

Accessible Business Rules Project

Country Companies

K. Rasmus

IBM Research

L. Degenaro, I. Rouyellou

IBM Software Solutions

D. Ehnebuske, B. McKee



Genesis

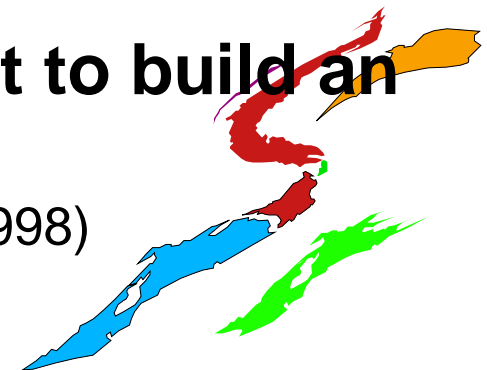
Joint Country Companies & IBM development effort

LUW (Life Underwriting Workstation) project included a framework for externalizing business rules

- CORBA-based, OS/2, C++
- Small number of rule types supported
- Placed in production early 1998

ABR (Accessible Business Rules) project to build an advanced rules framework

- Shipped as part of IBM Component Broker R1.3 (7/1998)



Motivation

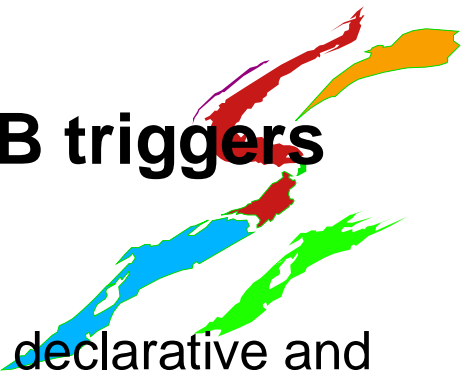
Domain experts see business processes as "themes and variations"

Those variations often implement policies or practices of the business (i.e., business rules)

Business rules have been embedded in applications

- Prone to inconsistency
- Difficult to maintain : "harvesting" is very difficult

Business rules have been embedded in DB triggers

- Hard to develop
 - More technically oriented than business oriented
 - May be hard to understand: SQL based languages mix declarative and procedural code ...
- 

Motivation (cont.)

- ABR allows to structure applications to match built in core behavior with variations specified, managed, and applied externally
 - Rule (variation) behavior is usually straightforward, but time and business-context dependent
 - Increase consistency of business practice
 - Provide better system understanding
 - Cut the cost of application maintenance
 - Enable domain experts to make new variations
- The structuring is done at the object level and fits naturally within it



ABR Overview

- Structured exit points from main code (trigger points)
 - Are explicitly identified during analysis and design
 - Most follow a small set of patterns and are automatically generated
 - Assemble runtime info and query the rule home to find the applicable rules
- An ABR rule is a persistent object encapsulating
 - A piece of code (executed when Rule.fire() is invoked)
 - Implements the "variation"
 - A number of "context" attributes
 - used to select the applicable rules at runtime (query)



Code without/with externalized rules

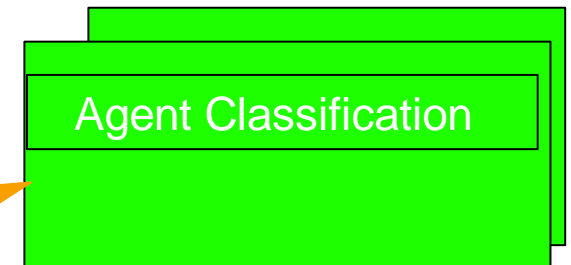
```
mmitPayment(agent, policy,desired_amount)

if( agent->level > 3 &&
  agent->hiringDate < current - 2 years
  && ....)) {
  amount = desired_amount;
}
if (agent_level < 3 && ...) {
  if (desired_amount < 2K && policy ..)
  else if ...
}
if ( //some other test on agent & policy){

/ set a number of variables ..
return (amount);
```

modification by technical people only
No reuse, Prone to inconsistency

```
commitPayment(agent, policy, desired_amount)
{
  CRule=
  findClassifier(agentbased_commitPayment);
  classification =
  CRule->fire(agent,desired_amount);
  Rule= findRule(commitPayment,classification);
  amount = Rule->fire(desired_amount);
  // set a number of variables
  return (amount);
};
```



better "business" understanding,
can be used anywhere where fiscal
commitment, has a non-technical
interface, ..

