Embed reference data in IBM ODM decision service implementations

Pierre Berlandier (pberland@us.ibm.com)
Senior Technical Staff Member, IBM ODM Services
IBM

Matt Voss (mattvoss@us.ibm.com)
IBM ODM Application Architect, Smarter Process Center of Competency
IBM

09 December 2015

Make reference data available to business rules at run time by embedding the data with the artifacts that are deployed to the Rule Execution Server in IBM Operational Decision Manager (ODM). Learn example approaches that are useful when the Rule Execution Server cannot access an external data service, for example in an IBM ODM on Cloud environment.

Introduction

Running a complex decision service that is implemented with IBM ODM often requires access to some enterprise reference data to evaluate and apply the business rules. This data is usually defined as lookup tables, or collections of key-value pairs that represent a relationship between two or more business concepts. Making these tables available to the rule engine can sometimes present a challenge to the application architecture, the change management process, or both.

The goal of this tutorial is to provide guidance and recommendations to make reference data available to business rules at run time, depending on size, frequency of change, and other considerations for the characteristics of the data. The tutorial focuses on two simple approaches where the reference data is embedded with the artifacts that are deployed to the Rule Execution Server: the ruleset and the managed execution object model (XOM). While other approaches might be more efficient or scalable, these two approaches have a significant importance for IBM ODM on Cloud, where all the data available to the rules comes either from the input payload provided during the decision service invocation or from the Rule Execution Server resources.

Context data and reference data

Most business rules-based decision service implementations are based on a simple, stateless model where a collection of input data elements is processed with a sequence of if-then rules to produce output data that represents the expected business decision. The required input data falls into the following two categories:

- Data that is specific to an instance of the decision service invocation. Examples include the characteristics of the loan applicant, the property for which you underwrite a loan, the details of the medical claim that you need to adjudicate, or the purchase order that you want to validate. For this tutorial, the data is called the *transaction data* or the *context data* for a decision.
- Data that represents reference points that are used in defining and applying the business policy. This data typically represents external regulatory or compliance rules or internal characteristics on how the company conducts its business. For example, it might represent the maximum interest rate that is legally allowed by each State, or the list of valid medical procedure codes, or the type of items that cannot be shipped to a postal code. In this rest of this tutorial, this type of data is called the *reference data* for a decision.

From a high-level point of view, a key function of business rules is to match values that are issued from the *context data* (either directly or derived through business rule computations) against values that are selected from the *reference data*, to eventually produce the wanted decision.

Consider the example of a rule from a purchase order validation decision service, which has the role of excluding any item that is restricted for shipping in the state of residence of the buyer. Such a rule can be expressed in IBM ODM, as shown in Listing 1:

Listing 1. Example exclude restricted items action rule that uses reference data

```
definitions
  set item to an item in the line items of 'the PO';
if
  item is restricted in the state of residence of 'the buyer'
then
  add item to the exclusions of 'the PO'
      with reason "cannot ship item to the buyer's state";
```

The reference data table that is needed to support running this rule can define the list of items that are restricted in particular states. For example, the table might look like Figure 1:

Figure 1. Restricted items reference data table

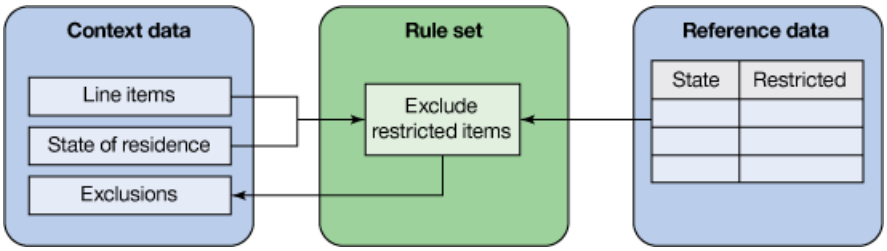
State	Restricted SKUs
AL	10909003, 10909051
AK	
AZ	
AR	10801150
CA	10801121, 10801122, 10801124
CO	

The business rule that is shown in Listing 1 uses both context and reference data elements. It implements part of the business policy by matching context data against reference data. Consider the following details in this example:

- The ordered items and the exclusions of the purchase order are part of the context data.
- The state of residence of the buyer is also part of the context data.
- The list of restricted items by state is part of the reference data.

Figure 2 illustrates the different type of elements that are involved in the example.

Figure 2. Context and reference data that support ruleset execution



Data characteristics

Context and reference data usually have different characteristics. Context data represents the details of an entity from the business domain. As such, it is usually modeled as a large and complex object hierarchy that includes several levels and multi-valued attributes. The more complex the decision, the larger and more detailed the object hierarchy, to reflect as many of the possible facets of the context that might be relevant to the business decision.

By contrast, reference data is usually a simple mapping table (potentially with multiple columns). It does not have the structural complexity of context data. However, the tables are often large, with hundreds or thousands of entries. For reference data, a more complex decision translates in tables that are more granular (for example, with more rows, with criteria at the level of the city or postal code, instead of the state). Or, a more complex decision includes more tables to support finer and more diverse decision criteria in the rules.

Consider the following other notable differences:

- Context data scope is the entire decision, and reference data scope is typically limited to a small number of specific rules or rule tasks within the decision.
- In most cases, all the data points from the context data are used to compute the decision, and only a few from the reference data set might apply.
- The context data is different from one decision service invocation to the next, and the reference data changes are infrequent (usually less frequent than changes to the rules themselves).

