Business Rules Engine

Flexibility, agility and understanding

Agenda

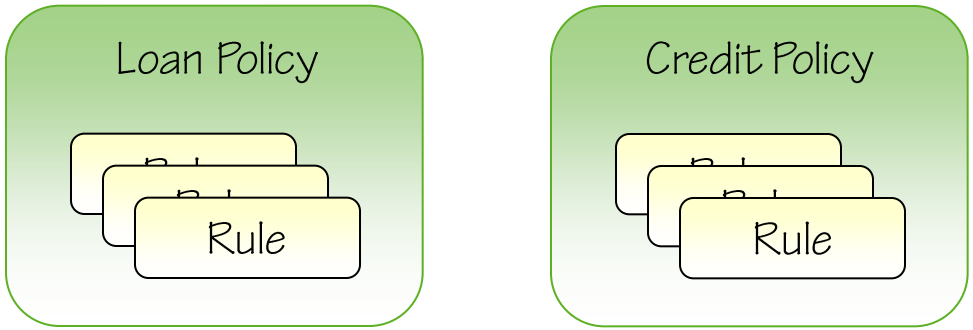
- Why a Business Rules Engine
- Concepts
- Programming
- Managing

Why business rules?

- **Externalizing business rules can increase flexibility and agility**
  - Application rules as easy to change as your users' minds
  - Enable better understanding of the rules and their meaning
  - Enable complex scenarios suited for rule style execution
- **Allow a user with domain expertise to manage the rules**
  - In theory, but tool support is not there
- **Make your life easier as a BizTalk developer**

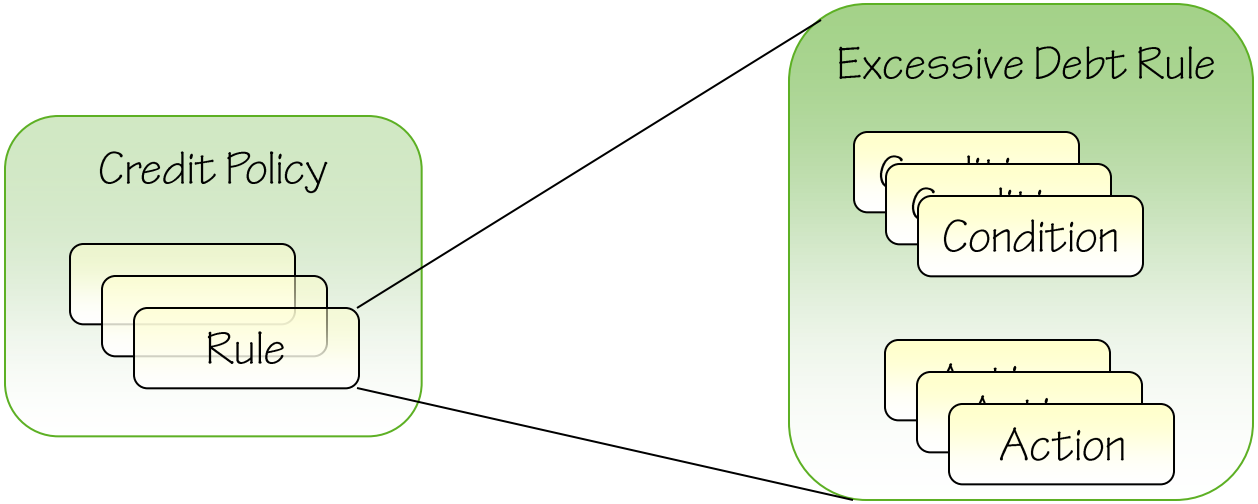
Business rule concepts

- **Policy**
  - A collection of business rules defining one targeted area of logic
  - The unit of deployment, versioning and execution
  - Can be monolithic or broken down into smaller components