Approach

Extend application analysis and design process

- Include identifying where, during business object interaction, the points of variability in application behavior arise
- Extract variations from use cases
 - *except when*
 - *unless*
 - *depends on*
 - *may be changing*
- Annotate analysis and design artifacts

Translate those points, during development, into "trigger points" -- pieces of code that interface with the ABR runtime to attach and execute business rules dynamically during application execution



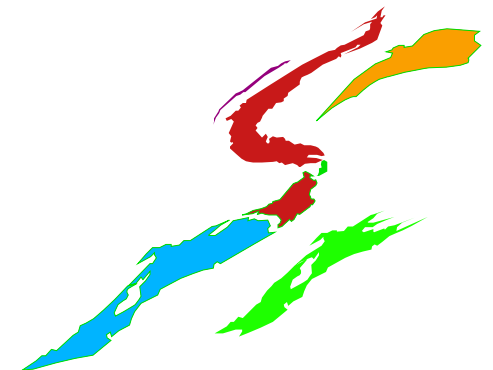
Trigger Point Analysis

What determines which rules are applicable?

- Called "trigger point context"
- Used as search criteria to find rules when the trigger point is encountered in the code

Three basic patterns

- Fixed business context
 - context is fixed (e.g., pre-condition), but applicable rules vary over time
- Situational business context
 - Applicable rules partly depend on business context computed at run time
 - Variable part determined by classification
 - Ex: income level, state of residence, trust
-MaximumLoanAmount(InvestorLevel)
- Jurisdictional context
 - Reviewing process by a hierarchy of jurisdictions



ABR : Runtime and Administration

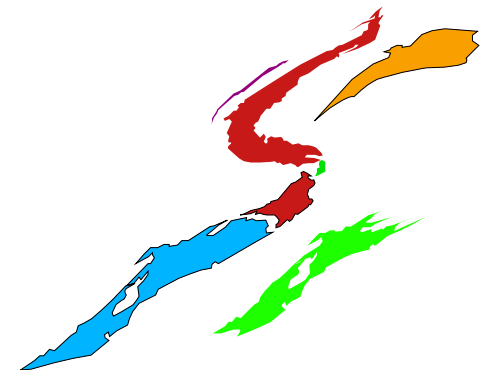
Overview

Runtime

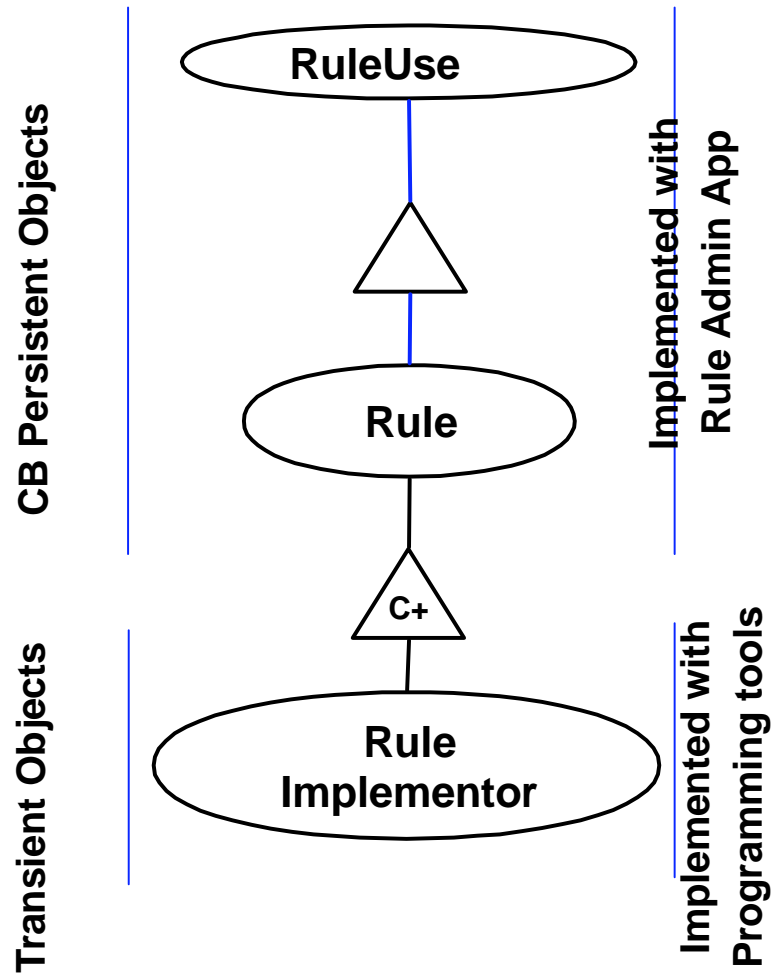
- runtime finds and instantiates rules
- info used to find rules= trigger point context