As mapping tables, reference data describes some relationship with the context data. However, it does not provide any prescription on how the relationship should be used. In most cases, the data makes a natural fit for the left side (the condition portion) of the rules in the ruleset.

The challenge is to find the best way to incorporate the reference data into the ruleset to make it operational at run time for the decision at hand, and possibly for other decisions that are part of the same application. The next section describes some factors that you should consider to help you select the right approach.

The code examples in this tutorial were developed with IBM ODM V 8.7.1.

Selecting an approach

The approach that is eventually selected to make reference data available to the decision service depends on multiple factors that derive from both functional and nonfunctional requirements of your solution. Carefully evaluate these factors, because there is no approach that fits all situations.

Local or remote invocation

In general, using a local invocation of a decision service (for example, using a rule session obtained from `IlrJ2SESessionFactory` or `IlrPOJOSessionFactory`) is a default, go-to solution, providing the most flexibility to access reference data. There is no consideration of network latency for exchanging large amounts of data between the invoked application and the decision service. Security considerations for accessing a database or other data services might be simplified.

By contrast, when you use remote invocation – for example, a custom web service hosted transparent decision service (HTDS) or monitored transparent decision service (MTDS) – carefully evaluate the amount of data that is exchanged over the wire. You might find it difficult to provide a remote Rule Execution Server, potentially deployed in the cloud, with the necessary access to the data sources. For example, for IBM ODM on Cloud, access to external services or data sources are not allowed for security reasons.

As a side note, keep in mind the following tip if the request payload is large for remote invocations. Enterprise JavaBeans (EJBs), if you use a rule session obtained from an `IlrEJB3SessionFactory`, might be a more efficient option than web services. If you use heterogeneous languages or environments, consider using a message-driven rule bean in your solution.

Sharing

The same reference data can be used in different ways by different applications and can have many types of different consumers. Whether or not the data is exclusive to the rules often drives how it is incorporated into the ruleset. In the case of multiple consumers, the data always must be available to all consumers and kept synchronized for all consumers.

Governance

The expected change frequency, the ownership, the software development lifecycle, and other governance considerations strongly influence where you should maintain the reference data.
The decision governance process needs to include the necessary steps to ensure that each new ruleset that is deployed is using the proper version of the reference data. This requirement is especially important when the source for the reference data is not managed through IBM ODM. Regardless of its nature, each repository must be hosted, managed, version-controlled, and secured.

Volume

The volume of the reference data that needs to be managed might affect the ruleset size, the performance, and the business user experience of authoring the rules in the Decision Center. Large amounts of reference data usually preclude embedding the data in rules.

Testing

The reference data must be made available for testing. It is easy to understand that if you use the Decision Center Business Console for testing, the reference data store must be accessible from Decision Center. Likewise, the reference data must be accessible for rule developers who are testing in Rule Designer. Offline testing requires that the data source is available in the local environment.

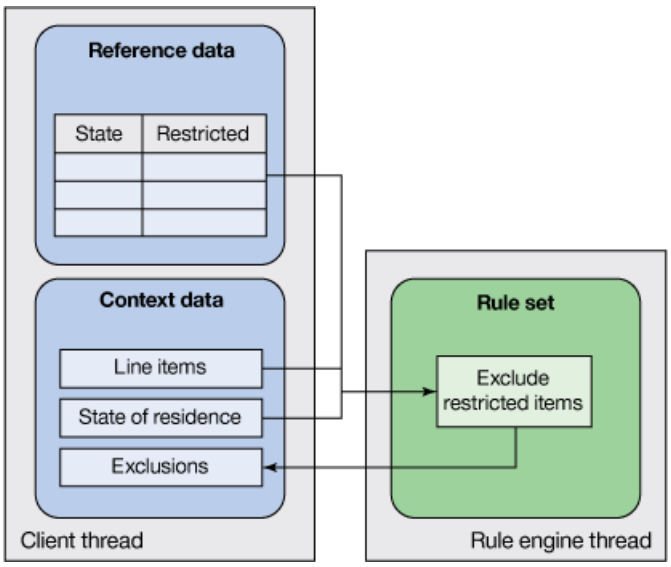
Enriching the request payload

Before discussing the details of solutions that embed the reference data into deployable Rule Execution Server artifacts, consider a common strategy to make reference data available to the rules before invoking them on the Rule Execution Server. You make the data available by augmenting the request payload. Either filter and enrich the context data, or provide a set of handles to the reference data tables that can be used in the rules.

In general, the reference data is sourced from database tables, and it is cached by frameworks such as the Java Caching System from Apache Commons (commons.apache.org/proper/commons-jcs) and Ehcache, an open source, standards-based cache (www.ehcache.org).

As shown in Figure 3, it is the responsibility of the decision service client thread to gather the data that is available after the ruleset is running. The client thread in Figure 3 is usually part of a custom web service that exposes the decision service to clients in a service-oriented architecture and encapsulates the invocation of the IBM ODM service after the enrichment step.

