Smart Defaulting

High Level Architecture Specification

Draft 3

April 9, 2009

Table of Contents

6 Issues / Unknowns / Risks

1 Overview

1.1 Description

This document presents a proposed design to implement ISDA ticket defaults for Swaption's Little and Big tickets. Current design proposal will be further enriched as other defaults dimensions are investigated and analyzed.

1.2 Document References

ID	Document Title	Document Author	Description

Table 1. Document References

1.3 Assumptions / External Dependencies

ID	Short Name	Description
1.		

Table 2. Assumptions / External Dependencies

2 Requirements

2.1 The design should be based on rules based architecture

- The defaults generated by rules should be mapped to an object model that is representative of dataset to avoid further transformation on server side
- The design should be extensible
    - Ability to add new defaults
    - Ability to add additional input criteria
- Rules should be easily maintainable. In the future we may provide GUI to maintain the rules
- There should be an way to turn on/off specific rules
- Implementation should be capable of identifying duplicate rules and, possibly, mutually exclusive rules.

2.2 Static defaults

The defaults that are static in nature should be table driven or configuration driven and for those defaults that are dynamic in nature implement the rules in a high-level language (i.e. java) or rule engine (i.e. Drools)

2.3 Grammar

Simplified grammar is to be used to express rules. For complex rules allow the use of Java

2.4 Optimization

Rule execution should be context aware and not fire all rules regardless of the event.

2.5 API

APIs should be exposed for the application to invoke rules by passing required input criteria

3 Architecture

3.1 Architecture Overview

Due to the number of rules that define the relationship between Swaptions's Little Ticket, Big Ticket, XML, Ticker and other reference entity parameters - "rule based" architecture will be provisioned. The following are key benefits of using rule engine:

a) Declarative Programming – rules are much easier to read then code.

b) Logic and Data Separation - breaking the coupling of data and logic

c) Speed and Scalability - rules are especially efficient when datasets do not change entirely (as the rule engine can remember past matches)

d) Centralization - by using rules, we are creating a repository of knowledge - ideally rules are so readable, they also serve as the documentation.

e) Explanation facility - rule systems effectively provide an "explanation facility" by being able to log the "decisions" made by the rule engine (and why the decisions were made).

f) Understandable rules (readable by domain experts) - rules can look very close to natural language. They lend themselves to logic that is understandable to domain experts who may be non technical (as all the program plumbing is in the usual code, hidden away).

3.2 Key Technical Components

3.2.1 Stubs

Reference data look up will be stubbed out for the phase 1 of the POC and will be represented by a relevant abstraction.

3.2.2 Proxies

Since data model is not currently well defined or finalized and to keep business rule independent of data model considerations, a level of abstraction is needed to insulate rule implementation from implementation specific volatilities (i.e. xml dataset structure).

BigTicket proxies – responsible for relaying rule instructions to the datasets.

```
public class Ticket
{
  private Dataset dataset;

  public void setCreditEventBankruptcy(String applicability)
  {
      String path = "CreditEvent.NotStandard.Bankruptcy";
```

```
        Field field = DatasetHelper.findField(dataset, path);

        return field.setValue(applicability);
    }
}
```

3.2.3 Rules

Phase 1 of the POC will use Drools and open source rules engine currently JBoss Rule 3.0.1 (http://labs.jboss.com/portal/jbossrules) Other rules engines can be evaluated. Overall architecture, where possible, will not be vendor dependent.

There 3 ways of defining a rule:

3.2.3.1 Drools Script and Java

Example of using drools scripting -

```
rule "North America"
        when
                InputParameters (domicile == "North America") // matching
                bigTicket : BigTicket()  // reference to big ticket
        then
                bigTicket.setCreditEventBankruptcy("False");
end
```

3.2.3.2 Natural language (domain specific grammar)

Here we are able to define our own grammar. This creates a layer of separation between rule authoring (and rule authors) and the domain objects that the engine operates on.

```
rule "North America"

        when
                Domicile is "North America"
        then
                Bankruptcy Credit Event is "Not Appoicable"
end
```

3.2.3.3 Decision Table

Decision tables are a way of representing conditional logic, and are well suited to "business" level rules. For example

Field Name	Domicile	Issuer Type	Currency	Seniority	DocClause	Value
Bankruptcy	North America	Sovereign				Not Applicable
Bankruptcy		Corporate	EUR			Applicable
Bankruptcy			GBP			Applicable

This translates into 3 individual rules

a) If Domicile is North America and Issuer is Sovereign the Bankruptcy is not applicable
b) Else, if Issuer is Corporate and Currency is EUR then Bankruptcy is applicable
c) Else, If Currency is GBP then Bankruptcy is not applicable

3.3 Component Diagram

Note that currently rule engine does not participate in XML-to-Dataset conversion as shown on the diagram – this is a long term goal.