Administration

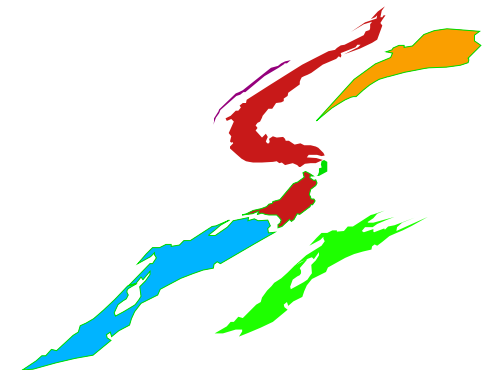
- can alter the behavior of the rule driven application by changing the set of business rules
- search/create/delete/update of rules



Business Rule Structure

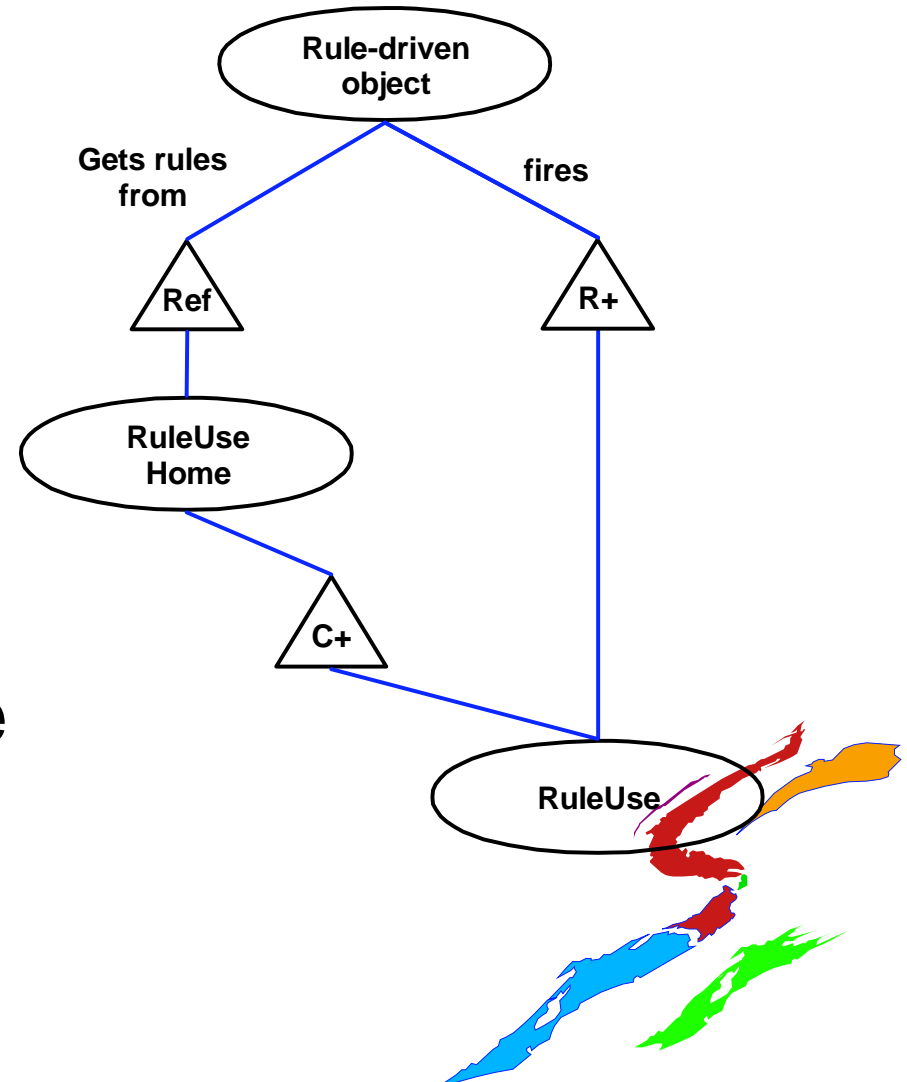


- Links trigger point context to Rule
- Parameterizes rule algorithm
- Implements rule algorithm