Figure 3. Reference data through request enrichment

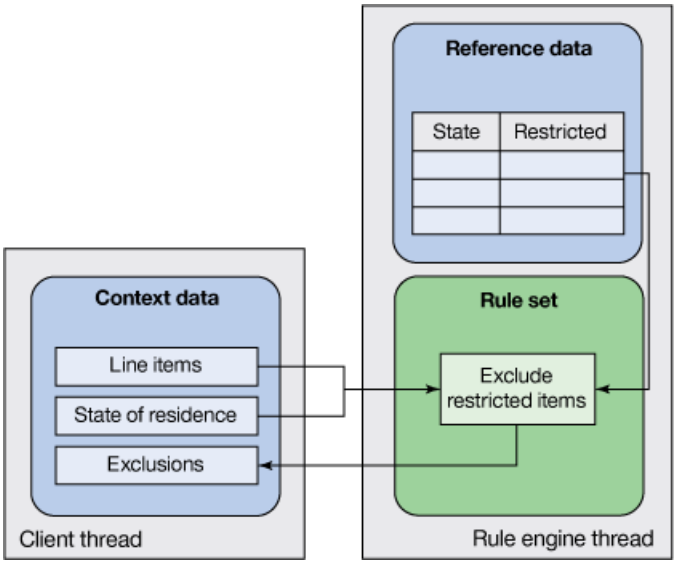


Because enrichment directly affects the size of the payload, this strategy is sensitive to whether the decision service invocation is local or remote. Also, since the whole set of reference data needed is gathered for the decision service before its invocation, more data than is actually needed might be retrieved, affecting performance. Finally, avoid including business knowledge when filtering data for enrichment. For more details about this approach, see Accessing external data in a rules application.

Capturing reference data in the XOM

In many situations, it is preferable to gather the reference data during the ruleset execution. Similar to the request payload enrichment case that is described in the previous section, the data itself is maintained outside of the rules and pulled in when the rules are running. However, the rule enging thread retrieves the data. The organization of components looks like Figure 4:

Figure 4. Capturing reference data in the XOM



You can store the reference data in either an external data source (for example, a database) or as a file in the XOM. For details of the database approach, see Accessing external data in a rules application.

The alternative, which consists of capturing reference data in the XOM, essentially relies on bundling a file resource with the managed XOM. The file is read at execution time and evaluated. The approach used to evaluate the contents depends on the nature of the data. The data can be evaluated either within the XOM layer, completely transparent to the business user, or converted into objects in the business object model (BOM), which can then be used in the rule authoring task. The following implementation example demonstrates how the file is read and the results are used in rules downstream.

Implementation example

Capturing reference data in the XOM essentially relies on bundling a file as a resource in an XOM archive that is published and managed by the IBM ODM Rule Execution Server. The resource file

is loaded and processed at run time by a data provider Java class. The class is made part of the BOM so that it can be used to write rules.

The following example demonstrates an example implementation of a data provider class and how it can be used in rules downstream. The reference data from Figure 1 can be stored in a flat file where each line represents the restrictions for one state. The first token of the line indicates the state, and the rest of the lines are a coma-separated list of stock keeping units (SKUs). The content looks like the example in Listing 2.

Listing 2. Example reference data in a flat file

```
AL 10909003,10909051
AZ
AR 10801150
CA 10801121,10801122,10801124
CO
…
```

The `ReferenceDataProvider` class in Listing 3 reads the file and translates the contents to the appropriate restriction objects. It follows the singleton pattern design, which restricts the instantiation of a class to one object, and it includes a public synchronized `getInstance` method that calls a private constructor.

In the first invocation, the constructor parses the data set, and it stores the parsed data set in a hash map of `ShippingStateRestriction` objects, indexed by the corresponding state.

Listing 3. ReferenceDataProvider class definition

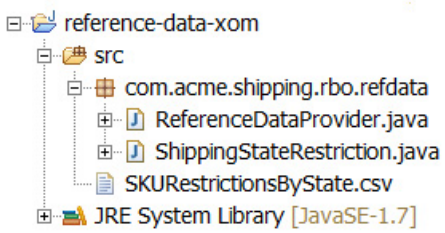
```
public class ReferenceDataProvider {

    private static ReferenceDataProvider instance = null;
    private Map<StateType, ShippingStateRestriction> restrictions = null;

    private static final String LINE_SPLIT_BY = " ";
    private static final String SKUS_SPLIT_BY = ",";

    private ReferenceDataProvider() throws IOException {
        if (restrictions == null) {
            parseDataSet();
        }
    }

    public static synchronized ReferenceDataProvider getInstance()
            throws FileNotFoundException, IOException {
        if (instance == null) {
            instance = new ReferenceDataProvider();
        }
        returninstance;
    }

    public ShippingStateRestriction getRestriction(StateType state)
    {
        return restrictions.get(state);
    }

    private void parseDataSet() throws IOException {
        String dataFilename = "SKURestrictionsByState.csv";
        InputStream inputStream =
            getClass().getClassLoader().getResourceAsStream(dataFilename);
```

```
        BufferedReader br =
            new BufferedReader(new InputStreamReader(inputStream));
        restrictions = new HashMap<StateType, ShippingStateRestriction>();

        String line;
        while ((line = br.readLine()) != null) {
            String[] entry = line.split(LINE_SPLIT_BY);
            StateType state = StateType.valueOf(entry[0]);
            List<String> skus = Arrays.asList(entry[1].split(SKUS_SPLIT_BY));

            ShippingStateRestriction stateRestriction =
                new ShippingStateRestriction();
            stateRestriction.setState(state);
            stateRestriction.setSkus(skus);
            restrictions.put(state, stateRestriction);
        }

        br.close();
    }
}
```

In this simple example, the entire contents of the file is loaded into the map, pushing the data filtering to the BOM or the rules. If the content is large, you can take a more discriminate approach for what is loaded and retained in memory.

