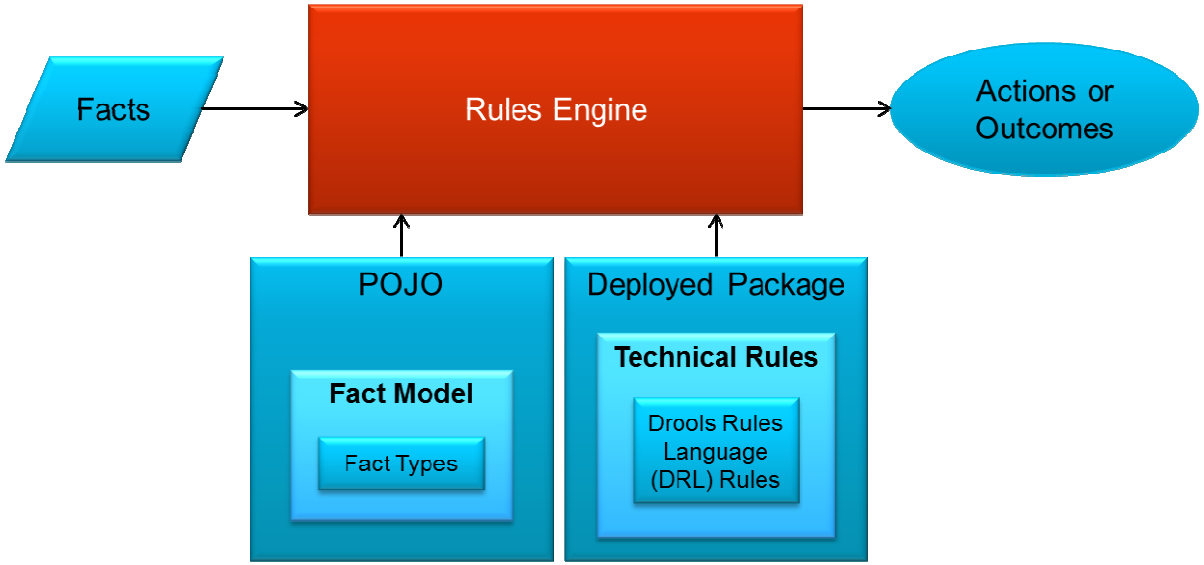
A Rules Engine Experiment:
Lessons Learned on When and How to use a Rules-Based Solution

June 24, 2014



MITRE

Agenda

- **Discuss BRMS Rules Experiment conducted for a MITRE sponsor**

  The Hypothesis to be Tested

  The BRMS Rules Engine can:
  - Support the externalization of business rules from the body of the code, enabling rules to be customized to meet a given need or modified over time without touching the code
  - Enable business analysts (non-technical personnel) to create and/or modify business rules

- **Discuss BRMS Lessons Learned**

- **Discuss Rules Management Lessons Learned**

The BRMS Rules Experiment

- **Examined functional completeness for use on a Case Management Project**
  - Primary Use Case: Data Validation and Consistency Checking
  - General Purpose Use Cases: Work Assignment, Scheduling, many others
  - Operational Modes: Batch, On-Demand
- **Examined operational characteristics**
  - Compatibility with the rest of the chosen architecture (Tomcat, Informix)
  - Local Rules Tailoring (hundreds of sites)
  - Release Management & Versioning
  - Usability
  - Performance / Scalability
  - Fault Tolerance
  - Security
- **Scope was originally intended to encompass the best commercial and open source products**
- **Sponsor limited scope to the rules portion of a specific product: Red Hat BRMS (the then current version, version 5.3.1)**
  - Commercialized version of the open source Drools product
  - Already in use in other parts of the sponsor

