $context : Context( count < 10 )
    cheeseList : ArrayList(size > 10)
     from collect Item( name == "brie" )
  then
    $context.reset();
end
`collect`

rule "If we continuously have less than 10 brie items, then do a discount"
duration 60000 //1 minute
when
    $context : Conext( count >= 10 )
    cheeseList : ArrayList(size < 10) from collect Item( name == "brie" )
then
    // do discount
end
Accumulate

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

`accumulate`

duration 60000 // 1 minute

when

    Integer( intValue < 3 )

from accumulate( Item( name == "brie",
         $weight : weight ),
    init( int totalWeight = 0, count; ),
    action( count++;
               totalWeight += $weight; ),
    result( new Integer( x ) ) );

then

    // do discount

end
Line Debugger and new Rete Viewer
Rule Flow
Pluggeable Dialects

- Return-value, predicate, evals and consequences can now specify dialects, now supports Java and MVEL.
  - Cheese(type == "stilton",
    eval(price == (new Integer(5) + 5)),
    price == (new Integer(5) + 5))
  - Assert (new Person()) ( name = “mark”, age = 31 );
Why MVEL

• **Reflection/bytecode(JIT) compilation and execution modes.**
  – For huge systems we need to be able to avoid excessive bytecode generation, but still have the option for bytecode JIT for performance sensitive areas.

• **Fast reflection mode.**
  – We originally started with our own language JFDI, which was designed to be a simple and fast reflection based language, the idea is all work is done at compile time so runtime is just a series of reflection invokers. This design has been carried through to MVEL, so that it has good enough reflection performance. Where as other languages have to drop reflection mode and use bytecode to get any reasonable level of performance.

• **Pluggeable resolvers.**
  – Dictionary population is too slow, MVEL can resolve it's variable direct from the provided resolvers, which we make array based for performance.

• **Size.**
  – MVEL is currently <>
Why MVEL

• Custom language extensions.
  – MVEL is extending the language to support rule friendly constructs, in particular block setters. So I can do "modify (person) ( age += 1, location = "london" )" with the ability to treat that as a transaction block so I can run before and after interceptors on the entire block. This is made easier through the use of macros, so we can define our own keywords and have them expanded into mvel code.

• Static/Inferred typed or dynamic modes.
  – Variables can be untyped and totally dynamic.
  – Variables can be statically typed or type can be inferred, casting is supported.
  – Optional verifier for "typed mode", disallows dynamic variables and ensures all types and method calls are correct. Which helps with.
    Authoring time validation.
    Code completion.
    Refactoring.

• Configurable language feature support.
  – Language features can be turned off.
  – We don't want imperative flow structures in the "then" part, no 'if' 'switch' etc. Rules should be declarative, "when this do that" not "when this maybe do that".
BRMS?

- Business Rules Management System

- Why?
  - For managing a whole enterprises declarative rules
  - (and knowledge assets)
  - eg 5000 + rules for mortgage pricing
  - Business focussed view, not developer focussed
  - Versioning, editing, validating, FINDING (!), approving, searching, controlling, auditing, XXX-ing.

- Needs to complement developer tools, NOT REPLACE
Rule explorer with categorisation
Categorisation of assets is critical

- It's how you find stuff
- Categories are completely user/business driven
Dublin Core

- Encourage structured classification
- Future archeologists may be able to make sense of it ;)
- It's a prescriptive set of attributes to attach to an asset
Business friendly rules

- Controlled rule creation, authoring
Versioning

- We developers take it for granted
- Its good
- Business Analysts need it
  - But they have manually manage their requirements/rules documents
  - Have a manual workflow
  - Have manual versioning
  - No body knows the horrors I have seen
According to QLD MOTR Act 2003, we cannot insure this age range of drivers according to our current licence.

They will need to have a "guarantor" type relationship with a previously approved customer (we can give the customer a discount too, as a consolation prize!).
BRMS
BRMS
Create a new top level category.

Category name: Fibonacci

Description: fibonacci category

Categories aid in managing large numbers of rules/assets. A shallow hierarchy is recommended.

Current categories:
- HR
- Finance
- Draft

Refresh view:

Create a new category:
Delete the currently selected category:
BRMS
BRMS

```
rule "Rule_1"
when
  Person( age <= 42 , age > 21 )
  b : Board()
  not Board( cost > 1200 )
then
  b.setCost( 1200 );
end
```
Technical versus business rules

• A powerful inference engine allows you to solve hard problems
• Not ALL of the hard problem is technical
  – That's the “business” part of the rules
• For the parts that are technical, use the technical rule language
  – “Make the easy parts declarative, and the hard parts procedural”
Technology involved

- BRMS “client” is a web app
- Ajax via GWT
- JCR (Jackrabbit default implementation)
  - popular standard for content management
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?**