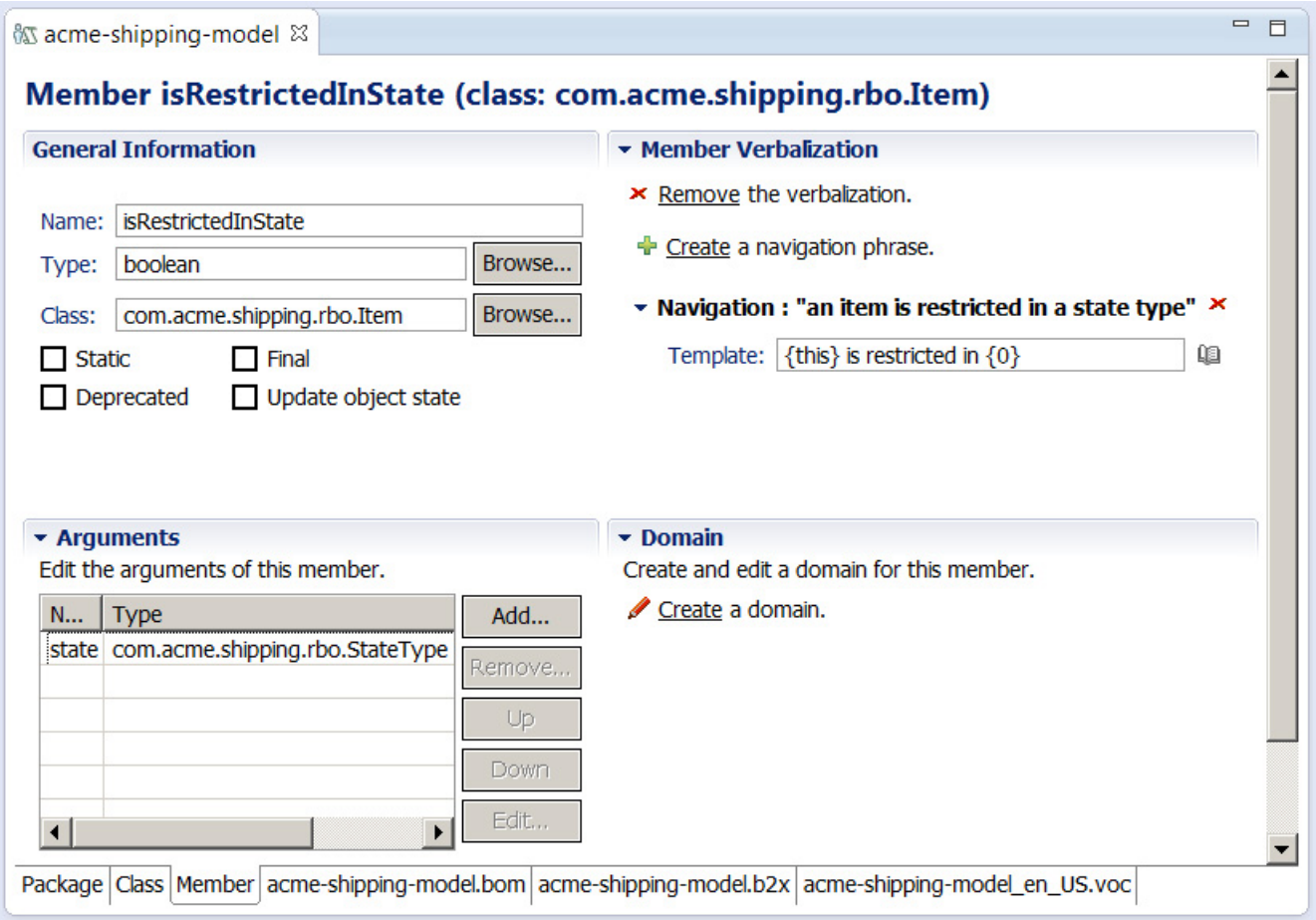
The reference data file itself is contained in the XOM project, and you can find it in the Java class-path. Figure 5 shows an example project structure with both the `SKURestrictionsByState.csv` resource file and the `ReferenceDataProvider` class.

Figure 5. Example XOM project organization



In the BOM, you can use various patterns to expose the relevant entities to the rules. For the example rule in Listing 1, you can use a verbalized virtual BOM method for the `Item` class to evaluate whether the item is restricted. The definition of this method is illustrated in Figure 6.