Rules Management with BRMS

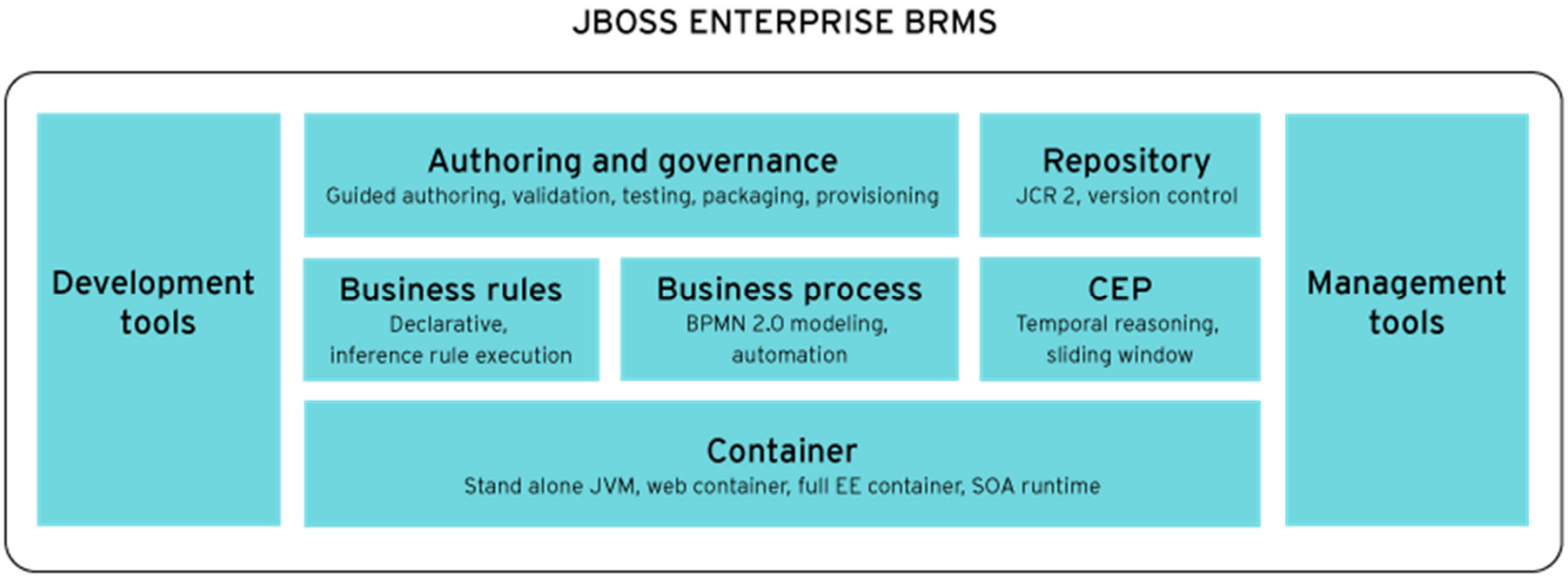
Statistical Reporting Validation Rule Types

- **Format Check**
  - Checking the format of a field (e.g., date, numeric, length)
- **Internal Consistency Check**
  - Checking that multiple fields within a single fact are consistent with one another
- **External Consistency Check**
  - Checking that fields from multiple facts and possibly different fact types are consistent with one another
- **Code Table Check**
  - Checking the validity of a field against a table of values and effective date ranges

