RULES-BASED ANALYSIS WITH JBOSS DROOLS : ADDING INTELLIGENCE TO AUTOMATION

E. De Ley, D. Jacobs, iSencia Belgium, Ghent, Belgium

Abstract

Rules engines are less-known as software technology than the traditional procedural, object-oriented, scripting or dynamic development languages. This is a pity, as their usage may offer an important enrichment to a development toolbox.

JBoss Drools is an open-source rules engine that can easily be embedded in any Java application. Through an integration in our Passerelle process automation suite, we have been able to provide advanced solutions for intelligent process automation, complex event processing, system monitoring and alarming, automated repair etc.

This platform has been proven for many years as an automated diagnosis and repair engine for Belgium's largest telecom provider, and it is being piloted at Synchrotron Soleil for device monitoring and alarming. After an introduction to rules engines in general and JBoss Drools in particular, we will present its integration in a solution platform, some important principles and a practical use case..

PHYSICAL SCIENCES AND INTELLIGENT PROCESS AUTOMATION

General Vision

In today's world of physical scientific research, the usage of and the dependency on advanced technology has become crucial. As the boundaries of research shift, the requirements on tools for exploration, experimentation and verification become ever more complex and extreme.

The required investments, both in costs and in time, are typically huge and can no longer be duplicated by all parties involved in related research domains. Collaboration in the design and usage of advanced "centralized" or "shared" research infrastructure, like synchrotrons, becomes mandatory.

The institutes offering such infrastructure then effectively become service providers for all the parties involved in related research disciplines, both for internal scientists and for visiting researchers (or groups).

Offering complex infrastructure as a service also implies extensive support organizations and processes, like project approval procedures, safety and security regulations, HR, IT support, resource planning, repair workshops, vendor management etc.

Which in turn implies, maybe a bit paradoxically, that the more advanced and exotic the technological requirements for executing scientific research, the more the organizational aspects get closer to traditional requirements for professional service providers in non-scientific domains. But with a clear long-term advantage, compared to many professional organisations : scientific institutes have a single long-term goal - *offer the best-possible infrastructure and support for advancing science*. This clear goal can drive the complete organisation to operate in a transparent and collaborative way.

It is our vision that a *common open software platform for intelligent process automation* can be of tremendous value to assist an organisation in achieving this goal, across the majority of its activities by removing administrative duplication, reducing risk for errors in each step, removing the need for continuous on-site presence and combining advanced security with consistency and transparency where needed.

Such a platform can be used for supporting, automating and orchestrating a wide range of processes, from traditional workflows (such as project approval) to advanced algorithms for resource allocation (e.g. beamlines, guest rooms, login accounts etc.), from automated machine monitoring of the accelerators and control sequences for beamlines, to offering secured access to experiments' results.

Concretely for Synchrotrons

Automation in synchrotrons is typically associated with device drivers, sensors, control buses and other technical components, driven by scripts and hard-coded programs. But the scope for automation can become much larger by increasing the level of abstraction. This in turn allows adding advanced features like automated analysis, diagnosis and decision management through the integration of e.g. a rules engine, which is the scope of this paper.

AUTOMATED ANALYSIS AND DECISIONS

Some example uses of enriching automated processes with analysis and decision logic are:

- *What to do next?* Complex and/or dynamic routing or filtering logic based on previously obtained information
- *Is what we did OK?* automated analysis or validation of results obtained in previous steps
- *Just wake me up when you need me.* Intelligent monitoring systems, integrated in the same platform to automate basic control processes and business processes
- *Wait a moment, how did we get in this mess?* Correlating acquired data and elementary

analysis results to obtain a final diagnosis on a problem situation

- *Oops, how can we get this resolved?* Automated advises for corrective actions in case of problems
- *Don't bother me any more for this.* Automating simple recurring decision processes

Automated diagnosis or decision management can be decomposed in the following stages :

- Autonomous monitoring or discovery
  Continuously track some important performance indicators and generate an alarm when a value becomes suspect.
- Context-sensitive analysis of raised alarms
  Typically involves looking at many different datasets and measurements.
  Uses thresholding, trending etc. to identify exceptions (e.g. taking your fever)
- Consolidation of obtained information and analysis results towards a diagnosis or decision (e.g. "you've got a simple flue"). This may optionally include advises for problem resolution or repair.