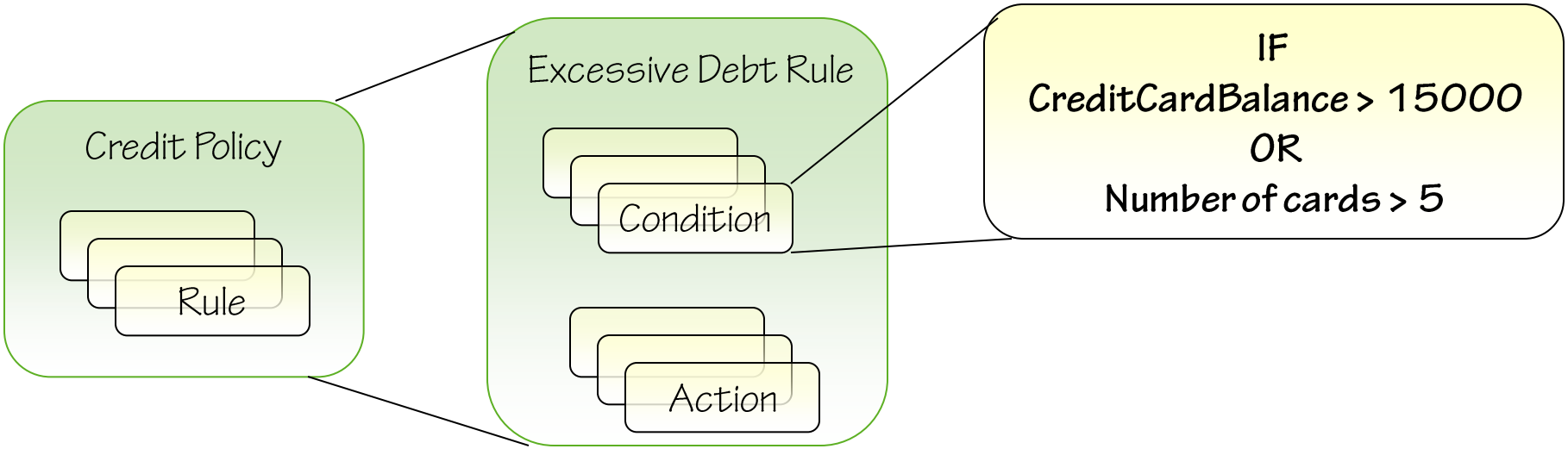
Business rule concepts

- **Rule**
  - A collection of conditions and accompanying actions
  - Rules are collected into a policy to be executed
  - Rules can be prioritized to indicate the order of execution