Figure 6. Example BOM method definition to access reference data



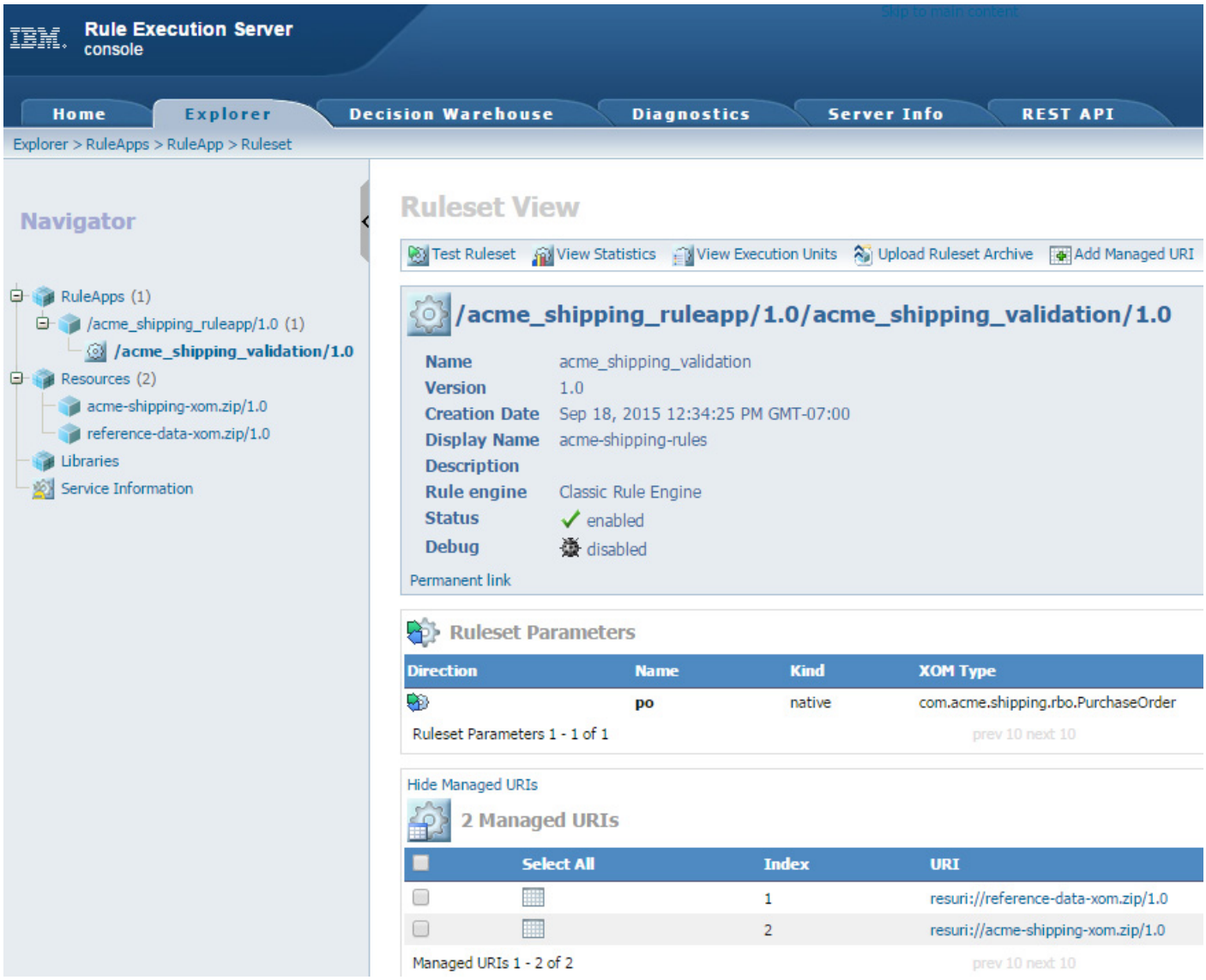
The BOM-to-XOM code snippet in Listing 4 shows the method body that uses the `ReferenceDataProvider` class. It gets the appropriate state restriction from the map and returns `true` or `false` as appropriate.

Listing 4. BOM-to-XOM code to access reference data

```
ReferenceDataProvider provider = ReferenceDataProvider.getInstance();
ShippingStateRestriction restriction = provider.getRestriction(state);
return (restriction.getSkus().contains(this.getSku()));
```

The managed XOM that captures the reference data should be independent from the core set of XOM classes that support the context data. Then, the reference data XOM archive can be deployed independently from the other managed XOM resources. Figure 7 shows the deployment of the purchase order validation decision service, which uses two separate resources: `acme-shipping-xom.zip` (which contains the core set of XOM classes) and `reference-data-xom.zip` (which encapsulates the required reference data).

Figure 7. Example managed XOM in the Rule Execution Server



Design considerations

Review the following considerations to decide whether to adopt the file resource solution as the repository for the reference data:

- *Impact of change:* Changing reference data involves changing to a deployable Java artifact. It doesn't affect the service implementation, which does not need to be redeployed. The updates remain confined to the artifacts managed by the Rule Execution Server component.
- *Reference data filtering logic:* When the reference data filtering rules are subject to change or are complex, loading the data in the ruleset provides some advantage over hard-coding the data filtering in the decision service. It allows the rules to evolve without redeployment of the service implementation, and it does not put business logic in the service-layer code. However, significant complexity in the filtering rules can be a candidate for a separate ruleset.
- *Change frequency:* Frequency of change to the reference data should be low, perhaps a few times a year. Because a change to the reference data requires an XOM redeployment,

which usually requires some approvals and code change, don't use this approach with highly dynamic sets of data. Also, the governance process should provide specific directions on how to implement change to the reference data.
- *Reference data size:* The typical size of reference data is a few megabytes. All or part of it can usually be loaded at run time without straining the Java virtual machine (JVM) heap memory. An Excel spreadsheet or a plain text file usually supports easy manipulation of large amounts of data.
- *Resource availability:* Sometimes, the hardware resources or the human resources to implement and manage a database for reference data are scarce, difficult to secure, or not available. In these types of environments or when the additional governance overhead is not palatable to the business user, the file-based approach provides a quick means for business users to maintain the data that is relevant to them, and change is kept in scope of the rules.
- *Data availability:* If data is deployed in the XOM, there are no dependencies on remote data sources. This approach overcomes some of the shortcomings of remote data access in the ruleset (such as exception management). It can be an alternative if the reference data size precludes implementation as rules in Decision Center.

