Rule Engines and Java Technology

NYJavaSIG

May 23, 2001

Colleen McClintock
JRules Product Manager
cmcclintock@ilog.com

Agenda

- Business Rules
- Business Rule Engines
- Inside the Rule Engine - performance tips for writing rules
- J2EE Pet Store Demonstration
- J2EE Integration
- Rule Engine Standards
- Q&A

Case Study: Loan Purchasing

- Major institution in the secondary mortgage industry
- Loan Purchasing Application processes over 5,000 loans per day
- Maintained by business users and a business analyst since 1996 with no developer involvement!
  - Monthly business rule changes
- Turnaround time for business policy changes went from 1-3 months to 3 days

Java and Business Rules- *"Write Once, Run Away!"*
Chris Roberts , Sun Microsystems

Benefits of Business Rule Technology

- Business rule engine can be a standard component in system architecture
- Improves the software development process by:
  - Lengthening software lifespan
  - Minimizing time required to modify software when business and market conditions change
  - Incorporating business people into the software development and maintenance process

Business Rules

- Statements of business policy that describe and control the structure, operation, and strategy of an organization

IF LIEN TYPE IS FIRST MORTGAGE
THEN OCCUPANCY STATUS MUST BE PRINCIPAL RESIDENCE

PORTFOLIO MUST HAVE <= 30% IN TECHNOLOGY SECTOR

PRIMARY BENEFICIARY ALLOCATION + SECONDARY BENEFICIARY ALLOCATION MUST = 100%

IF INVESTOR AGE IS UNDER 35
AND INVESTMENT OBJECTIVE IS GROWTH AND INCOME
THEN RECOMMEND 50% GROWTH AND INCOME STOCKS

IF AVERAGE DAILY BALANCE EXCEEDS $5000
THEN RECOMMEND INTEREST BEARING ACCOUNT

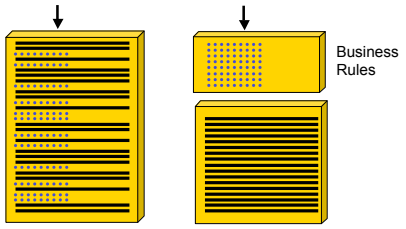
Identifying and Modeling Business Rules

- Identify business rules during requirements and analysis phases
- Use rules to represent business policy that changes frequently
- Identify rules that will be exposed to business people
- Rules should be atomic units of business policy
  - Smallest expressible standalone unit of policy
  - Change independently
- Capture rules in "structured-English" then express using terms from UML model or class diagrams during design phase

Implementing Business Rules

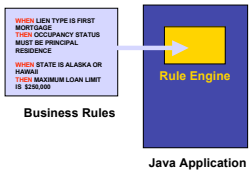
We would like to take business rules from here…

to here.



Business Rule Engines

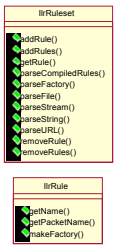
- Business rules are:
  - Expressed declaratively
  - Externalized from application code
- Rule engine:
  - Is a Java object in your application
  - Evaluates and executes business rules



Rule Engine Class

- Rule engine class consists of methods to control the rule engine
  - Created with an associated rule set
  - Set of of objects known as working memory
- Rules applied to objects of working memory
- Rule engine has methods to trigger rules on demand

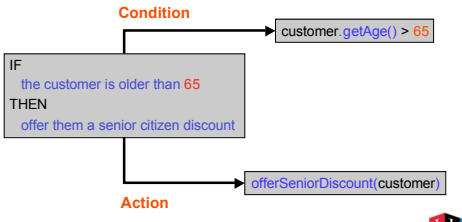
Rule Set Class



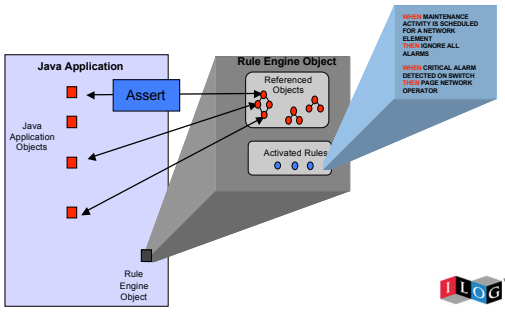
- Rule set class maintains a collection of rules
  - Rules are added using one of the parse methods
  - Rule sets may be shared by multiple rule engines
  - Rules exposed through instances of Rule class