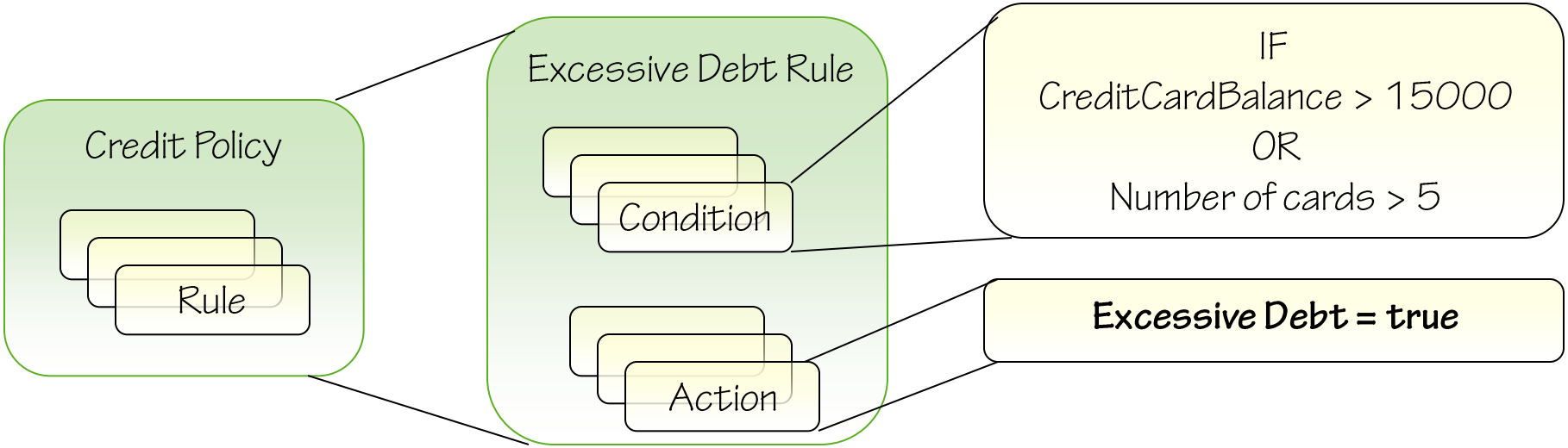
Business rule concepts

- **Condition**
  - An expression of predicate logic to determine if the business rule should perform its action
  - Predicates can be combined using OR, AND, NOT binary operators
  - Predicates include simple comparison, range comparison, Regular Expressions, and XPath existence (custom methods also allowed)



Business rule concepts

- **Actions**
  - Define the steps that should be taken when a rule evaluates true
    - Setting values in XML document
    - Updating SQL database or DataTable
    - Setting .NET properties or calling .NET methods
    - Rule Engine methods



Business rule concepts

- **Facts**
  - The items that rules act on for evaluation and execution
  - .NET types (Simple or Complex)
  - XML documents or fragments (Schema Based)
  - SQL Connection or .NET DataTable/DataRow

Understanding facts

- **Facts represent the objects used in rule evaluation and actions**
  - Facts must be asserted into the engine before rule evaluation
  - Multiple facts of the same type will cause multiple evaluations
  - Multiple nodes in XML or multiple .NET class instances
- **Facts may be asserted in three ways**
  - Passed by the calling application when executing a policy
  - Asserted by a Fact Retriever at runtime (Long Term Facts)
  - Asserted by a Fact Creator at runtime (Testing/Development only)

Understanding facts

- **Fact instances are required even for .NET classes where only static methods are used (e.g. String.Format, DateTime.Now)**
  - 2006 added optional registry settings to change this
- **Facts dragged from the browsers in the rule composer have default values which can be changed to modify behavior**
  - Choose DataConnection versus DateTable/DataRow
  - Modify the XPath statements used to access values

Business rule concepts

- **Vocabulary**
  - Abstractions of facts to make rule creation and editing easier
  - Allows use of natural language for XPath, .NET or SQL
  - Also used for good dev practices (constants & enumerations)
  - Can be versioned and published like policies

Vocabularies

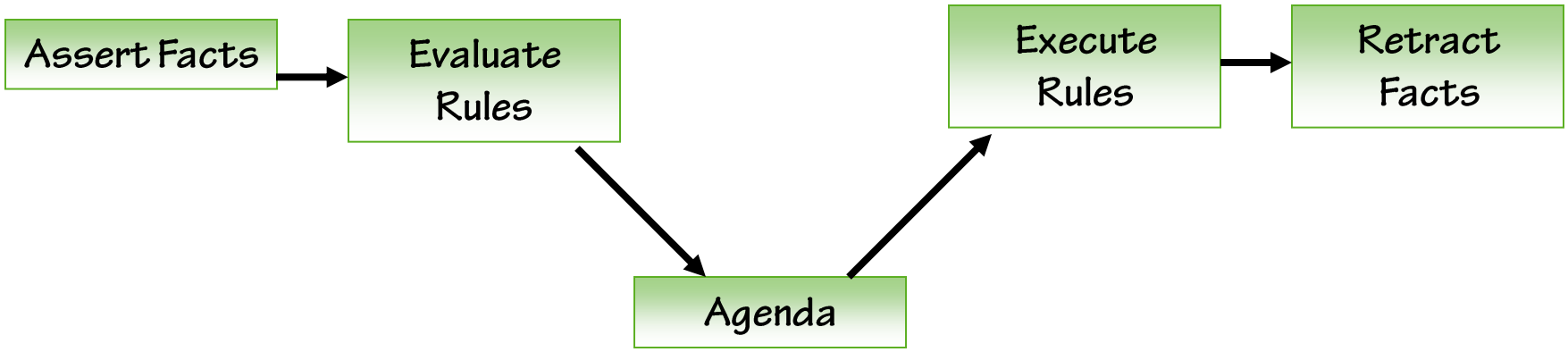
- **Create vocabulary definitions to abstract code details**
  - Allows building policies with statements like
  - Set discount amount = .15
  - Instead of .NET code:
    - ShoppingCart.set_Discount(.15)
  - Or XPath:
    - /*[local-name()='Order' and namespace-uri='http://example.org/order']/*[local-name()='Discount' and namespace-uri()='http://example.org.order'] = .15