MITRE

Overview of JBoss Enterprise BRMS

Components and Capabilities



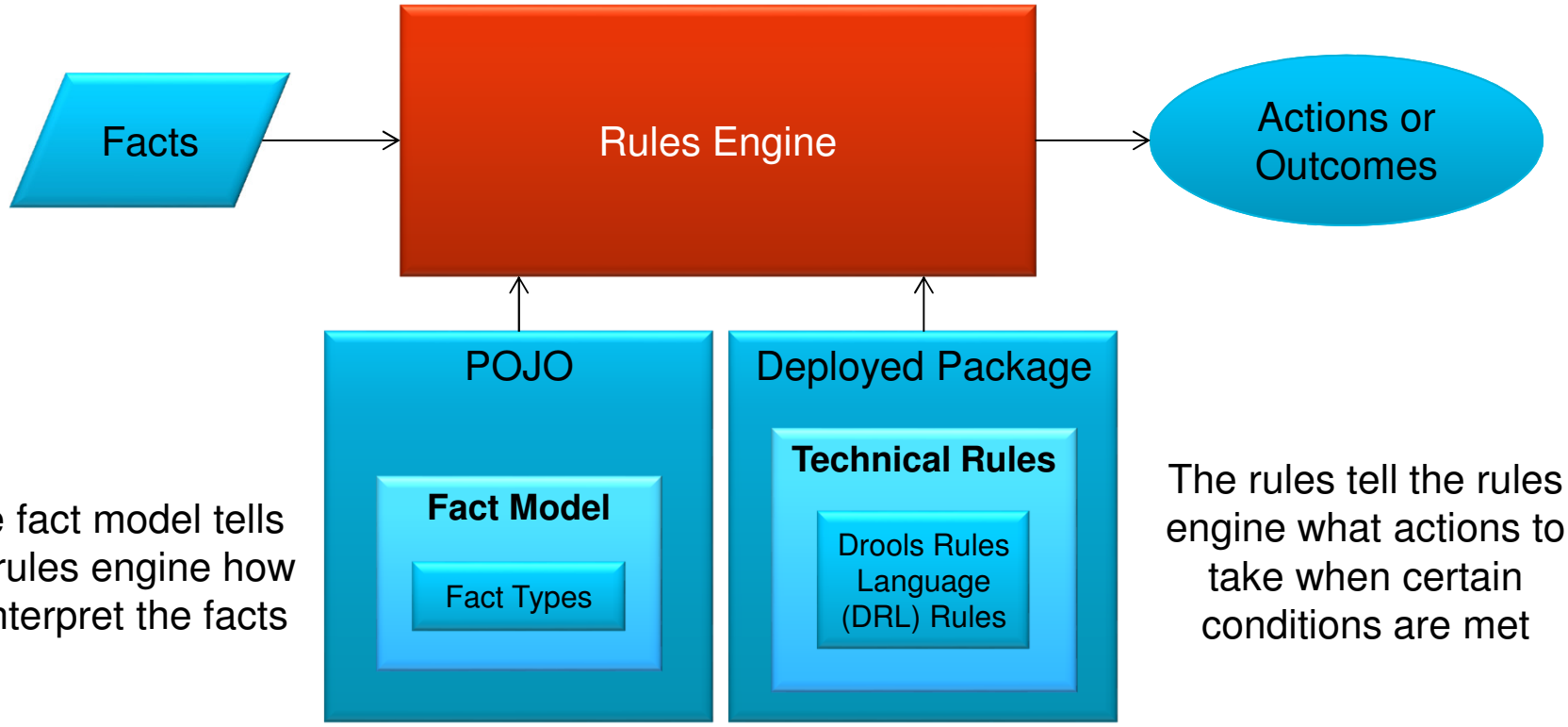
MITRE

Rules Terminology

Facts are the data records to be processed

The rules engine applies the rules to the facts

The rules engine takes the specified actions based on the rules that fired



The fact model tells the rules engine how to interpret the facts

The rules tell the rules engine what actions to take when certain conditions are met

MITRE

Rules Terminology Definitions

Decision Table Rules	A type of business rule defined through a Web-based decision table editor or through import of a spreadsheet that allows many related rules to be edited and managed together
Drools Rules Language (DRL) Rules	A Drools-native rule language. All other forms of rules within Guvnor are ultimately converted to this format.
Fact	A data record against which the rules will be applied
Fact Type	A simple or compound data type corresponding to the data records the rules will be applied against
Fact Model	The collection of fact types used by a particular package
Guided Editor Rules	A type of business rule defined through a Web-based editor that assists the user in developing the rule
Guvnor	A Web–based interface supporting rules creation, modification, and management
Knowledge Base	A container for one or more packages
Package	A container for related rule assets, including the corresponding fact model
Test Scenario	Allows a developer to define rules tests and execute them within the Web interface

MITRE

Two Approaches to Rule Definition

- **Condition-Action Rules**
  - Drools Rules Language (DRL) Editor
    - Create rules using a text editor in the DRL format
  - Domain-Specific Language (DSL)
    - Allows DRL expressions to be replaced with domain-specific or English equivalents
  - Guided Rule Editor
    - Guided the user in the creation of a rule through context-sensitive options
- **Decision Tables**
  - A way to generate rules driven from data entered in tabular form
  - Useful if a collection of rules exist that share the same fact types, conditions and actions, but vary by the data values each uses
  - Not recommended for rules that do not follow a set of templates, or where there are a small number of rules
  - Each row of the table provides data that is combined with a template to generate a rule
  - Provides control over what parameters of rules can be edited, without exposing the rules directly
  - BRMS supports a spreadsheet approach for managing decision tables, but also provides a Web-based alternative, the Guided Decision Table Editor

The Structure of a Rule

Drools Rules Language (DRL) Example

```
rule "name"
    when
        Left-Hand Side
    then
        Ride-Hand Side
end
```

Conditions

Actions

- **A rule is made of *conditions* and *actions***
- **When all the conditions are met, a rule may "*fire*"**
  - i.e., the actions will execute
- **Example**