3.4 Rule Design

The following key concerns are evaluated:

a) Some rules are more important then others.
b) Most specifically matching rule is to be executed first
c) Only one change per field is allowed. Changing the field more then once signifies gaps in rule definition.

3.4.1 Salience

Salience, or Priority, is a way to weighing rules. The fundamental principle followed here is as follows: more specific rules are more important then less specific rules. For example, consider 2 rules

Rule 1: If Domicile is North America and Issuer is Sovereign then Bankruptcy Credit Event is not applicable

Rule 2: If Currency is EUR then Bankruptcy Credit Event is applicable

Since the conditional ordering is - Domicile then Issuer then Currency – Rule1 wins as it is more specific then Rule2.

We turn this into a concrete numeric model:

Conditions (executed left to right)					**Weight**
Domicile	**Issuer Type**	**Currency**	**Seniority**	**DocClause**	
10000	***01000***	***00100***	***00010***	***00001***	***11111***
North America	Corporate	USD			11100
Latin America		USD			10100
		GBP			00100
		USD			00100

Empty cell represents “wildcard” or “any” condition – i.e. “any domicile” or “any issuer type”

3.4.2 Rule Groups

Consider these 2 rules:

Rule 1: If Domicile is North America and Issuer is Sovereign and Currency is USD the Bankruptcy Credit Event is not applicable

Rule 2: If Currency is USD then Bankruptcy Credit Event is applicable

In the example above Rule 1 and 4 overlap – whereas Rule 4 is a more general version of Rule 1. If such a scenario was to occur, we would expect Rule 1 to file and Rule 4 to be ignored. This is accomplished by grouping rules into “activation-groups” a drools term for rule groupings.

3.4.2.1 Flow

Rule invocation is a multi-step process

1) Rule definitions are loaded
2) Rule definitions are compiled into a Rule Base – a singleton shared across the application
3) Working memory is instantiated – this is the execution space of this operating instance
4) Rule arguments are asserted – rule engine performs object analysis and prepares them for rule execution.
5) Rules are fired – rule engine evaluated the rules conditions against objects located in the working memory and fires rule that match up.

3.4.2.2 Optimizations

1) Working memory should not be instantiated per call. While it is possible to run in a stateless (service oriented) mode, the performance won’t be as optimal as a Stateful model. Working memory should be reused where possible. In our case we will keep a single instance of the working memory per Session.
2) Rule arguments should be kept dynamic where possible. Object model should follow Java Beans property change notification. This will enable rule engine to be aware of object changes without having to reassert the object into the working memory which is quite expensive.

3.4.2.3 Instrumentation

Stat4J will capture metrics across individual rule invocations to enable further analysis of rule performance we’ll be able to collect the following:

1) Rule average run time
2) Rule maximum run time
3) Rule invocation count
4) Rule average memory usage
5) Rule maximum memory usage