Rule Translation

- Rules operate on the Java objects in your application

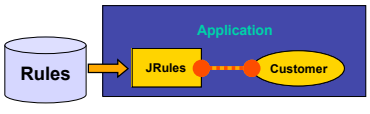


Rule Engine Environment



Business Rule Engines

Rule Engine Integration

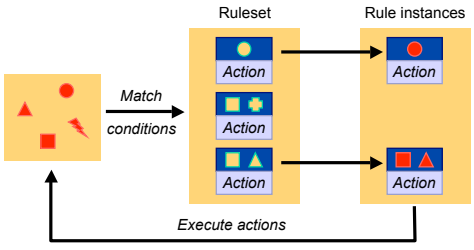


Fire the rules

```
IlrContext engine = new IlrContext();
engine.getRuleset().parseFileName("rules.ilr");
engine.assert(customer);
engine.fireAllRules();
```

Inside the Rule Engine

Rule Evaluation Cycle



Inside the Rule Engine

Example of Update

```
IF
  ShoppingCart.purchaseAmount > $200
THEN
  Customer.category = "Gold"

IF
  Customer.category == "Gold"
THEN
  ShoppingCart.discount += 5%

IF
  ShoppingCart.discount > 15%
THEN
  ShoppingCart.discount = 10%
```

Inside the Rule Engine

Naïve implementation

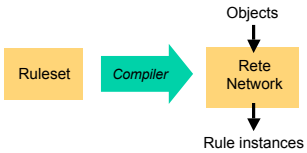
- Iterate over objects and rules at each working memory change
- Memorize rule instances to avoid double execution
- Not scalable
  - 100 rules and 10,000 objects

    =>1,000,000 tests at every change

Inside the Rule Engine

The Rete Network

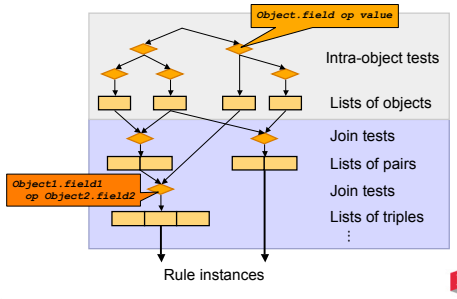
- Rule conditions are compiled into a network



- Minimizes the number of evaluations
  - Tests are shared between rules
  - Changes are propagated incrementally

Inside the Rule Engine

Shape of Rete Network



Example of Rule Conditions

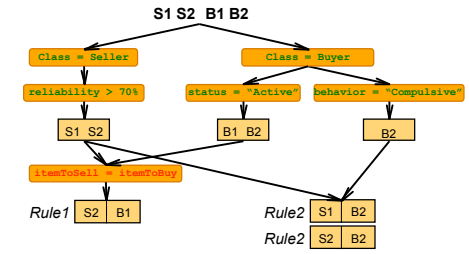
Rule1

```
IF
  Buyer.status = "Active"
  AND Seller.itemToSell = Buyer.itemToBuy
  AND Seller.reliability > 70%
THEN
  ...
```

Rule2

```
IF
  Seller.reliability > 70%
  AND Buyer.behavior = "Compulsive"
THEN
  ...
```


Example of a Rete Network




```
IF
  Buyer.status = "Active"
  AND Seller.itemToSell = Buyer.itemToBuy
  AND Seller.reliability > 70%
THEN
  ...
```

```
IF
  Seller.reliability > 70%
  AND Buyer.behavior = "Compulsive"
THEN
  ...
```