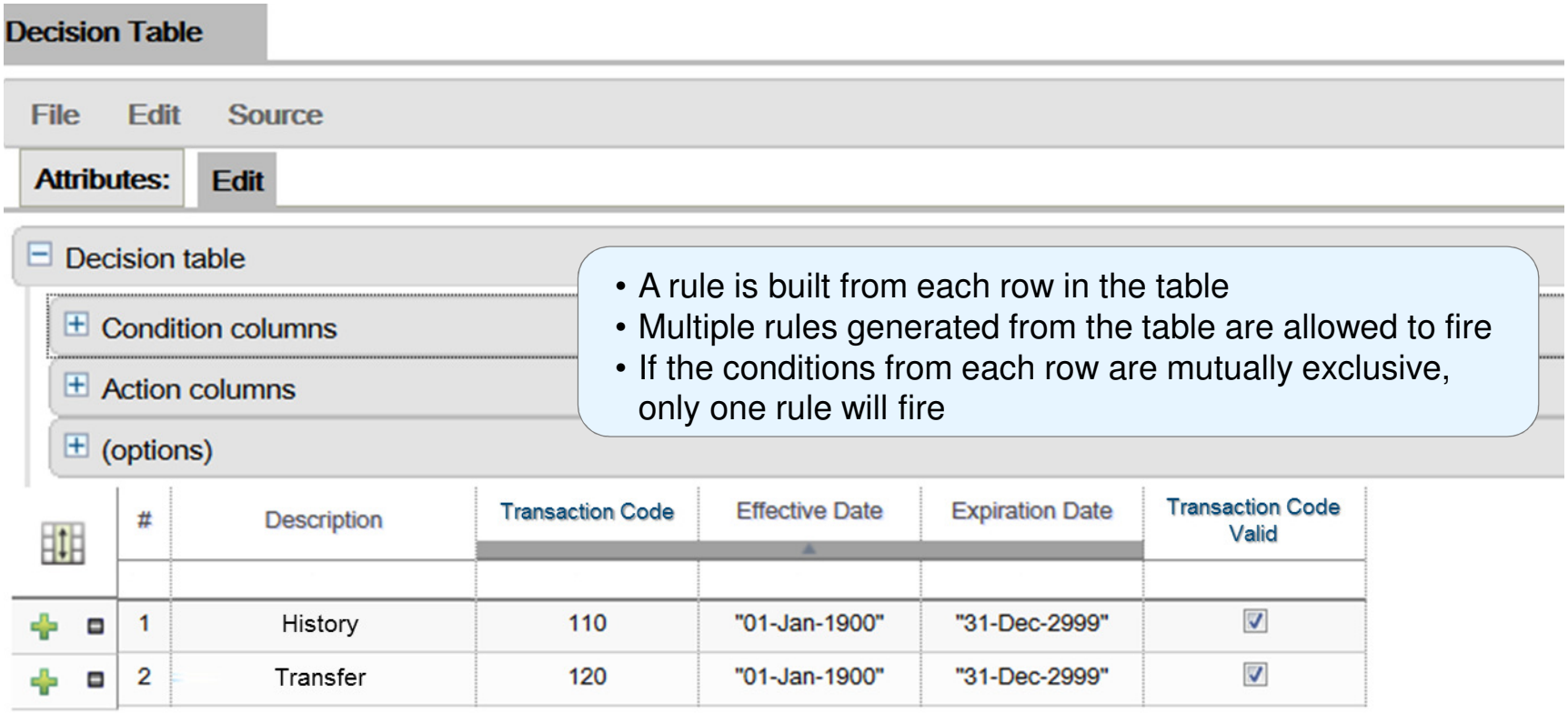
```
rule "Hello Carol"
    when
        Person ( name == "Carol")
    then
        System.out.println( "Hello Carol" );
end
```

When a record of type *Person* is seen with a name field equal to "Carol"…

…send "Hello Carol" to the console

MITRE

A Guided Decision Table Example



#	Description	Transaction Code	Effective Date	Expiration Date	Transaction Code Valid
1	History	110	"01-Jan-1900"	"31-Dec-2999"	☑
2	Transfer	120	"01-Jan-1900"	"31-Dec-2999"	☑

Translation (for the first row):