RULE ENGINES

To cater for the above needs, the software system must be able to handle large volumes of data and to apply complex reasoning logic on it. An easy-to-understand reasoning approach uses conditional actions, also known as "if-then" logic. Such constructs are frequently used in traditional programming, but for representing complex context-dependent logic they quickly become a nightmare to understand or maintain since :

- it quickly becomes tricky to correctly represent all combinatorial possibilities of the conditions with nested if-then-else statements
- the logic is spread out across different parts of the code base
- the result becomes more and more fragile with each incremental adaptation

Rule engines are specialized software systems for applying conditional actions (if/then rules) on data. The use of a rule engine brings following benefits :

- business logic, expressed as rules, can be externalized from the application code. The business logic can then be maintained in a centralized way with dedicated tools.
- rules can be defined in human-readable form in text files, spreadsheets etc.
- rules are often defined in a declarative way. Starting from facts that you know are true, the desired outcome or action is defined.
- rule engines are optimized to evaluate large rule bases, find matching conditions and trigger the corresponding rules.

The term "rule engine" is used in different ways. The ones which interest us are also known as "production rule systems". The central part of such a system is an inference engine that is able to match rules against facts or data to infer conclusions which result in new facts or in actions. Through optimized matching algorithms such engines are able to scale to high volumes of rules and facts.

INTRODUCING JBOSS DROOLS

Drools[1] is the leading open-source rules engine written in Java. It was started in 2001 and became a prominent Java rules engine with its 2.0 release. In 2005 the Drools project became part of the JBoss group and since then Drools also became known as JBoss Rules. In 2006 JBoss was acquired by Red Hat. Besides funding the core development team, they provide professional services like support and training. The current stable version is Drools v5.2.

The Drools suite contains several modules that together form their *Business Logic Integration Platform,* of which the following ones are of interest for our purposes :

- Drools Expert : the actual rule engine
- Drools Fusion : an integrated engine extension to support event processing and temporal reasoning
- Drools Planning : automated resource planning

Drools Engine

The core of the Drools suite is an advanced Inference Engine using an improved Rete algorithm[2] for pattern matching, adapted for object oriented systems, and for Java in particular. Rules are stored in the production memory, while facts are maintained in the working memory.

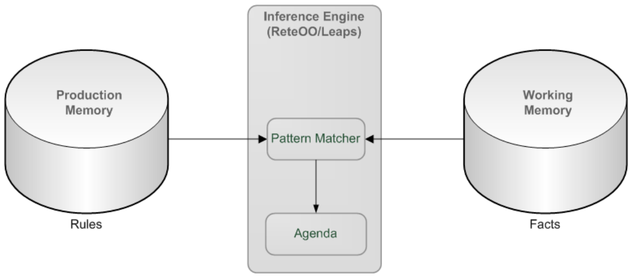

Figure 1 : The main parts of a rule engine.

The production memory remains unchanged during an analysis session, i.e. no rules are added or removed or changed. The contents of the working memory on the other hand can change. Facts may be modified, removed or added, by executing rules or from external sources. After a change in the working memory, the inference engine is triggered and it determines which rules become "true" for the given facts. If there are multiple selected rules, their execution order will be managed via the Agenda, using a conflict resolution strategy.

When a selected rule has been executed, and one or more changes were done in the working memory, the inference engine goes to work again, adapting the agenda and executing the rule that has now reached the highest priority. This iterative process continues until the agenda is empty. Then the analysis session terminates and results can be queried from the working memory or through the usage of global variables.