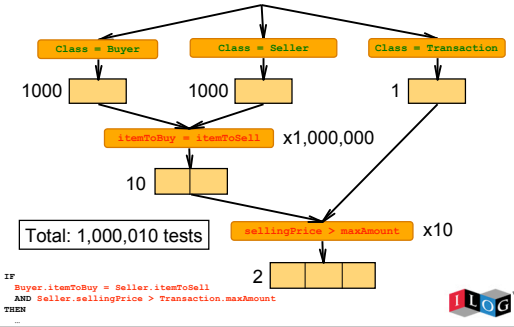

Performance Hint #1

- Join tests are combinatorial
- Cascade of tests follows the order of tests in rules
- Put the most restrictive test first
- Can dramatically reduce combinatory explosion

Rule 1

```
IF
  Buyer.itemToBuy = Seller.itemToSell
  AND Seller.sellingPrice > Transaction.maxAmount
THEN
  ...
```


Network for Rule 1

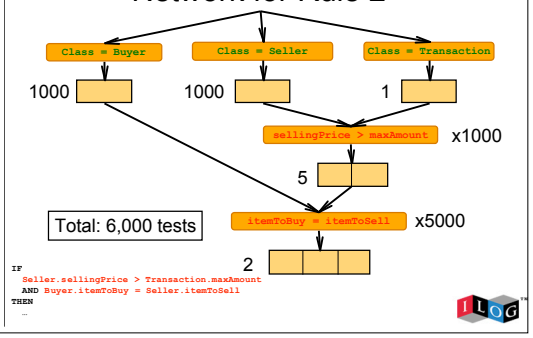



```
IF
  Buyer.itemToBuy = Seller.itemToSell
  AND Seller.sellingPrice > Transaction.maxAmount
THEN
  ...
```


Rule 2

```
IF
  Seller.sellingPrice > Transaction.maxAmount
  AND Buyer.itemToBuy = Seller.itemToSell
THEN
  ...
```

Network for Rule 2



Performance Hint #2

- Test sharing improves performance
- Some Rete compilers are sensitive to the syntax and order of tests
- Use uniform syntax in rules
- Use same order for tests in rules

Rules with No Shared Tests

Rule1

```
IF
  Seller.status = "Active"
  AND Seller.sellingPrice > Seller.maxPrice
  AND …
```

Rule2

```
IF
  Seller.maxPrice < Seller.sellingPrice
  AND Seller.status = "Active"
  AND …
```


Rules with Shared Tests

Rule1

```
IF
  Seller.status = "Active"
  AND Seller.sellingPrice > Seller.maxPrice
  AND …
```

Rule2

```
IF
  Seller.status = "Active"
  AND Seller.sellingPrice > Seller.maxPrice
  AND …
```


Writing Rules for Performance

- Put the most restrictive tests first
- Use a uniform syntax in rules
- Use the same order for tests in rules

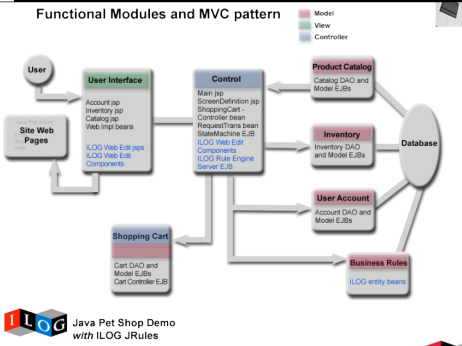
Rule Engines in J2EE

- Rule engine can be embedded directly in an EJB
  - Stateful session bean or entity bean (rule context must be serializable)
  - Rules can reference EJB objects (entity beans have a different object identification mechanism - `EJBObject.isIdentical()`)
- EJB Rule engine server
  - Implemented as stateless session bean
- External rule engine server
  - Connect to via stateless session bean