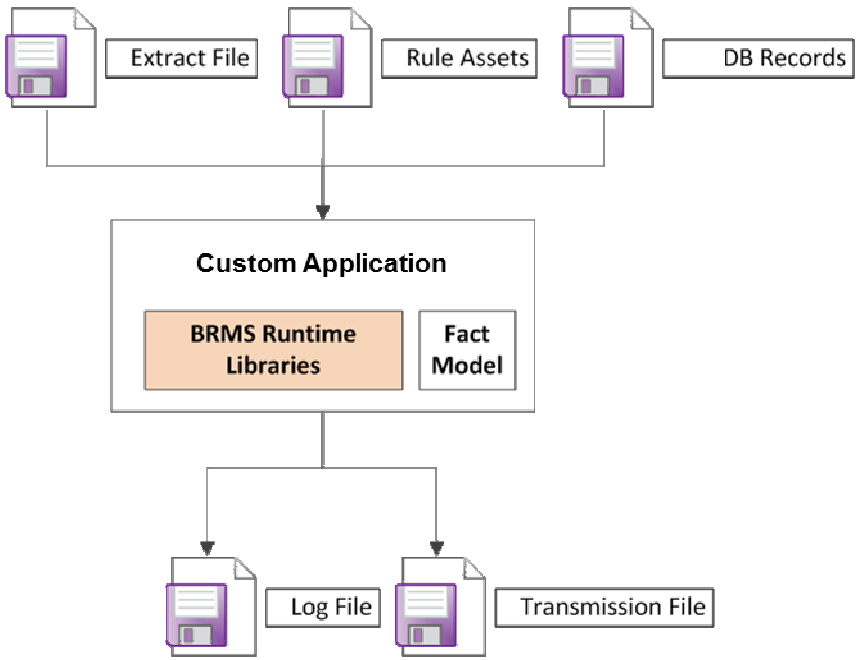
When a record is seen where
- the Transaction Code is 110, and
- the Transaction Date is >= 01-Jan-1900 and <= 31-Dec-2999

Set the record's Transaction Code Valid field to true.