Rule engine execution

- **Business rule engine is responsible for "executing" policies**
  - Assertion of facts into the engine
  - Evaluation of conditions
  - Execution of actions when conditions are true

Rule execution, basic flow

- **Facts Asserted**
  - ALL rules evaluated
  - For each condition that evaluates to true, add rule to agenda
  - Rules in agenda have their actions executed in order of priority
  - Facts are retracted



Rule Execution - Chaining

Order
Total = 110
Shipping = ?
Level = Gold

Order
Total = 110
Shipping = 0
Level = Gold

Order
Total = 99
Shipping = 0
Level = Gold

Order
Total = 99
Shipping = 10
Level = Gold

Rule A

If Order.Total > 100

Then Order.Shipping = 0

Rule B

If Order.Total <= 100

Then Order.Shipping = 10

Rule C

If Customer.Level = Gold

Then Order.Total =

Order.Total * .90

Managing Policies and Vocabularies

- **All business rule policies are versioned**
  - Once a business rule policy is published, it is immutable
  - To make a change, a new version of the policy must be created
  - Policy versions can be copied and pasted in the editor
  - Published versions can be deleted – but remove with care

Business Rule Policy Lifecycle

- Saved – persisted to the rule store, but not ready for use
- Published – ready to be used by others to develop against
- Deployed – the version is available to be executed