4 Integration

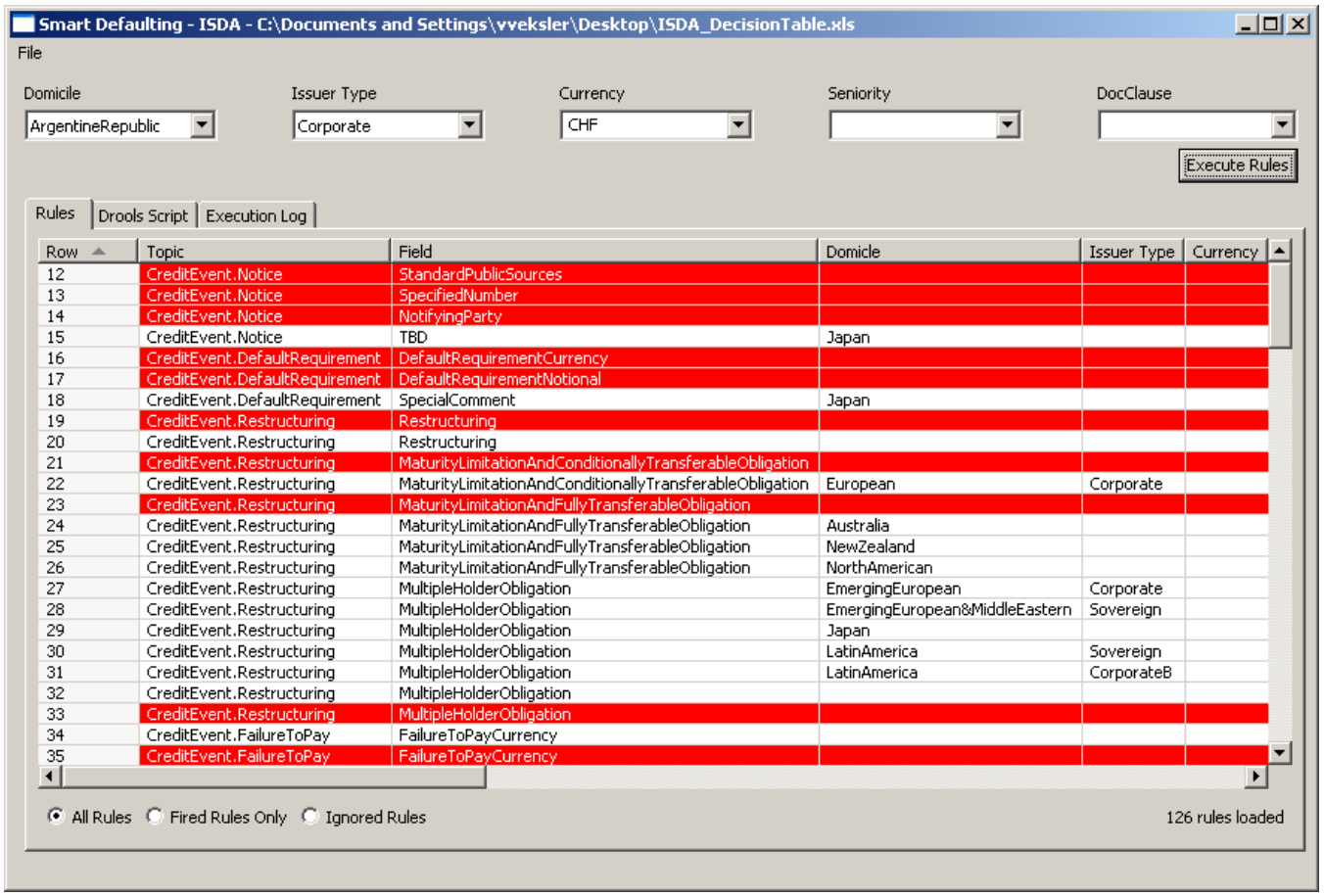
4.1 Overview

Rule engine execution is an “enrichment” process. It can be called from anywhere in the system whenever there is a requirement to evaluate ticket defaults. For example – dataset initialization and relevant field changes (i.e. ticket change)