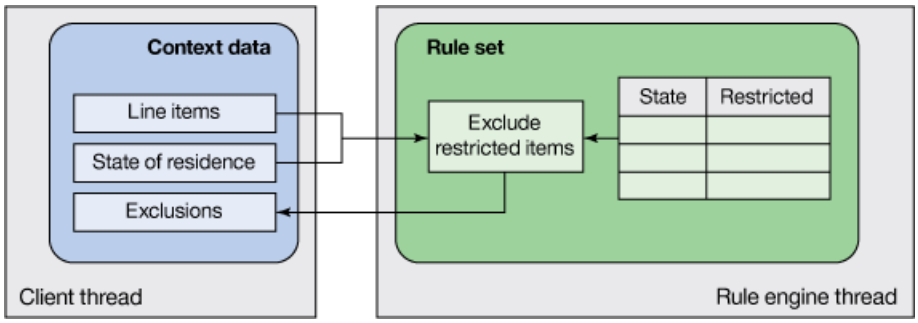
Capturing reference data in the rules

At first glance, decision tables in IBM ODM look like reference data tables. So, you might wonder if you could just leverage rules to capture their reference data, and deploy the data through the ruleset. You might choose this approach for the following reasons:

- Conserving a single repository, the Decision Center, to store and manage all the artifacts that are associated with the implementation of the business policies: Business users must learn only one user interface to manage their business decisions and can perform all their updates through that one interface.
- Involving and managing a single runtime artifact (the RuleApp)to deploy and run the business policies: This approach simplifies architecture and change management.
- Avoiding any synchronization issues between the rules and the reference data: Because the reference data is deployed with the ruleset, there is no risk that it is updated too early or too late with respect to the definition of the business rules.

If you use the exclude restricted items rule example in this tutorial, the components organization looks like Figure 8.

Figure 8. Capturing reference data in the ruleset



Implementation example

Because the reference data might be needed in several rule tasks in the ruleset (and possibly across multiple rulesets), the implementation of the decision table should not be tied to a specific BOM class. Instead associate it with generic ruleset variables to make it reusable. Consider the following possible implementation in the example in Figure 9:

Figure 9. Variables supporting the reference data decision table

Variable Set: variables

Name	Type	Verbalization	Initial Value
state	com.acme.StateType	state	
restrictedSkus	java.util.List	restricted SKUs	

To support the definition of the decision table example from Figure 1, this implementation example uses two ruleset variables, one called `state` and another called `restricted SKUs`, which is a collection of SKU string values.

Using these variables, you can build the decision table in Figure 10 to represent the reference data:

Figure 10. Decision table implementing reference data

	State	Restricted SKUs
1	AL	*{ "10909003", "10909051" }*
2	AK	
3	AZ	
4	AR	
5	CA	*{"10801150"}*
6	CO	*{"10801121", "10801122", "10801124"}*

Then you can encapsulate the table in a simple subflow that can be reused if needed. In order for the table to fire properly, the `state` variable must be initialized with the target state before you run the sub-flow. In this example, you initialize the state variable with the value of the state residence of the buyer, using the following statement:

```
set state to the state of residence of 'the buyer' ;
```

After the reference data table is run, you can leverage the outcome to collect the exclusions. The rule initially presented in Listing 1 then looks like the example in Listing 5:

Listing 5. Implementation example of capturing reference data in rules

```
definitions
  set item to an item in the line items of 'the PO';
if
  the SKU of item is one of 'restricted SKUs'
then
  add item to the exclusions of 'the PO';
      with reason "cannot ship item to the buyer's state";
```

Design considerations

Carefully weigh the following points when considering capturing reference data in rules:

- The change frequency for the reference data should be reasonable. Ideally, the frequency should be comparable to, or lower than that of the other business rules. Although deploying a rule change is a much lighter process than an application code change, it still is more involved than updating the content of a database.
- The size of the reference data should be reasonable. Although IBM ODM can easily accommodate rulesets involving tens of thousands of rules (even more efficiently with the Decision Engine in IBM ODM V8.5.1), you do not want the ruleset size and execution time to be dominated by reference data.
- Information that is represented as reference data should not directly and uniquely associated with elements from the input context data. For example, a black list that collects account numbers that are suspected of being fraudulent is not a good candidate for a decision table. Because it is directly dependent on context data (the client account number), the list will most likely go against the two previously mentioned points: It changes more frequently than the business policy, and its size eventually grows very large.
- The owners of the rules should also own the reference data. Otherwise, the reference data owners must be included in the IBM ODM decision governance process and learn how to author and maintain rules, or the rule owners must proactively gather and update the reference data tables.
- Ideally, the reference data that is captured by rules should only be relevant in the scope of the rulesets that use them. In particular, the reference data should not be replicated from another repository (such as an outside database or document), which might cause data inconsistencies, unless the rules are automatically generated from the external source.
- Conversely, it is important to realize that the reference data that is captured by rules is not be easily queried by other systems or applications as it is from a database. While the Decision Center API allows retrieving and accessing the definition of rules and decision table entities, it requires relatively complex custom code, and yields unacceptable performance for anything beyond a reporting activity.

In addition to straightforward management of the reference data rules through the Decision Center, the following two sections propose alternatives that are useful for large data or data that is pulled out automatically from a data source.

Rule Solutions for Office

The Rule Solutions for Office component of IBM ODM provides Microsoft Office add-ins to allow creating and editing business rules in Microsoft Office documents. From Decision Center, business users can publish rules from a rule project to Microsoft Office documents called RuleDocs, work on the documents offline through Microsoft Word and Microsoft Excel, and later update the Decision Center project by importing the RuleDocs back into IBM ODM. For projects already using Rule Solutions for Office to manage rules, you can also use it to manage reference data decision tables.

