
ORACLE®

A Business Rules Perspective on a Standard Rules Language

Gary Hallmark

April 28, 2005

Outline

- Benefits of business rules
- Rule enabled business application lifecycle
- Benefits of standardization
- Business rule requirements by example
- Conclusion: standardize current practice

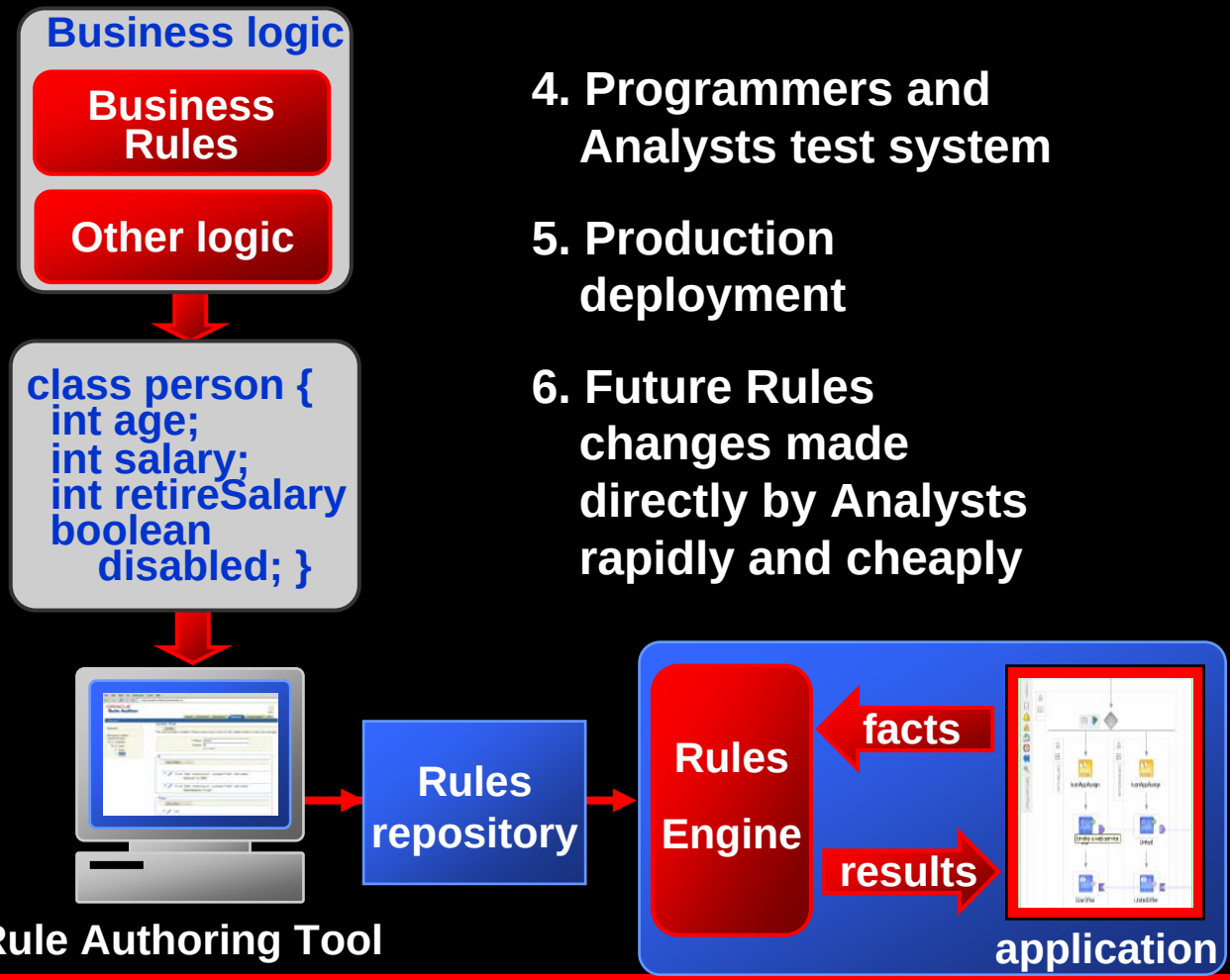
Benefits of Business Rules

Greater agility, better decisions, lower cost

- capture important and changing business policies
- store in a repository separate from application code
- business analysts, not programmers, modify rules in response to
  - changing regulations
  - competitive pressure
  - customer demand