MITRE

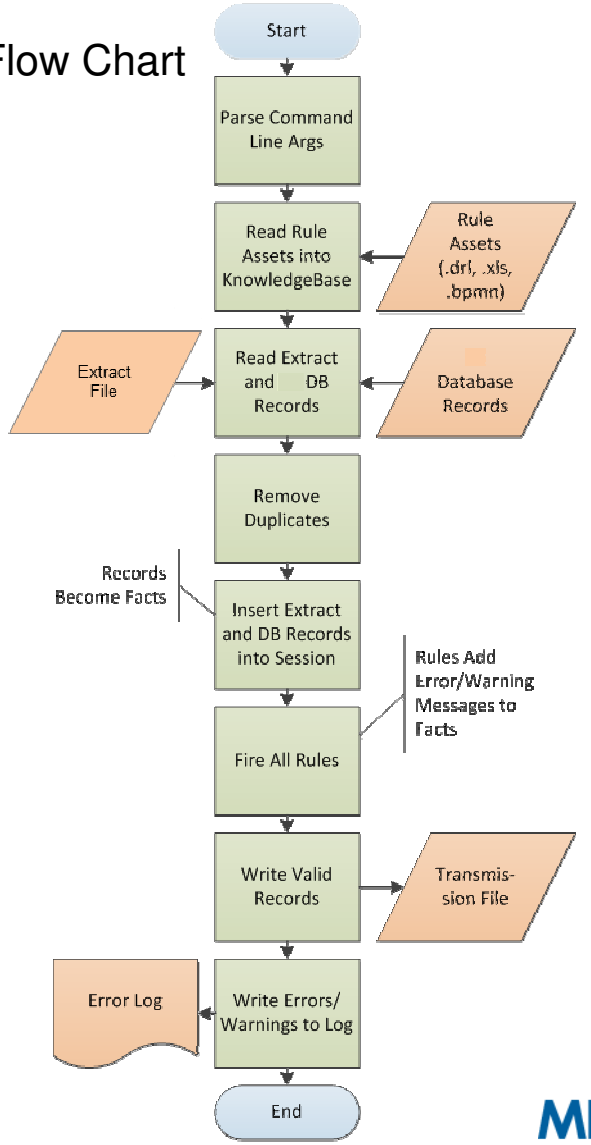
Embedded Rules Solution

An Application Utilizes the Rules Engine API



BRMS Test Harness implementing Batch Processing Data Validation

Test Harness Flow Chart



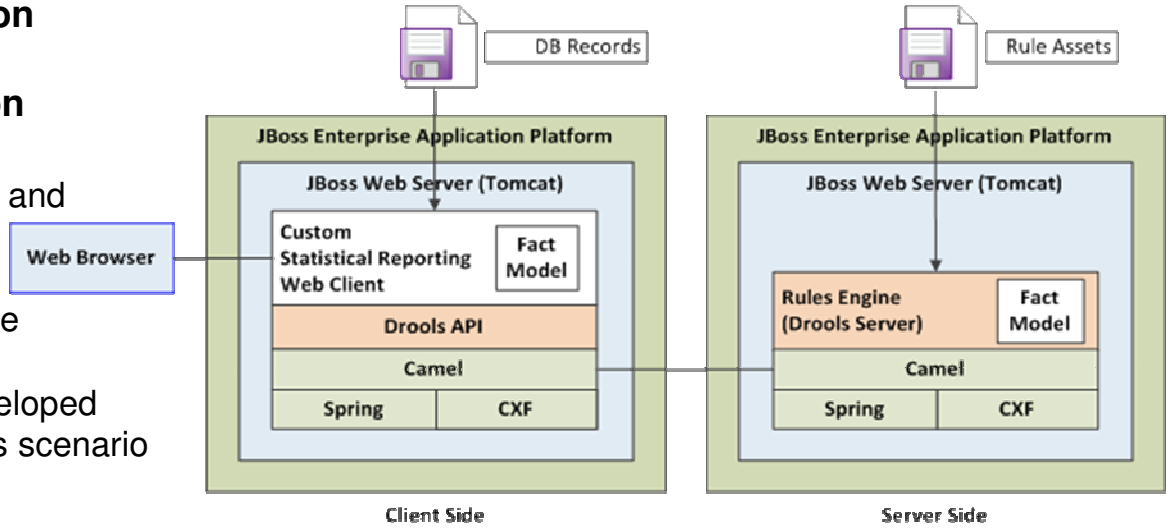
MITRE

Service-Based Rules Solution

Drools-Camel Server Sample Application

BRMS Service Configuration implementing On-Demand Data Validation

- Elements shown in **red** are Drools and BRMS components
- Elements in **green** are open source infrastructure components
- Elements in white are custom-developed components built to implement this scenario



In this scenario:

1. A Web-based Client presents a form that prompts the user to provide a data record
2. The user completes the form
3. The Client converts the Web form data into a fact and checks for a duplicate record in the database (labeled DB Records above)
4. If the fact is not a duplicate, the Client serializes the request containing the fact to the Drools Rules Engine (the Server) via Camel
   -- Camel is a lightweight integration framework
5. The Server reconstitutes the fact and fires the preloaded rules against the fact
6. The Server sends the results back to the Client, which presents the results through the user's Web browser

MITRE

Lessons Learned from BRMS Rules
(1 of 3)

- **BRMS was able to implement the range of rules needed to support the primary use case, as well as many of the secondary use cases**
- **BRMS 5 did not provide a service-based offering**
  - We used the Drools-Camel Server (DCS) sample application to achieve our goals
  - DCS will become a BRMS service-based rules capability in the next major release of BRMS
- **Based on the experiment's usability evaluation, BRMS rules authoring should be done collaboratively by technical personnel and business users**
  - Business users would be unlikely to be able to define their own rules without assistance
  - However, business users may be able to perform limited rule customization
- **BRMS does not support the ability to define local rules that can override globally-defined rules**
  - This capability is needed by our customer
    - Has hundreds of sites, each with their own local rules that take precedence over enterprise-wide rules (with some exceptions)
  - Our team was able to custom build this capability on top of the libraries provided

MITRE

Lessons Learned from BRMS Rules
(2 of 3)

- **Performance / Scalability**
  - Red Hat states that solutions using BRMS with 700K rules and millions of facts yield response times in milliseconds[1] (though no statement was made about the complexity of the rule set)
    - Based on the embedded BRMS library implementation
  - Vertical Scalability: in our tests, the DCS configuration did not scale well when additional CPUs and memory were added
    - Processing time was not the significant factor in response time
    - 90% of the total response time was due to delays from data marshalling/unmarshalling, with the rules engine contributing <10% regardless of arrival rates
    - Lacked the time to fully explore configuration options that might have improved scalability
  - Horizontal Scalability: instantiating several rule engines in separate VMs would have been a logical way to scale
    - This approach was not pursued because it had licensing implications
  - However, even under very high loads (well above that expected under operational conditions for this customer) the engine processing time remained in the *msec* range
  - Rule engines are memory intensive applications; careful management of the knowledge sessions is needed to avoid memory leaks