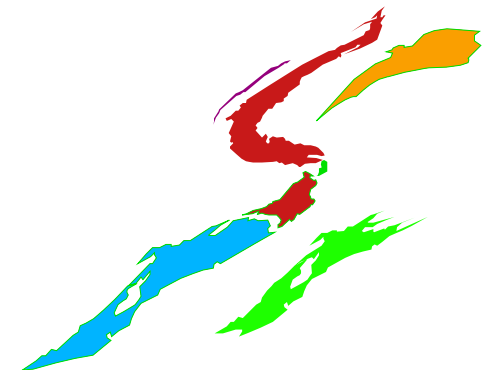
Rule Firing

- Rule_driven object = object with trigger points (TPs)
- Most TPs are generated by framework
- Typical TP code assembles context info, queries the RuleUseHome to find the "right rules", fires them and combines their results



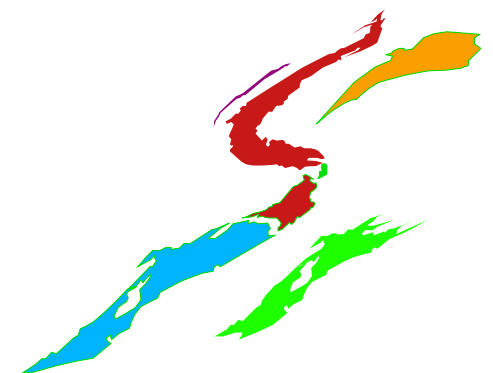
Rule Administration/Management

- Find existing RuleUses and Rules
- Verify many specifications
- Build new RuleUses
- Build new Rules by
 - adding mapping data to "template" rules
 - compounding other rules
 - creating new rule from scratch
- Create new RuleImplementors
 - Implement new algorithms
 - C++ or Java



Our Findings

- Performance
- Where To Trigger and Execute Rules
- Rules in Analysis and Design
- Management of Rule Failures (when a rule returns a "constraint failed" result)
- Rule Administration and Deployment



Performance

To externalize versus embed

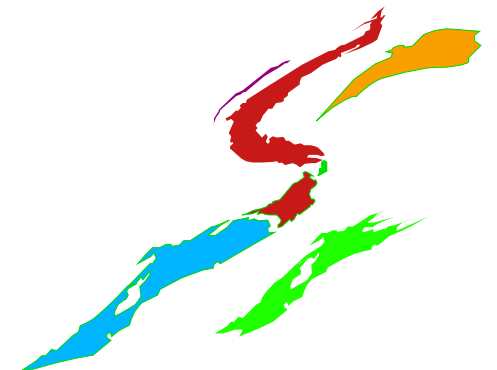
- Cost/benefit decision
- Performance the primary cost

Caching

- Caching "not found"
- Cache invalidation

Removal/relocation of rules as system stabilizes

- Test versus production rules
- Discovery of costly rules



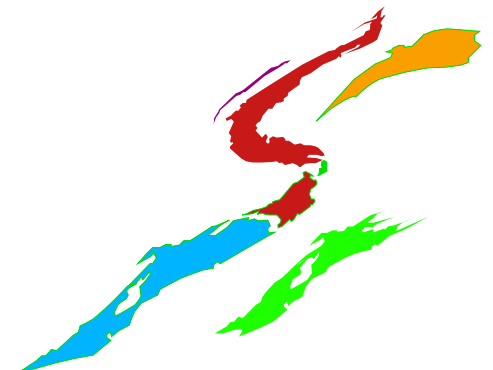
Some numbers (Life Underwriting Project, Country Companies)

- 40 users production system
- 74 classes on server (business view)
- 121 K LOC on the server

180 "company wide" rules ended up being externalized on the server

- 25 reusable rule algorithms (e.g., range)
- 180 derived from those through mapping ...