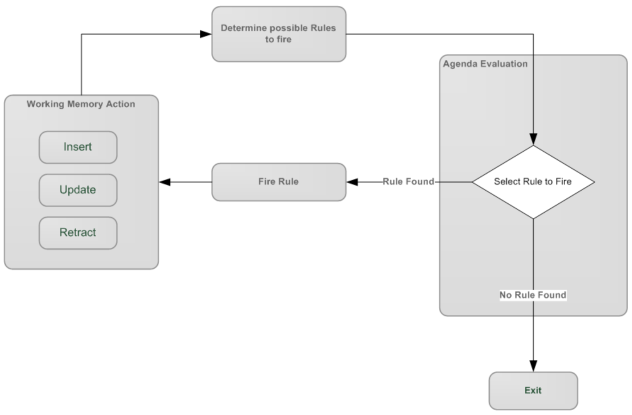

Figure 2 : Iterative inferencing.

Options for Rules Definitions

Drools offers different ways to define rules. The native rules language will feel familiar for Java developers, with the addition of advanced expressions for specifying conditions. For example :

```
rule "Raise prio if 3 pending alarms"
when
    $system : System()
    $alarms : ArrayList( size >= 3 )
      from collect( Alarm( system == $system,
       status == 'pending' ) )
then
    # Raise priority, because $system has
    # 3 or more alarms pending.
   # The pending alarms are in $alarms.
end
```

When repetitive rule patterns occur, one can set-up more readable approaches to define the rules. One way is to create a rule language dedicated to the problem domain, so-called DSL rules.

Rules can then be expressed like :

```
when
  There is a Cheese with
  - age is less than 42
  - type is 'stilton
then
  Taste It
```

which gets translated at runtime in the technical rule language, based on the defined DSL mapping.

A third interesting approach to define rules is to use spreadsheets. This offers a familiar entry tool for non-IT persons to easily define large rules sets as long as they fit in a-priori defined rule templates.

INTRODUCING PASSERELLE

Within Passerelle [3], all processes are defined in graphical models that are executable by one of the available executors. In "real" production environments, the Passerelle Manager, our web-based process automation server platform[4], is the preferred execution engine. It offers a complete solution for process definition, maintenance, execution and follow-up.

In the Passerelle process engine, a process (a.k.a. sequence) is defined by a graphical assembly of actors that each perform a step of the complete process. Assemblies are stored in XML files. Actors get single well-defined responsibilities and are unaware of their surrounding "colleagues". They are only able to communicate via messages sent across channels between the sender's output port and the receivers' input port(s).This strong decoupling improves reusability and maintainability.

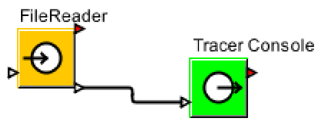

Figure 3 : Reading a file in Passerelle.

In the figure above, the FileReader actor will read a text file from a configurable location, and will send a message for each line to the next actor. In this simple model, this one just dumps the text content of each received message to a Console view.

There are actors available to control Tango devices, to make routing decisions in the process flow, to query databases, send e-mail or SMS etc.

INTEGRATING DROOLS

Drools is easy to integrate in any Java application via :

- Standard JSR-94 Java Rule Engine API
- Drools' own "Knowledge API" with more advanced capabilities, for example to be able to use the Fusion and Planning modules. Passerelle uses the Drools Knowledge API.

One can distinguish the following major API features, required for integration in a production-ready solution:

1. Build a binary version of a knowledgebase packages, from the different types of source
2. Select a knowledgebase package for execution
3. Start an analysis session
4. Feed facts into the working memory
5. Start the inference engine
6. Obtain the results
7. Release session resources when appropriate

Rules and Processes as Versioned Assets

For integration features (1) and (2), Passerelle includes a specialized repository service to maintain *process models*, and *rules assets*, grouped in *project* assets. The service offers a.o. version management, knowledgebase building and packaging, a simple API to retrieve pre-built binary knowledgebase packages and import/export to migrate project assets between test and production environments.

Actors with Intelligence?

Passerelle has specialized actor libraries to integrate the usage of the Drools rule engine in an automated process. These actors utilize the Drools API features (3)-(7) as described above. By configuring the actors with a reference to the desired rules package, a limited number of actor implementations can be reused in many different processes, each time with the matching rules-based logic.