Consider the following advantages of Rule Solutions for Office:

- When the reference data is owned by a department that is not directly involved with the maintenance of the decision service, the RuleDocs provide a simple way to have the external department make adjustments to the data without investing in learning how to use the Decision Center user interface and its associated governance process.
- Managing large decision tables in an Excel spreadsheet can prove a more efficient user experience than than using Decision Center. You and your team might find it easier to navigate large tables using Excel instead of Decision Center, for scrolling and resizing of the columns and rows. If you are working with large tables, Rule Solutions for Office provides an easier way to view all of the data in the table at once, although it is not necessarily faster than completing the same operations in Decision Center.

Consider the following limitations of Rule Solutions for Office:

- Rule Solutions for Office for IBM ODM V 8.7 and earlier releases officially supports Microsoft Excel and Word Versions 2007 and 2010. Microsoft Office 2013 is not officially supported.
- Rule Solutions for Office is not supported by IBM ODM on Cloud.

Automated rules generation

Instead of being authored and managed manually, the rules representing reference data can be generated automatically from a data source just before the ruleset extraction. An automated rules generation process can use the Decision Center API as demonstrated in Sample: Data sources for decision tables. Rules generation can be performed on demand, or just-in-time, as the first step of a ruleset extraction script.

Most of the time, you find that the data used for rules generation comes from a database. However, when the application requirements call for data files instead, you can store the files as resources, part of the rule projects. This approach ensures that the reference data files are always synchronized with the rules and properly versioned in the Decision Center repository.

Listing 6 shows how to retrieve a resource element from a rule project in the Decision Center repository, given the file base name and extension, and assuming that the Decision Center session argument is properly connected to the desired rule project.

Listing 6. Getting a resource element from a rule project

```
IlrResource getTableResource(IlrSession session,
                             String basename,
                             String extension)
  throws IlrRoleRestrictedPermissionException,
                    IlrObjectNotFoundException
{
    EClass eclass = session.getBrmPackage().getResource();
    IlrSearchCriteria criteria = new IlrDefaultSearchCriteria(eclass);
    for (IlrElementDetails element : session.findElementDetails(criteria)) {
 IlrResource resource = (IlrResource) element;
 if (resource.getName().equals(basename) &&
           resource.getExtension().equals(extension)) {
    return resource;
 }
    }
    return null;
}
```

The `getBody` method of the resource object that is returned by the methods in the example in Listing 6 allows you to retrieve the content of the file (see the `IlrResource` class documentation). With the content of the reference data file, you can use he Decision Center API to create and populate the decision tables.

Performance impact

As expected, capturing reference data as rules affects both rule management and rule execution time.

Authoring and maintenance

Very large decision tables (for example, several thousand rows) take significant time to load in the Rule Designer or Decision Center. For example, opening a decision table with 10,000 rows to edit in Decision Center on an example application server might take between 5 and 10 minutes, ruling out interactive editing of the artifact. Opening larger tables also can cause out-of-memory exceptions in either the Rule Designer, the Decision Center, or Rule Solutions for Office.

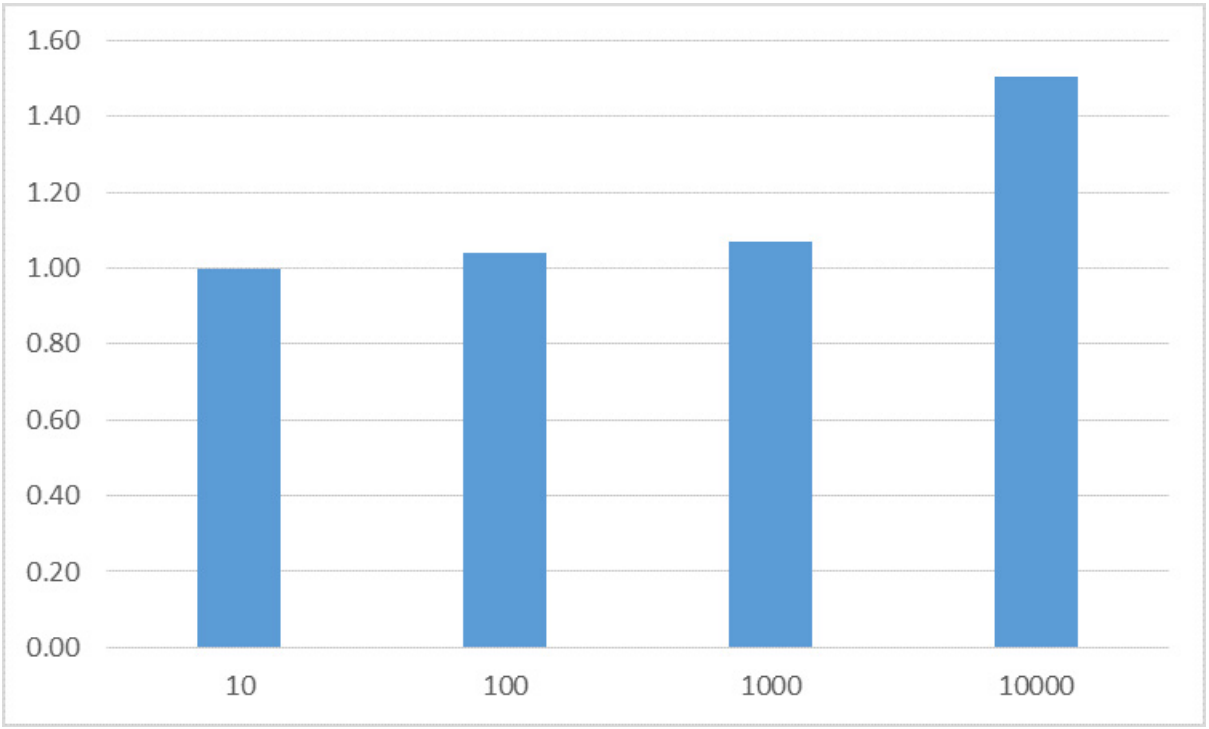
A published good practice for decision tables authoring and management is to keep them under 500 rows (see Limits on number of rows in a decision table in the References section). For tables that have a few multiples of the 500-row recommended limit, you can break up the original tables into smaller, more manageable chucks as a simple work-around.