Rule Enabled Business Application Lifecycle

1. Programmers and Business Analysts separate Business Rules from other logic
2. Programmers provide Facts and Actions to Rule Authoring Tool
3. Analysts define Rules



4. Programmers and Analysts test system
5. Production deployment
6. Future Rules changes made directly by Analysts rapidly and cheaply

Benefits of Standardization

- Lower cost and risk
- Increase market for business rules
- Improve quality through specialization
  - Natural language and graphical rule authoring tools
  - Rule engines for different environments—J2EE, .NET, database)
  - Rule content—*e.g.* yearly tax law compliance, Sarbanes-Oxley requires standard fact types (ontology), too
- Benefit to Oracle of standard rules and standard middleware
  - Acquired applications (Peoplesoft, JD Edwards, Retek, *etc.*) built on standards are easier for Oracle to integrate in a suite
  - Best of breed applications built on standards are easier for customers to integrate

Business Rule Requirements

- Support Natural Language and Decision Table interfaces
- Hide complexity of facts
- Invoke application methods and define functions
- Execute arbitrary actions
- Control order of rule firing
- Support forward and backward chaining

User Interface

W3C must standardize underlying XML syntax and semantics

- Structured Natural Language

  Business Analysts are not programmers or logicians

  age of driver is less than 30 *not* driver[@age < 30]
  nor op:lt(age(?driver), 30)

- Decision Tables

age greater than	age not greater than	vehicle type	risk
15	25	-	high
25	30	sports	high
30	70	sedan	low

Easy to spot gaps and overlaps

Hide Complexity of Facts

- Simple facts => simple rules, *e.g.* if age of driver is …
- Business analysts should see
  - A business term, *e.g.* “driver”
  - Documentation, *e.g.* “anyone in the household of driving age”
  - A set of named properties, *e.g.* “name”, “age”, “vehicle”
  - A set of methods, *e.g.* “vehicle usage percent”
- Programmers see more detail
  - Data source: XML document, database, Java class, RDF, …
  - Mapping info, *e.g.* “driver” maps to http://xmlns.oracle.com/rules/rate-quote.xsd:driver-info
- A programmer provides a library of fact definitions to the business analyst for use in rules

Invoke Application Methods and Define Functions

- Must be able to invoke methods on underlying application objects, even in rule conditions

  if vehicle usage percent of driver is greater than 0.5
  then …

- Must be able to define functions with rich set of operators in the rule language if business analysts are to maintain them

  function vehicle usage percent of driver is
  10000 / (miles per day of driver * 365)

Execute Arbitrary Actions

- Modify the knowledge base to support inference
  - if a customer spent $500 last month
    then set his status to GOLD
  - if a customer's status is GOLD
    then discount his shipping by 50%
- Append to a log of rule firings to explain decision
- Invoke a credit report web service to retrieve new facts
- Update a database
- Rules that can only describe actions require application code (an interpreter) to execute the descriptions
  - ties rule maintenance to application program maintenance

Control Order of Rule Firing

- Declarative is good
  - Rules can be optimized (*e.g.* Rete, Magic Sets)
  - Rules can be maintained without intimate knowledge of neighboring rules
- Declarative is not good enough
  - Not all rules apply equally all the time
  - Add state facts and extra rule conditions—cumbersome
  - Selectively load rule sets from rule repository
  - Rule priority (*aka* salience) – prone to abuse
  - Rule set focus, rule flow

Support Forward and Backward Chaining

- Forward chaining production rules satisfy most requirements with good performance
- Rete is efficient, proven, and mature technology
- Backward chaining is useful to fetch expensive facts on demand, *e.g.*

  if job status of applicant equals “unemployed”
  or reportee of credit report equals applicant
  and score of credit report is less than threshold
  then deny the loan

  - job status is cheap to check
  - credit report must be purchased and fetched from a web service
  - rule should fetch the credit history only if the applicant is employed

Conclusion

A standard rule language should support the features of mainstream business rules engines

- Market leaders (iLog, Blaze) are on the right track
- Focus on UI, repository, testing, and coverage tools
- Lower cost and risk to increase the number of users


QUESTIONS
ANSWERS