[1] E. D. Schabell, "JBoss Enterprise BRMS The Road to Large Scale Enterprise Solutions," 23 October 2012, p. 12. [Online]. Available: http://www.slideshare.net/eschabell/best-practices-webinar-14845040. [Accessed 30 July 2013].

Lessons Learned from BRMS Rules
(3 of 3)

- **Fault Tolerance**
  - BRMS is usually deployed within a Java EE environment (e.g., Tomcat or JBoss) and can leverage the clustering options in that environment to achieve fault tolerance and part of its scalability
- **Security**
  - BRMS includes a JAAS-compatible security model for its rule authoring component (Guvnor), though by default BRMS security is controlled by configuration files stored on the application server
  - BRMS supports role-based access that can be used to manage access to rules
    - Admin privileges are given to each user by default
  - It appears possible to have BRMS leverage identities managed elsewhere by JAAS, though BRMS defines its own roles that would have to be used by JAAS
  - Package signing (a way to make rules tamper-resistant) needs to be considered early in the design process since there are issues in trying to employ it after the fact
    - Packages are the units of deployment for rules in BRMS 5
    - For our purposes a single package was created for each court type-rule domain combination (e.g., district court statistical reporting)

Lessons Learned about Rules Management *(1 of 2)*

- **A rules solution is overkill for many applications. Choose a rules engine if you can answer "Yes" to all of the following questions[2]**
  - Does the algorithm involve significant conditional branching or decision making?
  - Are 3 or more conditions present in the rules (i.e., are the rules complex)?
  - Are the rules subject to periodic change and/or localization?
  - Is the code to be maintained over time?
  - Is performance not among the chief driving concerns of the system?
  - Can the project afford the cost and schedule of a rules solution?
    - E.g., licensing, training
    - In general, projects must have a duration greater than a year for the ROI to pay off
  - Am I driving the application with data?
    - Rules are not intended to be used in a procedural manner
    - Instead, a rule 'fires' when the specified conditions are met by a new fact or when modifications are made to an existing fact
- **Once it is determined a rules solution fits the system needs, consider the following[3]**
  - Am I allowing my rules to be opportunistic?
    - Concentrate on what conditions should cause a rule to fire, not when it should fire
  - Am I using proper rule-based control techniques?
    - Attempts to create an order the rules will fire in should be avoided; properly defined rules will seldom need an ordering
  - Am I forming my patterns and actions correctly?
    - Do not place actions in the LHS of a rule; only conditions stating what should fire the rule should be present in the LHS
    - Actions in the RHS can contain Java invocations, member function invocations of fact records, and even functions in other programming languages

[2] G. Rudolph, *Some Guidelines For Deciding Whether To Use A Rules Engine*, July 2008. http://herzberg.ca.sandia.gov/guidelines.shtml.
[3] J. Morris, *The Zen of Jess II: A Jess Mantra*, October 2005. http://www.jessrules.com/jess/zen.shtml.

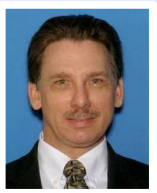
MITRE

Lessons Learned about Rules Management *(2 of 2)*

- **These tools date back to Expert Systems from the '80s, but there has been little progress on standards definition in this technology domain**
- **Tools in the rules management domain all attempt to solve the same problem**
  - Externalization of business rules from the body of the code, enabling rules to be customized to meet a given need or modified over time without touching the code
- **However, each tool has its own approach to solving the problem**
  - There is no migration path from one tool to another
- **The ultimate goal of these tools – not yet achieved – is to put business rules in the hands of the business stakeholders**
  - To provide the flexibility needed to cover the range of possible business rules, the business rules languages are far from intuitive and require significant technical understanding
  - Best practice today involves business stakeholders and technologist well versed in the rules language working together to create or modify rules

MITRE

Contact Information



MITRE

- **Cris Hutto**
  - **Lead Software Architect**
  - **jchutto@mitre.org**
  - **703-983-5719**