Execution

In general, if you use the Decision Engine, the performance of decision tables scales well as the number rows in the table increase. Consider the results of a test using a decision table with a structure similar to the one shown on Figure 11 (essentially one condition column and one action column), using different number of rows, and looking-up a large number of different values across the table.

The execution time used as reference in this tutorial is based on an example table with 10 rows. Figure 11 shows the execution time for other tables (with respectively 100, 1000 and 10000 rows), expressed relatively to the execution time of 10 rows.

Figure 11. Comparison of execution time of reference tables of different sizes



The graph shows that there is no significant difference in looking up a value in tables with 10, 100 or 1000 rows. Looking up a value in a 10,000-row table is only 1.5 times more costly than in a 10-row table.

As a final reference point for execution performance, consider that using a Java hash table to implement the reference data table is about 400 times faster than using a decision table.

Hybrid approach

An alternative option to manage reference data as Rule Execution Server deployable artifacts is a hybrid of the two options in the previous sections. It consists of using the rule repository to author and manage the reference data as decision tables, and then generating the reference data XOM resource from the tables to redeploy it whenever an update is needed. This approach is based on the following steps:

- The reference data is captured and maintained in decision tables, as described in Capturing reference data in the rules.
- The ruleset generation uses a ruleset extractor that filters out the reference data tables from the ruleset archive.
- The reference data XOM resource archive is generated based on the content of the decision tables, and uses the Decision Center API to retrieve the source tables. Then, you parse and process them using the Decision Table API (see `IlrDTModel` and the associated classes).

Consider the benefits of the hybrid approach:

- The reference data is captured and managed, and versions are created in the rules repository. You don't need an external management system for reference data, which simplifies the overall decision governance process.
- At run time, the reference data is accessed through the XOM, which is more efficient than through the execution of decision tables. It allows a cleaner integration into the BOM.

However, the hybrid approach requires investment in writing some nontrivial custom code against the Decision Center API that translates the decision tables to the XOM archive and automates the deployment of the XOM archive to the Rule Execution Server.

Also, IBM ODM on Cloud does not allow deploying a customized Decision Center enterprise archive (EAR) file, and the IBM ODM on Cloud Decision Center does not allow you to open remote connections through the `IlrSession` API. Therefore, this option cannot be applied to IBM ODM on Cloud.

Harnessing the reference data in rules

After you choose a method to store the reference data for a decision, consider designs for how to incorporate the reference data into the ruleset.

The first option consists of filtering the reference data to some extent and enriching the context data with the resulting set. For example, the filtering and enrichment can be one or several attributes or flags that you add to the context data to represent more characteristics of the context data that are derived from the reference data. In the example in this tutorial, imagine that each item instance is enriched with the list of states in which it is restricted. Essentially, the context BOM object gets populated from the reference data through a transformation. With this approach, you must pay attention to avoid couching business logic in the transformation.

As a variant, another option is creating more business objects and adding them to the model to represent the reference data. These objects can be populated from the reference data and injected into the rule execution context.

Regardless of the chosen design, defer the access the reference data to when it is needed in the decision. It is common that a business decision has many paths in its logic, and not all the paths require access to all the reference data elements. Consider just-in-time access to the reference data and properly caching the data after it is retrieved.

Conclusion

This tutorial presented two approaches to manage reference data with resource files when the Rule Execution Server cannot access an external data source. The first approach relies on rule project resources to generate reference data as decision tables in the ruleset. The second approach uses files that are deployed with the XOM resource to initialize a Java object singleton pattern.

You can apply the examples and approaches that you learned to your work with decision services.

Acknowledgements

The authors would like to thank Franck Delporte and Peter Holtzman for their review of this tutorial and their suggestions.

Resources

- Accessing external data in a rules application
- Strategies for managing reference data in a business rules application using IBM Operational Decision Manager
- dwAnswers discussion about limits on rows in decision tables
- IBM Operational Decision Manager Developer Center
- IBM Operational Decision Manager documentation on IBM Knowledge Center

About the authors

Pierre Berlandier

Pierre Berlandier has worked as part of ILOG and IBM professional services for 20 years. He helps his clients succeed with different incarnations of the rules programming paradigm, from expert systems, to intelligent agents, including today's business rules applications.

Matt Voss

Matthew (Matt) Voss has worked with expert systems and decision management technologies for more than 10 years, building applications for healthcare, manufacturing, finance, and mortgage industries. He joined IBM in 2013 after several years consulting for IBM ODM engagements.