Demonstration

- Java Pet Store sample application enhanced with a rule engine
- Business rules used to display banners and apply discounts to shopping cart items
- Web-based rule editor used to create and edit rules via the administrator interface

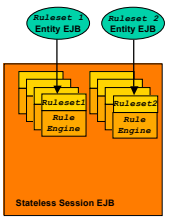
Functional Modules and MVC pattern



Java Pet Shop Demo *with* ILOG JRules

JRules Rule Engine Server

- Rule Engine Server EJB
  - Stateless session bean
  - Retreives ruleset from entity bean
  - Pooled by container
- Pools rule execution contexts
  - Saves ruleset parsing and context creation time on each subsequent invocation of server
- Call rule engine server: `invokeRules(String,IlrSessionData)`
  - Primary key identifies ruleset entity bean
  - Session data contains objects to be evaluated and returned



JRules Rule Engine Server

JRules Rule Engine Server

```
public static IlrContextSessionHome getIlrContextSessionHome() throws
javax.naming.NamingException  {
          InitialContext initial = new InitialContext();
          Object objref = initial.lookup(JNDINames.ILR_CONTEXTSESSION_EJBHOME);
          return(IlrContextSessionHome)PortableRemoteObject.narrow
                  (objref,IlrContextSessionHome.class);
  }
```

- Get a reference to the home object
  - JNDI look-up

JRules Rule Engine Server

```
private static void  initRuleEngine(  ) {
      try {
        contextHome = EJBUtil.getIlrContextSessionHome();
        session = contextHome.create();
          } catch (javax.naming.NamingException ex) {
                  Debug.println("NamingException in initRuleEngine caught:" + ex);
          } catch (javax.ejb.CreateException ex){
                  Debug.println("EJB Create exception in initRuleEngine caught:" + ex);
          } catch (Exception ex) {
                  Debug.println("initRuleEngine caught:" + ex);
          }}
}
```

- Use the home object to create the Rule Engine Server EJB
  - EJB container pools stateless session beans and retrieves bean from pool to service request
  - EJB container creates and destroys beans to resize the pool

JRules Rule Engine Server

```
public Object[] processCart( IlrContextSession session, Collection items, String ruleset ){
    try {
       Object[] cartItems = items.toArray(); // array of shopping cart items
       Object[] discountItems = {};          // declare array of returned discount items
       // IlrSessionData has two parameters - the argument objects and the result object
       IlrSessionData data = new IlrSessionData(cartItems,discountItems);
       // invoke the Rule Engine Server and return the response
       Object[] response = session.invokeRules(ruleset,data);
       /* response contains returns an array containing two elements. The first element is
          the result object, as specified in data. The second element is a java.lang.String
          containing the messages printed to ?context.out by the executed rules */
       return response;
    } catch (Exception ex) {
         Debug.println("Discount Pricing caught:" + ex);
    }
```

- Create IlrSessionData
- Call invokeRules method

Rule Engine Standards

- Java Rule Engine API – **JSR 94**
- Extend J2SE and J2EE with a standard rule engine API
- JSR – http://java.sun.com/jcp/jsr/jsr_094_ruleengine.html

OASIS

XML.ORG

- **SRML** – Simple Rule Markup Language
  http://www.oasis-open.org/cover/srml.html
- Subset of rule language constructs common to forward chaining engines
- Share and execute rules across applications
- Other business rule XML initiatives:
  - RuleML http://www.oasis-open.org/cover/ruleML.html
  - BRML http://www.oasis-open.org/cover/brml.html

Summary

- Java business rule engines can:
  - Help you develop more adaptable applications
  - Help you deliver business rules to your business people
- Features to consider when evaluating vendors:
  - 100% Pure Java
  - Direct access to Java Objects
  - Product architecture and API
  - Performance
  - EJB integration
  - XML data access/rule representation
  - Support for delivering business rules to business people
- Evaluate products against your application requirements- a feature checklist comparison is not very useful!

Questions