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
Drools Introduction

Processes
  - RuleFlow

Rules
  - Forward Chaining
  - Backwards Chaining

Temporal Reasoning
  - CEP/ESP

BAM

Agents

Uncertainty Systems to express truth degrees

Solver
Eclipse IDE

```java
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"
    when
        State; name == "B", state == State.FINISHED |
        c : State; name == "C", state == State.NOTRUN |
    then
        System.out.println(c.getName() + " finished");
    end
```

The selected working memory has no globals defined.
Eclipse IDE

The selected working memory has no globals defined.

- Activation created: Rule Bootstrap a=A[NOTRUN]
- Object asserted (1): A[NOTRUN]
- Object asserted (2): B[NOTRUN]
- Object asserted (3): C[NOTRUN]
- Object asserted (4): D[NOTRUN]

- Activation executed: Rule Bootstrap a=A[NOTRUN]
  - Object modified (1): A[FINISHED]
  - Activation created: Rule A to B b=0[NOTRUN]

- Activation executed: Rule A to B b=0[NOTRUN]
  - Object modified (2): B[FINISHED]
  - Activation created: Rule B to C c=C[NOTRUN]
  - Activation created: Rule B to D d=D[NOTRUN]

- Activation executed: Rule B to C c=C[NOTRUN]
  - Object modified (3): C[FINISHED]

- Activation executed: Rule B to D d=D[NOTRUN]
  - Object modified (4): D[FINISHED]

- Activation created: AgendaItem id=1262
  - RuleName = "B to C"
  - c=Static id=1263
  - d=Static id=1270

- Activation created: AgendaItem id=1263
  - RuleName = "B to D"
  - c=Static id=1269
  - d=Static id=1271

- MAIN[focus]= AgendaGroupImpl (id=1259)
  - [0]=State (id=1265)
    - NOTRUN= 0
    - changes=PropertyChangesSupport (id=1294)
    - name= "C"
    - state= 0
  - [1]=State (id=1269)
    - FINISHED= 1
  - [2]=State (id=1270)
  - [3]=State (id=1271)
Guided Editor (Eclipse)

Guided rule editor

WHEN

Person

- age is less than 42
- name is equal to Bob

Vehicle

THEN

(options)

Update constraints

Add a restriction on a field
Multiple field constraint
Add a new formula style expression
Variable name
```sql
// rule "Driver in unsafe area for marginal age"
when
Policy type is 'COMPREHENSIVE'
Driver is less than 25 years old
Driver has a location risk profile of 'HIGH'
then
<Driver has a location risk profile of '{risk}'>
<Driver has an age of at least {age}>
<Driver has had more than {prior} prior claims>
<Driver has had {number} prior claims>
<Driver is between {lower} and {upper} years old>
<Driver is greater than {age} years old>
<Driver is less than {age} years old>
<Policy has not been rejected>
<Policy type is '{type}'>

end

// rule "Driver unsafe for third party"
when
Policy type is 'THIRDParty'
Driver has had more than 2 prior claims

end
```
<table>
<thead>
<tr>
<th>Language Expression</th>
<th>Rule Language Mapping</th>
<th>Object</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>There is a customer ticket with status of &quot;{status}&quot;</td>
<td>customer : Customer() ticket : Ticket</td>
<td>[condition]</td>
<td></td>
</tr>
<tr>
<td>There is a &quot;{subscription}&quot; customer with a ticket status of &quot;{status}&quot;</td>
<td>customer : Customer(subscription =...)</td>
<td>[condition]</td>
<td></td>
</tr>
<tr>
<td>Log &quot;{message}&quot;</td>
<td>System.out.println(&quot;{message}&quot; )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Escalate the ticket</td>
<td>ticket.setStatus(&quot;Escalate&quot;); update</td>
<td>[consequent]</td>
<td></td>
</tr>
<tr>
<td>Send escalation email</td>
<td>sendEscalationEmail( customer, ticket</td>
<td>[consequent]</td>
<td></td>
</tr>
<tr>
<td>Base pricing rules</td>
<td>Age Bracket</td>
<td>Location risk profile</td>
<td>Number of prior claims</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>Young safe package</td>
<td>18, 24</td>
<td>LOW</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MED</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MED</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOW</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LOW</td>
<td>0</td>
</tr>
<tr>
<td>Young risk</td>
<td>18.24</td>
<td>MED</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>18.24</td>
<td>HIGH</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>19.24</td>
<td>HIGH</td>
<td>0</td>
</tr>
<tr>
<td>Mature drivers</td>
<td>25.30</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>25.30</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>25.30</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>25.35</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>
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:
The page at http://localhost:8080 says:
The snapshot called: NewSnapshot was successfully created.
rule "Rule_1"
when
   Person( age <= 42 , age > 21 )
   b : Board()
   not Board( cost > 1200 )
then
   b.setCost( 1200 );
Unifies Rules and Processes in a single engine

