Reusing rules within a ruleset to avoid creating redundant business rule variations

Skill Level: Intermediate

Raj Rao (rrao2@us.ibm.com)
ILOG Solution Architect
IBM

21 Sep 2011

IBM® WebSphere® ILOG® JRules is IBM's Business Rule Management System (BRMS) that enables business users to dynamically control automated business decisions with business rules. One of the advantages of using BRMS is that rulesets can be reused across different client applications – but this article is not about that. Instead, this article focuses on those situations where a naïve practitioner might clone business rules within a ruleset to apply the same business policy to disparate business entities or different contexts. This article presents such a scenario and outlines different techniques for reusing the rules across different contexts, thereby avoiding duplication and improving maintainability and performance.

Introduction

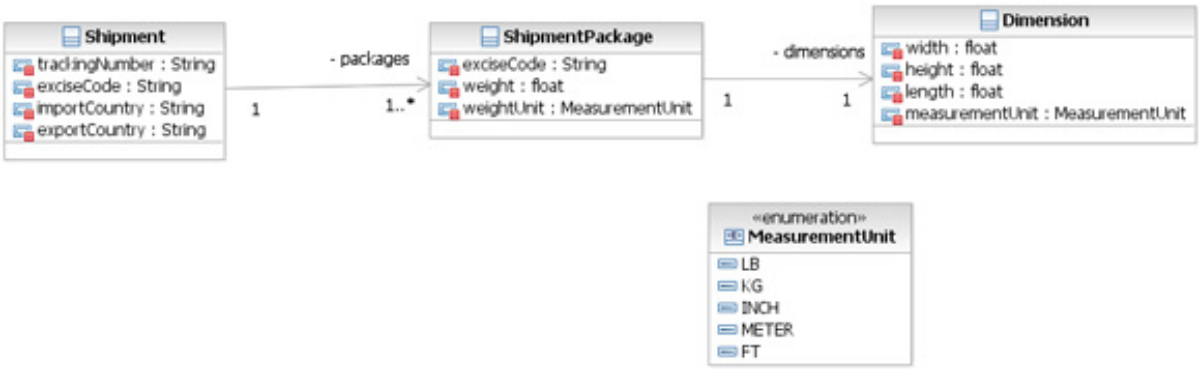
Occasionally, you come across situations where an inefficient implementation leads to the presence of multiple variants of the same business rule in a ruleset. One such scenario is described in this article, along with implementation options to overcome this problem.

In this sample scenario, a (fictional) shipping company called Magic Carpet Worldwide Shipping handles the logistics and paperwork related to international shipments. The company receives shipment orders from customers over the Internet. International shipping requirements change frequently, and so to handle this in an agile fashion, Magic Carpet uses BRMS to automatically validate all shipment orders.

A shipment consists of shipment packages, as shown in the shipment model in

Figure 1. A shipment object identifies the import and export country. Each shipment package has information about its weight and dimensions. Both shipment and shipment packages contain an excise code, which is a code used to categorize the item from an excise duty perspective.

Figure 1. Shipment model



The business policies that Magic Carpet wants to implement are:

- Shipment tracking number must be 14 characters.
- Import and export country must be valid.
- If export country is US, import country cannot be Cuba or Iran.
- Weight of package must not exceed 50 lbs or 22.5 kgs.

In addition, Magic Carpet has a number of policies that apply to excise code (on shipment as well as package). Some of these are general policies that check the format and syntax, while other rules are county specific:

- Rules that check the format of excise code:
  - Length must be 9 or 12 or 13.
  - If length of excise code is 9, the fifth character must be B or Y.
  - If length of excise code is 12 or 13, the seventh character must be L or P.
  - Excise code must not end with 9999.
- Country specific rules for validating excise codes:
  - If shipping to Puerto Rico or Mexico, excise code must start with MC.
  - If shipping to Euro zone countries, excise code must have 13 characters.
  - However, if shipping to or from India, Pakistan or Sri Lanka, excise

code must be 12 chars and the eigth character must be I. This takes higher precedence.

The number and scope of the rules are limited in this scenario to illustrate the main points, but it should be very easy to imagine a typical rule implementation having hundreds of such rules.

Prerequisites and download material

This article is written for the intermediate JRules developer, focuses on a specific area of implementation, and assumes a basic understanding of WebSphere ILOG JRules from a developer's perspective. See Resources for links to help you acquire the prerequisite knowledge for proceeding with this article. The product versions used in this article are:

- WebSphere ILOG Rule Studio V7.1.x
- WebSphere ILOG Rule Execution Server V7.1.x

In this article, only the relevant details of the JRules workspace are discussed. However, if you have Rule Studio 7.1.x installed, you can download the entire workspace, containing the different implementation options, and browse through all the details as a supplement to this article. