Using IBM Operational Decision Manager DVS simulation features for risk scoring analysis use cases

Zhuo (Joe) Fu
Jerome Boyer

April 02, 2014

This article presents how to extend IBM® Operational Decision Manager Decision Validation Service features to produce a KPI-based simulation on large datasets so that risk scoring analysis can be conducted by business users. The design and implementation considerations presented in this article are mainly based on IBM ODM V8.5, but they can apply to older product versions such as WebSphere® ILOG JRules V7.5 and V8.

Introduction

A common issue related to rule-based risk scoring analysis is the ability to perform intensive simulation testing around specific scoring aggregation based on a huge data volume. IBM Operational Decision Management (ODM) Decision Validation Service (DVS) is used to perform scenario-based simulation testing. Test scenarios are defined in Microsoft® Excel worksheets; with each scenario defining data and producing expected results. Excel is a well-known tool for business users, but for large datasets, it may become difficult to manage and scale. Also, with analytic requirements around aggregating KPIs, we need to access huge historical production datasets. This leads us to extend the IBM ODM DVS capability to get such data from external data sources.

Rule testing is a crucial and intensive task within rule development, due to the nature of business rules working together and the value of having the business users maintaining the rules. IT and production environment managers want to ensure that new deployed rulesets reach a level of quality that does not impact the production servers' service level of agreement. During the early phase of the ruleset development, we recommend to adopt a test driven development approach to implement the rules. Each rule is created after writing test plans. The tests are executed in the Rule Designer during the development (item #2 in Figure 1). Test cases are defined with business and rule analysts and business rule owner when performing the rule discovery and analysis tasks (item #1 in Figure 1), and developed by the rule developer.
Once the rule business object model is stable, the vocabulary is well-defined, and the ODM Business Object Model layer is complete, it is possible to transfer test case and rule creation to the business and rule analyst (item #3 in Figure 1) by using the IBM ODM Decision Center and Decision Validation Services components. Empowering rule authors with the ability to change the rules also means such rules should be tested in a non-regression test environment. Non-regression test suites can also be defined with DVS.

Figure 1. SDLC life cycle with business rule and testing practices

Testing validation is integrated in the new ODM V8.5 decision governance framework as a way to enforce control on ruleset quality, particularly once in production (see the IBM ODM V8.5 Information Center). It is important to note that even if Figure 1 illustrates a development life cycle; rule authoring, testing, and simulation apply continuously once the rulesets are deployed on the production server. The activities conducted in the "staging" phase of this high level plan apply to production.

For functional testing, the test team deploys the ruleset in a decision server and applies testing at the decision service level. The recommended approach is to consider a ruleset as a unit of functionality that gets exposed to the rest of the application. Therefore, it needs to be tested using the same functional testing strategy as for any other component of the application. The automated functional test tool can be used for that purpose.

The fourth high level activity shown in Figure 1 may happen when business requirements enforce the use of simulation to assess rule execution against the Key Performance Indicators (KPIs). The developer has to adapt ODM DVS for that.

We assume the reader is familiar with IBM ODM core functions as we will focus on describing ODM DVS.

Rule testing and simulation by using ODM DVS

ODM DVS is used to create testing and simulation solutions for rule developers, rule authors, and testing teams. These solutions are used to validate the correctness and effectiveness of rulesets.
The ability to test scenarios gives rule authors the assurance that the changes they made have the desired results. Figure 2 illustrates the components as part of DVS:

- **Rule Designer**: Enables you to test and simulate and to unit test rulesets during the development phase.
- **Decision Center**: Allows you to author rules and to execute test suites and simulations at the end of development and once in UAT and production.
- **Rule Execution Server**: Allows to execute rules and to trace data and fired rules in the Decision Warehouse.

**Figure 2. Rule testing and simulation architecture**

Figure 3 illustrates how DVS works inside the Decision Center by connecting to the Test Decision Server that has the Rule Execution Server and a Scenario Service Provider (SSP) installed.

**Figure 3. DVS and SSP relationship**

The SSP is an IBM ODM pre-defined web application component that provides services to execute testing scenarios and rules and to return scenario execution reports. The rulesets are deployed on a connected Rule Execution Server (RES). This simulation RES should not be the production server, but a dedicated platform using a simple WebSphere® Application Server standalone profile. Therefore, you need to deactivate any SSP.ear files on your production server. As part of
By default, rule authors define and populate test scenarios in a Microsoft Excel spreadsheet. Spreadsheet templates are created within either the Decision Center or the Rule Designer from the IBM ODM business object model (BOM) definitions. The templates define the BOM elements to be populated with input test values and expected output values. Not all BOM members are needed for the test scenario. The row in the spreadsheet represents a test case. Therefore, a spreadsheet includes multiple test cases and represents the test suite that can be part of an automatic non-regression testing environment. The test data is manually entered by the rule author. Once scenarios are defined, the Excel file is sent to the SSP, which loads the test data, creates Java™ instance objects, sends the objects to the rule engine, and gets the results back to build reports. The report compares the actual output data with the expected results defined in each test scenario.

The rule developers must publish the different rule projects to the Decision Center with the DVS Excel test scenarios. Those artifacts are stored in the rule repository or in a public folder for which all rule authors have access.

After reviewing the test suite report, the rule author can modify the rules and change the scenario input data to reflect an alternative use case and re-execute the test suite. This development process can be repeated until the user is satisfied with the results. Each subsequent execution is persisted in the repository, allowing comparisons with the previous execution results. In the Decision Center, test suites are versioned like other rule artifacts. Persisted in the repository, they are shared between rule developers, rule authors, and QA test engineers.

Figure 4 shows the results of running the default testing scenario format (Excel) in the Decision Center. The Decision Warehouse is not described in this article, but it is an out-of-the-box database table that persists the list of rules executed for a given data set.

**Figure 4. DVS logical components**

**Business rule simulation and custom scenario provider**

A business rule simulation is used to perform “what if” analysis against realistic data and the updated business ruleset. The main goal is to compare the rule execution results against KPIs so
that two different ruleset versions can be compared. Business users can gain valuable insight into
the potential impact of their changes. Good simulations mean executing a ruleset against a large
enough volume of realistic data, usually historical production data to represent the most interesting
business scenarios.

You can evaluate multiple KPIs for each simulation. Incremental computation of KPIs (as each
scenario is processed) provides high scalability and performance for typically large simulation
runs. Rule authors may analyze aggregate decisions and create reports from the results. They
can do this in Excel or other business intelligence tools (not provided by IBM ODM). In many
of the business applications using external rule sets, most realistic simulations require a DVS
custom scenario provider, a Java component, which we will describe in the next section. A custom
scenario provider is likely to use data from production databases to represent realistic values in
volume and semantic. This will be explored as we further design this aspect in later sections of this
article.

IBM ODM comes with a built-in integration point to allow new developed scenario providers to
be integrated into the different components like the SSP, Rule Designer, and Decision Center. To
illustrate this customization, let's see a typical risk scoring use case.

**Use case of risk scoring analysis**

This section describes a typical use case for using rule simulation testing to conduct risk scoring
analysis. It starts to list typical requirements for risk scoring analysis, then highlights the required
support based on DVS features and extensions, and finally describes a design around the DVS
extension within IBM ODM.

**Risk scoring analysis requirements**

Risk scoring is done within a business rule ruleset. Transactional data are sent to the rule engine,
and the rules compute score according to the data received. Multiple levels of scoring may apply
and many of the data combination could apply when determining a risk scoring. Listing 1 illustrates
a typical business rule to calculate a risk score by taking into account the assessment mode, the
type of transaction, the potential risk category, and any other business meaning attribute such as a
country code. The risk category may have been a derived or inferred attribute, computed by other
business rules earlier in the rule flow.

**Listing 1. Sample of risk scoring business rule**

```python
if
    the assessment mode of 'transaction' is "Mode-1"
and the entity type of 'transaction' is "Type-1"
and the risk category of 'transaction' is "Category-1"
and the country code of the primary party of 'transaction' is "US"
then
    assign the risk score of the primary party of 'transaction' to 85;
```

Business users understand the risk score distribution, and they use a simulation to assess how
the rule changes the impact of scoring. The common requirements for risk scoring simulation are
summarized as follows:
• Support risk scoring analysis based on a pre-defined score bucket, such as score ranges from 0-10, 11-20, and so on.
• Report percentage metrics based on risk score result such as score based eligibility rate.
• Support certain hit ratio for a list of pre-defined pattern of risk criteria.
• Process transaction with a timestamp within a time window, or filtered out on specific data attribute (like query on data element).
• Support different KPI definitions: score bucket size, score threshold, score value, and any combination of pre-defined KPIs (for example, calculate the percentage against each grouping or combined groups).
• Present certain aggregated simulation KPI results in a user friendly tabular format.
• Present simulation result organized by a hierarchical structure based on criteria like a risk bucket and specific data categories (for example, geography, account type, customer type).

DVS extensions

In order to support the business and functional requirements related to a simulation testing for the risk scoring analysis, the default DVS features have to be extended. These extensions include the following:

• Integrate with the dedicated data source to use the historical production data. This is done by adding new data providers used to populate test data from the historical database.
• Define new DVS Decision Center scenario renderers, which allow testers to specify data selection criteria including, for example, transaction date ranges. This requires a user interface customization for the Decision Center DVS page. However, we recommend that the simulation tests are conducted in one unique execution to generate all the necessary output because the test execution is a CPU-intensive process and should be performed only once. The different reports, based on the concerned KPI and taking into account filtering parameters, are generated in the same execution too. The filtering criteria and reporting configuration are defined at the beginning of simulation testing.
• Implement all the identified KPIs, which are used to generate the simulation KPI reports.
• Extend standard DVS reporting capability to generate the required KPI reports.

DVS extension design

When extending DVS, there are two main components to adapt: Decision Server and Decision Center. For the Decision Center, the user interface helps the rule author, who is performing the simulation to define the simulation configuration, select the KPI, and tailor the reports. The second extension is to adapt the scenario service provider (SSP), deployed to the ODM Decision Server, so that the test data is coming from external data sources and not Excel to compute KPIs and to return and persist simulation reports. Figure 5 illustrates the proposed extensions to ODM DVS, where the customization related components are highlighted.
Using IBM Operational Decision Manager DVS simulation features for risk scoring analysis use cases

Figure 5. DVS extension component view

The SSP extension contains two key features:

- A custom scenario data provider to load historical production transactional data, which is used as input for the ruleset execution.
- A KPI calculation function responsible to compute the various custom KPIs based on the rule execution results.

As shown in Figure 6, the whole simulation solution includes a "data feeder" component responsible to replicate real-time production transactions to the simulation input database, a dedicated "data provider" to load simulation input data into SSP, and a separate set of table to persist simulation results so that the external business intelligence product can perform other reporting. The simulation results are also sent back to the Decision Center, as the trigger for the simulation is done from the Decision Center. The simulation results are stored in the repository.

Figure 6. A sample rules simulation system architecture
The scenario data provider needs to connect to the related data source, applies certain query criteria based on the user input, and retrieves the data to build the test scenarios. The code can be based on traditional JDBC, or can use the Java Persistence API to speed up development. Figure 7 illustrates the key classes designed for the DVS simulation extension.

**Figure 7. The main DVS extension class diagram**

![Diagram of the main DVS extension class diagram]

Figure 8 illustrates the messages between the main objects involved in these DVS customizations:

- A simulation scenario is defined by specifying certain data input. The execution is triggered by the DVS user interface inside the Decision Center. Within the simulation scenario definition, data input (such as criteria to the filter transactional data) is specified via the customized DVS resource renderer class (for example, `SimulationScenarioSuiteResourcesRenderer`).
- When the simulation execution is triggered, the ruleset and scenario data input are captured by the `SimulationTestInputData` class and made available to the ODM SSP.
- SSP is responsible to load the transaction data based on the passed data input, via the custom scenario data provider (`SimulationDataScenarioProvider`, an implementation of the `IlrScenarioProvider` ODM API). The provider uses DAO (`XOMDataDAO` for this case) to load data from the historical transactions database and to build DVS test scenarios for each transaction data.
- SSP uses the RES API for the rule execution for each DVS test scenario, and returns back with the result so that the KPI results are computed. This is achieved via an implementation class of `IlrKPI` (`CombinedDVSSimulationKPI` in our case).
- After all the test scenarios are completed, the KPI results are formalized in a pre-defined format (via an implementation class of `IlrKPIResult`) and returned back to the ODM Decision Center. It is also possible to persist results in a set of dedicated database tables to be used for future decision analysis.
- To view the simulation result inside the ODM Decision Center, a customized DVS KPI result renderer, such as `XMLKPIResultRenderer`, is defined.
IlrScenarioProvider has four methods to override:

- Initialize the provider; for example, getting access to the external resource and getting the execution context injected.
- Get the number of the scenario.
- Get a specific scenario given an integer index.
- Close the resources.

DVS requires an implementation of the IlrKPIResult interface to represent the KPI result inside the Decision Center. A typical KPI result is represented as a single value, such as a percentage rate or a statistics value. However, for most risk scoring KPIs, multiple dimensional set of values are obtained. For example, the result of a risk score distribution KPI typically consists of a two-dimensional set of values with one dimension for the criteria grouping (such as assessment mode, entity type and risk categories, and so on), and another dimension for the score bucket (as illustrated in Figure 9). For this case, a worksheet document is best used to present the data, not only due to its tabular format, but also for the possible integration and consumption by other component and the Business Intelligence tool. Note that several KPIs are required to represent the required scoring distribution. The simulation report consists most likely of several worksheets, each of which represents one KPI.
Based on the above-described combined KPI results, an XML-based format is defined so that you can easily format them into an Excel document or a tabular HTML report later on. The following XML example shown in Listing 2 shows how to capture a sample Score Distribution KPI result.

Listing 2. XML-based DVS simulation of the KPI result

```xml
<?xml version="1.0" encoding="UTF-8"?>
<RiskScoreDistributionKPI>
  <TotalTransactionCount>543000</TotalTransactionCount>
  <ByAssessmentMode>
    <AssessmentMode>mode_1</AssessmentMode>
    <ByRiskCategory>
      <RiskCategory>category_1</RiskCategory>
      <ByScoreBucket>
        <ScoreBucket>0-9</ScoreBucket>
        <Count>13</Count>
      </ByScoreBucket>
      <ScoreBucket>10-19</ScoreBucket>
      <Count>6</Count>
      <ScoreBucket>20-29</ScoreBucket>
      <Count>33</Count>
      <ScoreBucket>30-39</ScoreBucket>
      <Count>16</Count>
      ........
      <ScoreBucket>moreThan200</ScoreBucket>
      <Count>56</Count>
    </ByScoreBucket>
    <RiskCategory>category_2</RiskCategory>
    <ByScoreBucket>
      <ScoreBucket>0-9</ScoreBucket>
      <Count>13</Count>
    </ByScoreBucket>
    <RiskCategory>category_1</RiskCategory>
    <ByScoreBucket>
      <ScoreBucket>0-9</ScoreBucket>
      <Count>13</Count>
    </ByScoreBucket>
    <RiskCategory>category_2</RiskCategory>
  </ByAssessmentMode>
</RiskScoreDistributionKPI>
```
KPI calculation and implementation considerations

To support the risk scoring KPI requirements, we choose one class called `CombinedDVSSimulationKPI` to capture all the KPI computations. This approach helps us to present all the KPI results in a single view, to share the DVS scenario results access between the different ways of computing the KPIs, and finally, due to the KPI uniqueness, improve performance for persisting KPI or send back to the Decision Center console. A typical implementation approach uses a dedicated class (`CompositeValue`) to store, for each KPI, one row of values in a tabular format, and interface definitions to represent the data record associated with each KPI, that will be updated when a test scenario got executed. This interface can be implemented differently based on the specific KPI. For the two dimensions risk score distribution KPI, a hierarchical structured is used. The implementation needs also to take into account how the KPI record gets updated when each test scenario is executed. Finally an XML-based result is generated from all the KPI records within `CombinedDVSSimulationKPI`, returned back to the ODM Decision Center, and persisted into a dedicated database.

To illustrate the above KPI computation, we can start from the business rule sample as shown in Listing 1 and describe its process as follows:

- If the test scenario triggers the sample rule, the risk score obtained is 85.
- This updates the KPI record identified using the hierarchical structure: Mode-1 > Type-1 > Category-1/(80-89).
- The distribution KPI is updated by adding 1 to the value associated with the above identified location.
- When all the test scenarios are completed, a full score distribution will be obtained as shown in the table in Figure 9.

Listing 3 illustrates a Java implementation of the above-mentioned hierarchical record structure and some of its core functions.

**Listing 3. KPI calculation implementation**

```java
public class HierarchicalKPIRecordImpl implements SimulationKPIRecordIF {
    /** the list of top-level grouping nodes */
    private List<KPIGroupingNode> topNodes;
    /** the KPI report name */
    private String reportName;
    /** to represent a list of score ranges to be used for the tabular form of KPI data. */
    private List<String> scoreRanges;
    /** default constructor. */
```
public HierarchicalKPIRecordImpl(String reportName, List<String> scoreRanges) {
    topNodes = new ArrayList<KPIGroupingNode>();
    this.reportName = reportName;
    this.scoreRanges = scoreRanges;
}

/**
 * @return List<KPIGroupingNode> the list of root KPIGroupingNode.
 */
public List<KPIGroupingNode> getTopNodes() {
    return topNodes;
}

......

/**
 * this function is to traverse the grouping tree, to find the location leave if existing. If not, it
 * will create such a leave, and add to the grouping tree.
 */
public KPIGroupingNode getCreateKPIGroupingLeave(HashMap<String, String> values) {
    KPIGroupingNode theTopNode = null;
    // starting from the top-level node:
    String topNodeName = GroupingNodeConfig.getInstance().getOrderedNodeDisplayName(reportName,0);
    String topNodeValue = values.get(topNodeName);
    if (!topNodes.isEmpty())   {
        int size = topNodes.size();
        for (int i=0; i<size; i++) {
            if (topNodeValue.equals(topNodes.get(i).getValue()))  // find the existing already.  {
                theTopNode = topNodes.get(i);
                i = size;
            }
        }
    }
    if (theTopNode == null) {
        theTopNode = new KPIGroupingNode(topNodeName, topNodeValue);
        addTopNode(theTopNode);
    }
    // then traverse the tree:
    int groupingSize = GroupingNodeConfig.getInstance().getOrderedGroupingNodeSize(reportName);
    KPIGroupingNode leaveNode = theTopNode;
    for (int k = 1; k < groupingSize; k++) {
        String groupingNodeName = .getInstance().getOrderedNodeName(reportName,k);
        String groupingNodeDisplayName = GroupingNodeConfig.getInstance().getOrderedNodeDisplayName(reportName,k);
        String groupingNodeValue = values.get(groupingNodeName);
        leaveNode = getCreateChildGroupingNode(leaveNode, groupingNodeDisplayName, groupingNodeValue);
    }
    if (leaveNode.getSummaryCount() == null) // To initialize if not yet.   {
        CompositeValue valueObj = new CompositeValue();
        valueObj.initializeValues(scoreRanges);
        leaveNode.setSummaryCount(valueObj);
    }
    return leaveNode;
}

/**
 * This function is to create or retrieve the grouping node, based on the specified value.
 */
protected KPIGroupingNode getCreateChildGroupingNode(KPIGroupingNode parent, String childName, String childValue) {
    KPIGroupingNode result = null;
    result = parent.findChild(childName, childValue);
    if (result == null) {
        result = new KPIGroupingNode(childName, childValue);
        parent.addChild(result);
    }
    return result;
User interface extension

In order for the business rule author to specify what kind of data to use for the simulation and to select the different KPI report options, a specific user interface (UI) is needed. Particularly, we developed a UI to define:

- The historical transactions start date-time and end date-time
- The entity type to filter the transaction data and any other data filtering needed
- The Simulation KPI report options, including:
  - Score buckets, for example, every 10 or every 100
  - Score thresholds, for example, user request threshold of 200. (Above this value, the score bucket will be called ">200").

The default DVS scenarios user interface within the Decision Center enterprise console (Figure 10) will be changed into something like the one shown in Figure 11.

Figure 10. Default DVS scenarios user interface

![Figure 10](image1)

Figure 11. Proposed DVS scenarios customization user interface

![Figure 11](image2)

This user interface renderer is an implementation of the interface, IlrScenarioSuiteResourcesRenderer, in the ODM Decision Center API. The main functions to be implemented are encodeAsEditor() and encodeAsViewer(). The first function is to produce an editor as shown in Listing 4, and the second function is to produce a viewer while the scenario is in review mode inside the Decision Center. The implementation is based on dynamic HTML, the technology used for the UI renderer within the Decision Center. The code shown in Listing 4 illustrates how a DateTime input field can be implemented inside the renderer.

Listing 4. Scenario Suite Renderer UI implementation

```java
public void encodeAsEditor(IlrScenarioFormatDescriptor formatDescriptor,
   Map<String, byte[]> resources, FacesContext context,
   UIComponent component) throws IOException
{
```
For a more detailed description, see the topic on the DVS simulation scenario suite UI renderer.