4.2 Testing

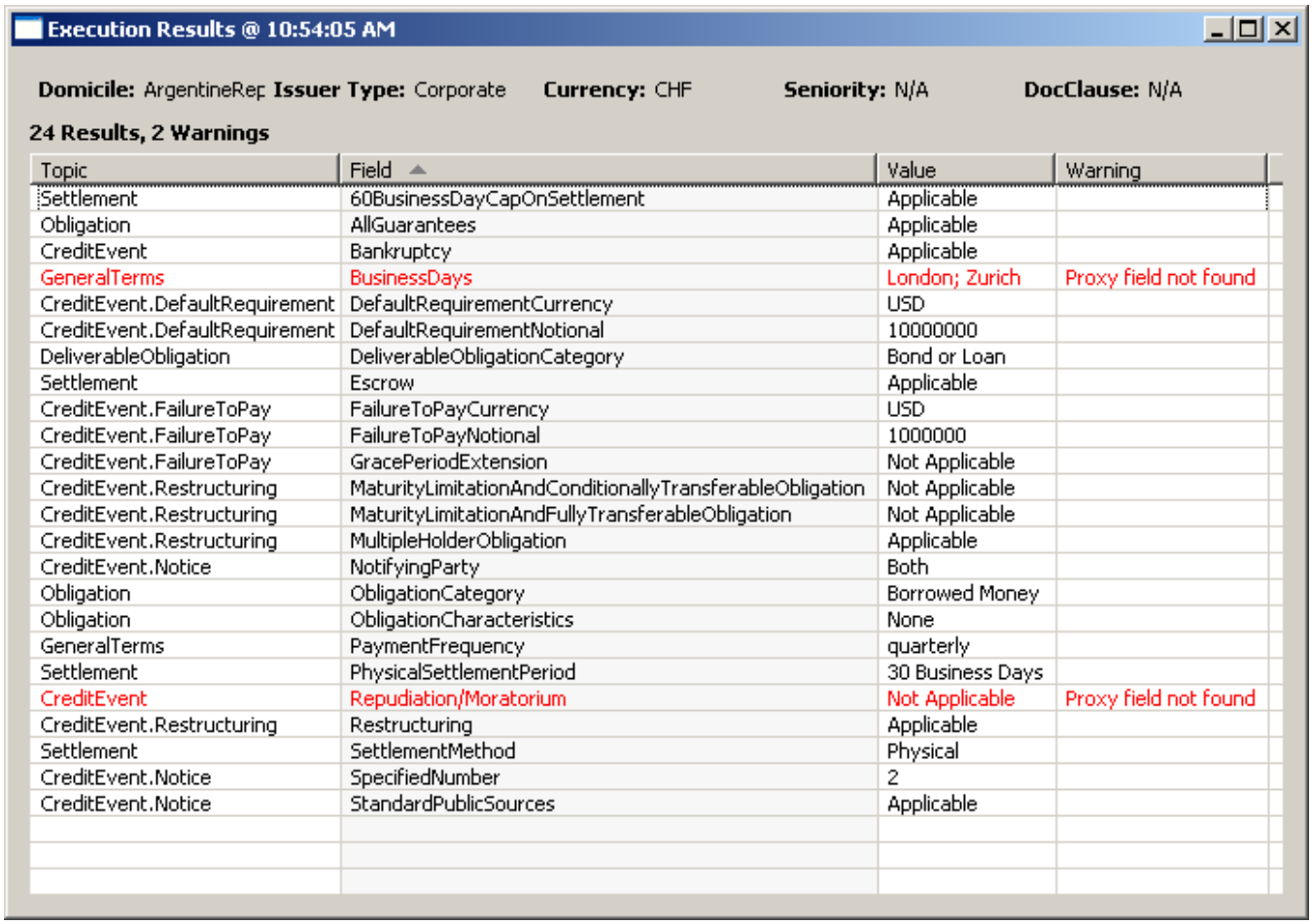
Test GUI has been created for decision table based Smart Defaulting

It can be downloaded via Java WebStart here http://nyfitw1009716/SmartDefaulting/ISDA/



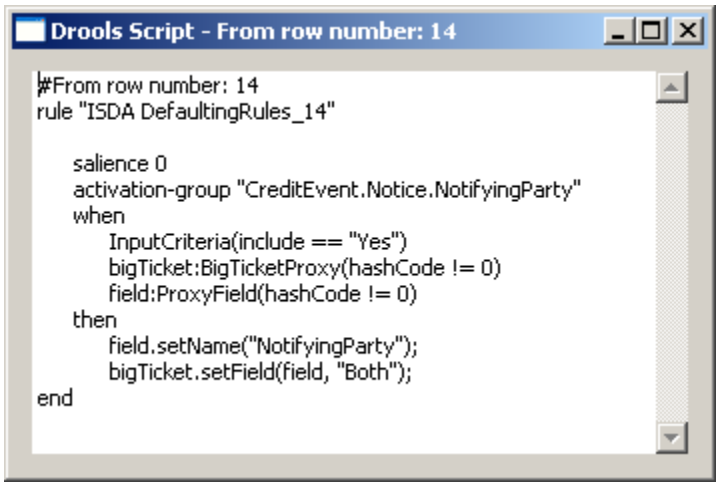
Row	Topic	Field	Domicile	Issuer Type	Currency
12	CreditEvent.Notice	StandardPublicSources			
13	CreditEvent.Notice	SpecifiedNumber			
14	CreditEvent.Notice	NotifyingParty			
15	CreditEvent.Notice	TBD	Japan		
16	CreditEvent.DefaultRequirement	DefaultRequirementCurrency			
17	CreditEvent.DefaultRequirement	DefaultRequirementNotional			
18	CreditEvent.DefaultRequirement	SpecialComment	Japan		
19	CreditEvent.Restructuring	Restructuring			
20	CreditEvent.Restructuring	Restructuring			
21	CreditEvent.Restructuring	MaturityLimitationAndConditionallyTransferableObligation			
22	CreditEvent.Restructuring	MaturityLimitationAndConditionallyTransferableObligation	European	Corporate	
23	CreditEvent.Restructuring	MaturityLimitationAndFullyTransferableObligation			
24	CreditEvent.Restructuring	MaturityLimitationAndFullyTransferableObligation	Australia		
25	CreditEvent.Restructuring	MaturityLimitationAndFullyTransferableObligation	NewZealand		
26	CreditEvent.Restructuring	MaturityLimitationAndFullyTransferableObligation	NorthAmerican		
27	CreditEvent.Restructuring	MultipleHolderObligation	EmergingEuropean	Corporate	
28	CreditEvent.Restructuring	MultipleHolderObligation	EmergingEuropean&MiddleEastern	Sovereign	
29	CreditEvent.Restructuring	MultipleHolderObligation	Japan		
30	CreditEvent.Restructuring	MultipleHolderObligation	LatinAmerica	Sovereign	
31	CreditEvent.Restructuring	MultipleHolderObligation	LatinAmerica	CorporateB	
32	CreditEvent.Restructuring	MultipleHolderObligation			
33	CreditEvent.Restructuring	MultipleHolderObligation			
34	CreditEvent.FailureToPay	FailureToPayCurrency			
35	CreditEvent.FailureToPay	FailureToPayCurrency			