Client side rules would have introduced 2 to 3 times that number of rules



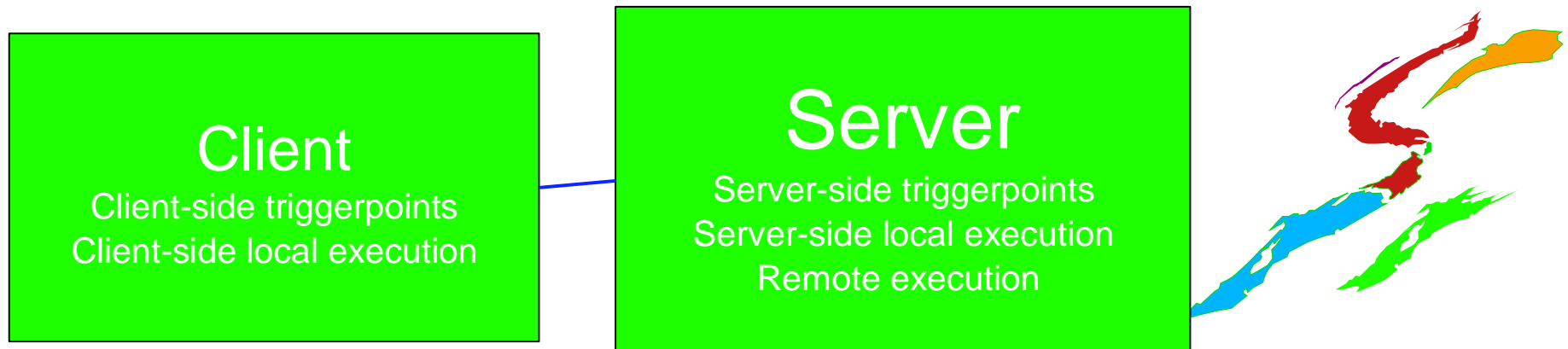
Where to Trigger and Execute Rules

Client versus server versus both

- Security
- Granularity and nearness

Local versus Remote

- Languages and execution environments
- Performance



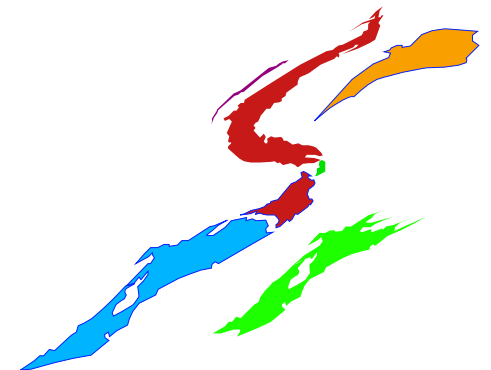
Rules in Analysis and Design

Identifying points of variability

- Use case review
- Reuse

Have analysts and designers write the rules

- Ease of understanding
- Meeting the needs of the users



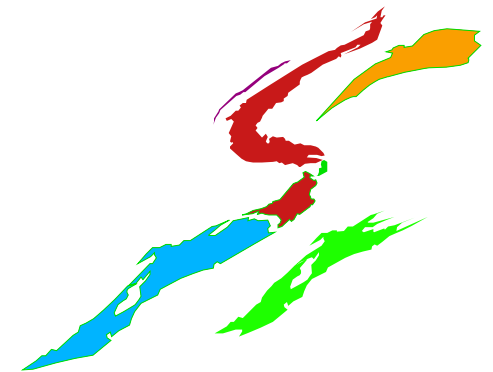
Management of Rule Failures (when a rule returns a "constraint failed" result)

Runtime Exceptions

- Generic exception on every throw
- Bubble up rather than rethrow
- Discovery of situations where exception needs to be handled

Administration time exceptions

- Specific exceptions on throw



Rule Administration and Deployment

Without an administration system

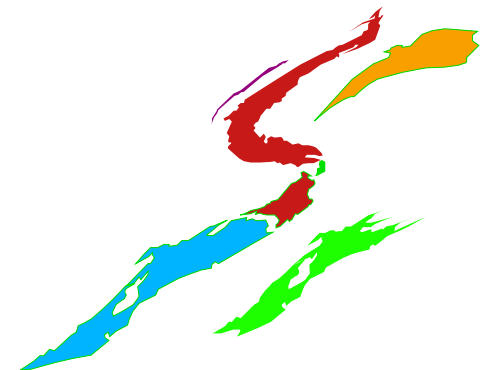
- Word Processors and scripts

With an administration system

- Queries
- Multi-user

Moving from test to production

- Export and load
- Regression testing



Summary

- Distributed object systems have many rules
- Externalizing select rules has benefit
- Biggest hinderance is performance
- Flexibility and responsiveness win in the end