- Rules (LHS When) and expressions can be used in splits, milestones etc
  - creates a much richer model
- Rules and Processes see, reason and react and process the same data
  - Do not have send messages between two different engines
- Multiple instances, of different processes, can be executing at the same time in a single engine.
- Processes and Rules interactive with each other.
  - A Process or Rule can change data, which can impact how another rule or process is executing.
- Integrated Tooling and APIs
  - Single api for execution
  - Audit logging and visual Audit tools
  - Single server side tooling for storage, configuration and management and deployment
Ruleflow features

- **Rule set nodes**
- **Control flow**
  - Sequence
  - Parallelism (split / join)
  - Choice
- **Nodes**
  - Actions
  - Milestone (= state)
  - Subflows
  - Looping
Rule Flow - Rules and Processes
Rule Flow - Split Constraint Editor
Unified auditing

- Activation executed: Rule Start Clinical Pathway X if diagnosed d=Diagnose: Diagnose disease X: Type unknown
  - 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
  - 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
  - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-16-17
  - Activation created: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown
  - Activation created: Rule RuleFlow-org.drools.examples.cdss.ClinicalPathwayX-12
- Activation executed: Rule Remove old diagnose d=Diagnose: Diagnose disease X: Type unknown
  - 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
    - 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
### Technical rule assets

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance extra items percent</td>
<td>Sep 20, 2007</td>
<td>Production</td>
</tr>
<tr>
<td>Insurance Calculate</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>
Current only supports forward chaining

- Plans to support both backward/forward chaining

when

\[ \text{\$p : Person} ( \text{\$age : age == 17} ) \]  // from Working Memory
\[ \text{legalDriver} ( \text{\$age, $state: } ) \]  // executes corellated query
\[ // and unifies results with \]
\[ // current tuple, queries \]
\[ // can nested and recursive \]

then

// prints all valid states
\[ \text{System.out.println} ( \text{\$p.name + “: “ + $state} ) ; \]
Temporal Reasoning (CEP/ESP)

- Reasoning over absolute time windows
  - This is the common case of reasoning over a slide time-window and/or aggregation of values: "when the average transaction throughput calculated over 1 minute goes below the threshold, raise the alarm"

rule "absolute time window"
  time-window 60
  when
    $throughput : Number( doubleValue < threshold ) from
    accumulate( Throughput( $current : current ),
      average( $current ) )
  then
    // raise the alarm
  end
Support separate time-windows per CE:

rule "absolute time window per CE"
when
    Stock( $id : id )
    Number( $monthAvg : doubleValue ) from accumulate(
        DayCloseStockPrice( id == $id, $value : value ),
        average( $value ) ) over 30 days
    Number( intValue >= 3 ) from accumulate(
        $o : SellOrder( completed == true, id == $id, value < $monthAvg )
        count( $o ) ) over today
then
    // sell stocks
end
Temporal Reasoning (CEP/ESP)

- Reasoning over relative time windows
  - this is a powerful feature of reasoning over variable time-windows defined in relation to other patterns. Security example: "since a user logs in, until the user logs off, when there is any privileged action for this user, allow it and log it to the audit log"

```java
rule "relative time window"
  since
    LogInEvent( $user : user )
  until
    LogOffEvent( user == $user )
  when
    $evt : Event( user == $user, privileged == true )
  then
    // log event
    // allow action
end
```
Temporal Reasoning (CEP/ESP)

- Reasoning over relative time windows
  - "Each user has a limited amount he can buy/sell from the moment stock trading start to the moment stock trading stops, every day, without authorization. Every order that goes beyond that set amount (based on the user authorization level and trader policy) requires authorization to be completed."
rule "relative time window"
since
   StockTradeStartEvent( $date : date )
until
   StockTradeStopEvent( date == $date )
when
   User( $user : user, $level : level )
   TraderPolicy( level == $level, $max : maximumAllowance )
   Number( $total : doubleValue) from accumulate(
      CompletedOrder( $value : value ),
      sum( $value )
   )
   $o : OrderRequest( value > ( $max - $total ) )
then
   // request autorization to complete order $o
end
Events have implicit time attributes and it must be possible to constraint events on its time attributes, using operators "after", "before", "between":

rule "time constraints between events"
when
   $order : $StockBuyOrder( $id : id )
   $StockBuyConfirmation( relatedEvent == id, this after [0,10] )
then
   // buy order confirmed
end
Temporal Reasoning (CEP/ESP)