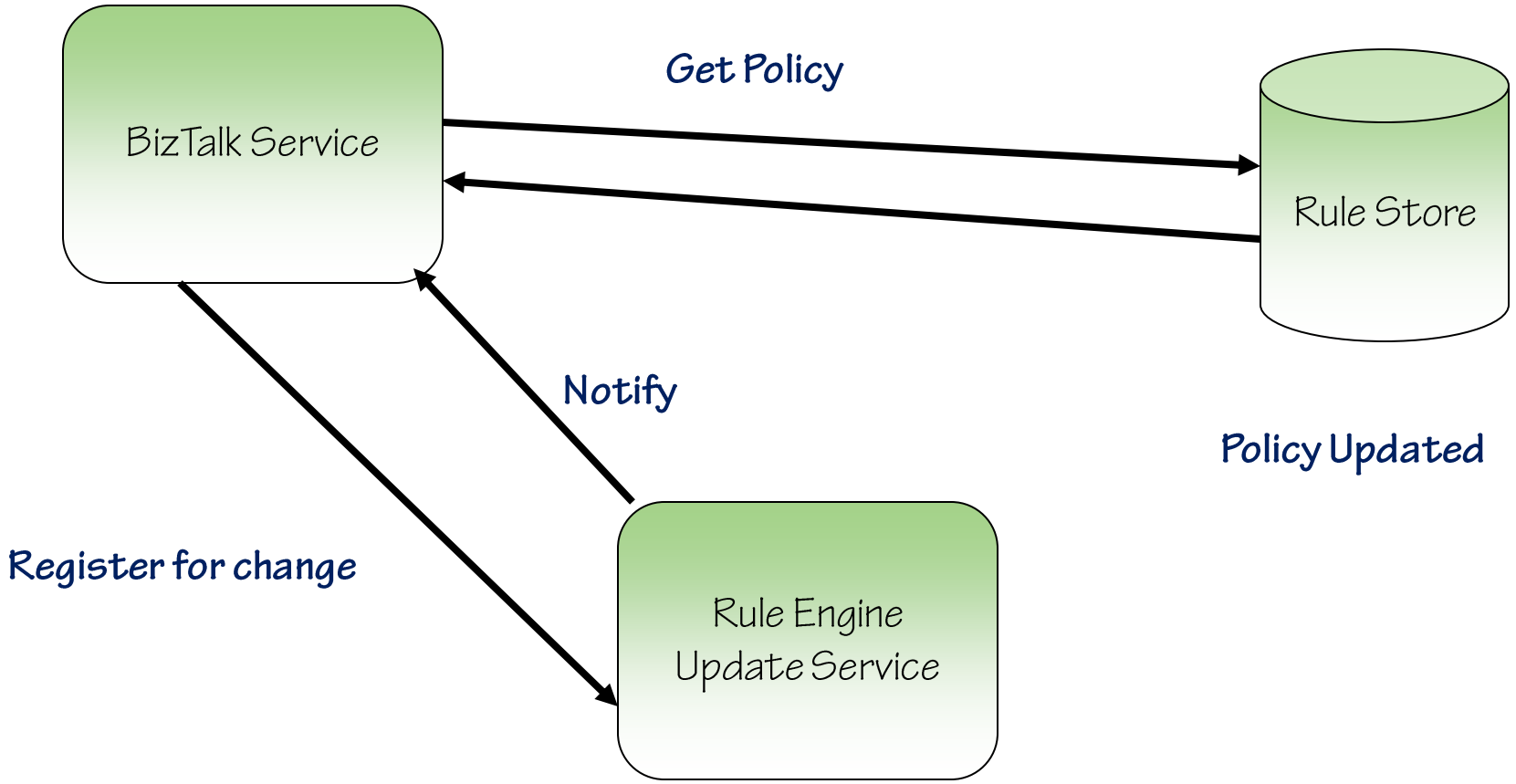
Managing vocabularies

- **All vocabularies are versioned**
- **Vocabularies have a simplified lifecycle**
  - Saved
  - Published
- **A vocabulary must be published before being used in a policy**
  - Published vocabulary versions are dependencies of policies
  - Cannot remove until all references to the version are removed
  - If you want the policy to use the new vocabulary, update the policy

Orchestration and Rule Engine

- **Orchestrations can call rules using the *Call Rules* shape**
  - Configure the shape with policy and parameters
  - Must enclose call rules shape in atomic scope
  - Optionally, call rules engine directly from expression shape
  - Messages can be used for XML facts
- **BTS uses RuleEngineUpdateService to ensure current version**

Rule engine update service



Programming business rules

- **You can program against Microsoft.RuleEngine.dll directly**
  - Create an instance of the Policy class
  - Optionally specify a version number
  - Call Execute passing in your facts

```
using (Policy p = new Policy("OrderDiscount", 1, 0))
{
    //create/initialize facts
    Northwind.BLL.ShoppingCart cart =
        new Northwind.BLL.ShoppingCart();

    //execute policy
    p.Execute(new object[] { cart });

    //check facts for results
    int result = cart.Total;
}
```

Programming business rules

- **Facts that are not .NET types need to be wrapped**
  - For XML documents, use the TypedXMLDocument class
  - For SQL database connections use the DataConnection class
  - For .NET DataSets, use TypedDataTable and/or TypedDataRow

```
//create/initialize facts
XmlDocument doc = new XmlDocument();
doc.Load("order.xml");
TypedXmlDocument typedDoc = new
    TypedXmlDocument("PS.ABTS.OrderProcessing.Schemas.Order",
    doc);

//execute policy
p.Execute(new object[] { typedDoc });

//check facts for results
doc.Save("updated_order.xml");
```

Programming business rules

- **Policies can be created through code**
  - Created on the fly for execution
  - Created in a custom tool and persisted to a rule store
  - Rules can be directly loaded from a *RuleStore* for execution
  - Generally not needed for most BizTalk scenarios
  - Provides support for building *custom editors*
  - Lightweight and flexible enough to be used without BizTalk

Summary

- Business rules provide flexibility
- Policies are made up of rules
- Rules use conditions and actions
- Facts are the targets of business rules
- Facts must be asserted for rules to execute
- Rules can be used in orchestration, or any .NET code

References

- **Risk Scoring with BizTalk Server 2004 and the Business Rules Framework**
  - http://msdn.microsoft.com/library/default.asp?url=/library/en-us/BTS_2004WP/html/1e2e50f7-6609-4eb2-a9a1-3a951700f840.asp