Smart Defaulting rule execution result screen



Topic	Field	Value	Warning
Settlement	60BusinessDayCapOnSettlement	Applicable	
Obligation	AllGuarantees	Applicable	
CreditEvent	Bankruptcy	Applicable	
GeneralTerms	BusinessDays	London; Zurich	Proxy field not found
CreditEvent.DefaultRequirement	DefaultRequirementCurrency	USD	
CreditEvent.DefaultRequirement	DefaultRequirementNotional	10000000	
DeliverableObligation	DeliverableObligationCategory	Bond or Loan	
Settlement	Escrow	Applicable	
CreditEvent.FailureToPay	FailureToPayCurrency	USD	
CreditEvent.FailureToPay	FailureToPayNotional	1000000	
CreditEvent.FailureToPay	GracePeriodExtension	Not Applicable	
CreditEvent.Restructuring	MaturityLimitationAndConditionallyTransferableObligation	Not Applicable	
CreditEvent.Restructuring	MaturityLimitationAndFullyTransferableObligation	Not Applicable	
CreditEvent.Restructuring	MultipleHolderObligation	Applicable	
CreditEvent.Notice	NotifyingParty	Both	
Obligation	ObligationCategory	Borrowed Money	
Obligation	ObligationCharacteristics	None	
GeneralTerms	PaymentFrequency	quarterly	
Settlement	PhysicalSettlementPeriod	30 Business Days	
CreditEvent	Repudiation/Moratorium	Not Applicable	Proxy field not found
CreditEvent.Restructuring	Restructuring	Applicable	
Settlement	SettlementMethod	Physical	
CreditEvent.Notice	SpecifiedNumber	2	
CreditEvent.Notice	StandardPublicSources	Applicable	

Rule execution results summary and warnings



Double clicking on a row in the summary view allows user to drill down to drools script that is behind a particular rule.

5 Examples

5.1 Configuration

5.1.1 Java

```
Collection<RuleConfiguration> list = new ArrayList<RuleConfiguration>();

list.add(new RuleConfiguration("rules/examples/Defaulting.drl"));

RuleBase ruleBase = RuleBaseLoader.load(list);
```

5.1.2 Spring Framework

```
<bean id="ruleBaseLoader" class="com.rules.RuleBaseLoader"/>

<bean id="ruleBase" class="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
  <property name="targetObject" ref="ruleBaseLoader"/>
    <property name="targetMethod" value="load"/>
    <property name="arguments">
      <list>
        <bean class="com..rules.RuleConfiguration">
          <constructor-arg><value>rules/ISDA_DecisionTable.xls</value></constructor-arg>
          <constructor-arg>
              <util:constant static-field="org.drools.decisiontable.InputType.XLS"/>
           </constructor-arg>
        </bean>
      </list>
    </property>
</bean>

<bean id="ruleRunner" class="com..rules.RulesRunner">
   <property name="ruleBase">
     <ref bean="ruleBase"/>
    </property>
</bean>
```

5.2 Usage

```
WorkingMemory wm = ruleBase.newWorkingMemory();
wm.assertObject(argument1);
wm.assertObject(argument2);

wm.fireAllRules();
```