**DVS report extension**

The DVS execution result is also presented in the Decision Center. To support this custom user interface, another ODM API, you need to implement the IlrScenarioSuiteKPIRenderer interface. The user interface layout may look like Figure 12, where a new button is added to support a downloading feature for the Excel based KPI report.
To implement this report extension feature, a simple servlet is added to the Decision Center web application and performs the report downloading function. The main implementation is responsible to convert the KPI simulation results in the format of XML (obtained from SSP back to the Decision Center) into an Excel document.

To create an Excel converter, you can use the 

- Apache foundation POI project. The main conversion function takes the in-memory KPI report as an XML document and transforms it into worksheets. Horrible SpreadSheet Format (HSSF) is the POI API that represents an Excel '97(-2007) file format. HSSF provides ways to create, modify, read, and write XLS spreadsheets. Due to the direct tabular representations based on the XML structure, the Excel conversion is quite straightforward, with the tasks of configuring various HSSF cell styles (for example, color and font) and creating HSSF cells based on the XML nodes and leaves by using the configured cell styles. Listing 5 illustrates a simple Java code to write value rows into an Excel sheet.

**Listing 5. Excel converter Java implementation**

```java
// start from a composite value.
Element valueEle = leave.getChild(DVSSimulationConstants.KPI_XML_VALUE_TAG);
String value = valueEle.getText();

HSSFRow valueRow = sheet.getRow(rowIndex);
HSSFCell rccell = valueRow.createCell(startColIndex, HSSFCell.CELL_TYPE_STRING);
    rccell.setCellValue(new HSSFRichTextString(value));

Element countsEle = leave.getChild(DVSSimulationConstants.KPI_XML_COMPOSITE_VALUE_TAG);
```
// write the count values one by one.
List<Element> childs = countsEle.getChildren(DVSSimulationConstants.KPI_XML_VALUE_PAIR_TAG);
for (Element childEle : childs) {
    Element countEle = childEle.getChild(DVSSimulationConstants.KPI_XML_VALUE_TAG);
    String count = countEle.getText();
    HSSFCell tccell = valueRow.createCell(startColIndex+2);
    tccell.setCellStyle(cellcs);
    double totalCountV = Double.parseDouble(count);
    tccell.setCellValue(totalCountV);
    startColIndex++;
}
......

DVS customization project within the ODM Rule Designer

To facilitate DVS customization, ODM provides a DVS customization template project that is integrated as an Eclipse plug-in. Figure 13 illustrates a sample DVS customization project that has been created for the DVS simulation implementation, with all the above-mentioned Java classes implemented in the "src" sub-folder.

Figure 13. DVS customization project structure

The project includes a configuration descriptor defining the key settings, such as the target deployment environment, the concerned rule project, the format of the data provider to be used, the KPI classes, and so on (see Figure 14).
Figure 14. DVS customization configuration settings

Figure 15 illustrates the settings that are defined in the "Risk Scoring Simulation" format, which contains the following key settings:

- **Scenario Provider**: Sets the name of the scenario provider (by default, Excel 2003). The section also displays the main scenario provider class, the renderer class for the Decision Center, and the name of the plug-in project for the new launch configuration.

- **KPI**: Sets the name of the Key Performance Indicator (KPI), the name of the KPI main class, and the name of the renderer class for the Decision Center.

- **Precision**: Provides the data precision to use in the tests.

- **Expected Execution Details**: Selects the expected execution details options that you want to make available for selection in the Generate Excel Scenario File Template wizard.

Figure 15. DVS format settings for a DVS simulation
Conclusion

This article presented a DVS extension approach to address a simulation-based risk scoring impact analysis. This approach created a score distribution report of simulation KPI results in a tabular format. Such a report gives the business user, especially the business rule author, insight on how to perform impact analysis based on a huge historical transaction dataset. All the related design and implementation considerations around the ODM DVS extension are discussed when similar DVS-based simulations are required to be designed and implemented. The DVS extension design and implementation described in this article is not only used for risk scoring impact analysis, they are also generic enough to address other use cases that extend the DVS simulation capability.
Related topics

- Decision validation activities
- Using the decision governance framework
- Custom scenario providers and parallel execution DVS sample
- API for DVS

© Copyright IBM Corporation 2014
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)