- Support to reason over the absence of events:

  rule "absence of events"

  when

    $temp : TemperatureReading( celciusGrade > $threshold )
    not SplinkerActivation( this after 10 )

  then

    // raise the alarm

  end
Temporal Reasoning (CEP/ESP)

Future

- More Complex forms of temporal reasoning
- More expressive forms of even object correlation
- Hybrid support for bayesian networks
- Scalability, High Availability
- Ideas from the community? :)

Drools
- Engines and their applications emit events
  - Current node in process, time between nodes in process, number of executions of a rule, total rule executions in an engine, total number of cheeses bought.

- Rules, Processes and CEP
  Are perfect for BAM

- Multi tier-ed (near/far) BAM
  - Internal self monitor (very near)
  - Local external monitoring (near)
  - Remote external Monitoring (far)
Agents

- OSGi for the container
  - r-osgi (or similar)
    - SLP for discover and directory
    - Distributed OSGi registries
    - p2p, p2m
  - FIPA messaging, instead of/as well as RMI?
    - High performance requires binary
      - XML adapters can be made for 3rd party support

- Leverage possible new CE
  - “from message”
    - Tuples enter the “message from” CE.
      - May optional send one or more messages
    - Incoming messages can join with one, a group or all Tuples
Uncertainty Systems

- Uncertainty Systems to express truth degrees and reason over partial data
  - Davide Sottara (dsottara@deis.unibo.it)

![Diagram of Uncertainty Systems]

- Forms of Uncertainty (measures)
  - Probability
  - Belief / Plausibility
  - Possibility / Necessity
  - Fuzzy Sets

- Approaches
  - Frequentist
  - Subjective (Bayesian)

- Given for Single Event
- Given for Set of Events

- Basic Mass Assignment Functions
- Aggregator (sum)

- Plausibility Functions
- Belief Functions
Uncertainty Systems

- Traditional Pattern
  - `Shower(temperature == "hot")`

- Pattern with uncertainty evaluator
  - `Shower(temperature == ~"hot")`

- Pattern with uncertainty evaluator and parameters
  - `Shower(temperature == ~(10, $x, 15, $y) "hot")`
Drools-solver solves planning problems

Geoffrey De Smeet

Drools-solver combines a search algorithm with the power of the drools rule engine to solve those planning problems
Planning problems

Solves planning problems, such as:

- Employee shift rostering
- Freight routing
- Supply sorting
- Lesson scheduling
- Exam scheduling
- The traveling salesmen problem
- The traveling tournament problem
The n queens example

- Place n queens that cannot attach each other on a n*n chessboard
- One of the examples of drools-solver
- Implementation explained in detail in the reference manual
Different search algorithms

Supports several search algorithms:
- Simple local search
- Tabu search
- Simulated annealing

You can easily switch the search algorithm, by simply changing the configuration.
Example configuration

<localSearchSolver>
  <scoreDrl> .../smartTravelingTournamentScoreRules.drl</scoreDrl>
  <scoreCalculator>
    <scoreCalculatorType> HARD_AND_SOFT_CONSTRAINTS </scoreCalculatorType>
  </scoreCalculator>
  <finish>
    <maximumMinutesSpend> 2 </maximumMinutesSpend>
  </finish>
  <selector> ... </selector>
  <accepter>
    <completeSolutionTabuSize> 1500 </completeSolutionTabuSize>
  </accepter>
  <forager>
    <foragerType> MAX_SCORE_OF_ALL </foragerType>
  </forager>
</localSearchSolver>
Score calculation with drools

- Uses the drools rule engine to calculate the score
- Adding extra hard and soft constraints to the score function is easy and scalable
Example score rule

rule "fourConsecutiveHomeMatches"
    when
        $m : Match($homeTeam : homeTeam, $day1 : day)
            Match(homeTeam == $homeTeam,
                eval(day.getIndex() == $day1.getIndex() + 1))
            Match(homeTeam == $homeTeam,
                eval(day.getIndex() == $day1.getIndex() + 2))
            Match(homeTeam == $homeTeam,
                eval(day.getIndex() == $day1.getIndex() + 3))
    then
        // ...
    end
Future plans

- More examples
- Refactor of the move generation (selector), to allow more efficient search algorithms
- Make moves declarative
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?