In this way it is possible to apply rules to analyse the results of the work done in the previous steps of the process, and act on the obtained analysis results, all within a common paradigm of sequences and actors.

Support Tools

Passerelle Manager stores all analysis results, including timed task execution traces, in a relational database. The web UI includes rich views on this data, to aid with application support and with control and improvement of the analysis quality.

EXAMPLE : DIAGNOSIS OF TELECOM LINE - AND SERVICE QUALITY

For the largest Belgian telecommunications provider, a solution has been implemented and is operational since approximately 5 years. To reduce the waiting times for the customers that are contacting the support help-desk, there was a need to be able to automatically trigger a number of tests and measurements on the impacted telephone line and associated services. This should happen even before the customer gets in contact with a help-desk operator, i.e. while still in the waiting queue. Then the operator already has all detailed test results available about the condition of the telephone line, ADSL services etc, when the customer is passed through. Also field technicians, working on-site at their customers can use this automated Diagnosis And Repair Engine.

Each day, approximately 35000 diagnoses are performed for about 700 help-desk operators and 2500 technicians, with dedicated rules for different customer segments, line technologies etc. The result is a dramatic reduction in error rates and a significant increase of "first-time-right" indicators.

EXAMPLE: MONITORING BEAMLINE SOURCE POSITIONS

In this scenario, the goal is to continuously compare beamline source positions to reference values, and to check for slow drift. When the measured position starts drifting, or deviates from the reference position by more than a given threshold, a number of possible root causes must be evaluated (like temperature circuits, insertion device changes etc), and a diagnosis on the plausible cause(s) must be produced.

This requires analysing a stream of position measurement events to :

- compare them against reference values obtained from a snapshot
- check measurement values for drift within a certain time window

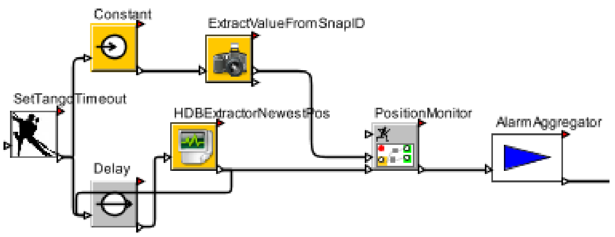

Figure 4 : A fragment of the monitoring sequence.

The monitoring sequence includes an actor to read the reference values from a snapshot and to feed them via an event stream input to the PositionMonitor actor. The HDBExtractorNewestPos actor periodically reads the latest position measurements from the archive and sends these as events to the PositionMonitor. This one can then compare the measurements with the snapshot and also check for drift in measured positions in a time window of e.g. 15 seconds. The below presents an example of using a Java domain model in rules :

```
rule " BLSrcPosition drift > 0.01 within 15 s"
  when
    bl1 : BLSrcPosition($blName1 : beamLine,
        $blPos1 : position )
        from entry-point "BL SrcPos Stream"

    bl2 : BLSrcPosition($blName2 : beamLine,
        beamLine == $blName1,
        this after [0s,15s] bl1,
        eval(position.distanceTo($blPos1)>0.01))
        from entry-point "BL SrcPos Stream"
  then
    // raise an alarm...
end
```

CONCLUSIONS

We have presented the concepts and the added value of integrating a rule engine in a software platform for intelligent process automation. Drools offers many advanced features that are of interest. Besides using standard production rules, it is also possible to perform temporal event-based reasoning and resource planning.

The resulting solution platform can be applied in many different domains, ranging from automated diagnosis and repair in telecommunications customer support to monitoring synchrotron infrastructures to automating beamline experimental sequences.

REFERENCES

[1] JBoss Drools : http://www.jboss.org/drools/
[2] Charles Forgy, "On the efficient implementation of production systems." Ph.D. Thesis, Carnegie-Mellon University, 1979.
[3] Passerelle project information : http://www.isencia.be/services/passerelle and http://code.google.com/a/eclipselabs.org/p/passerelle/
[4] E. De Ley et al. : "Web-based Execution of Graphical Workflows : a Modular Platform for Multifunctional Scientific Process Automation", ICALEPCS 2011.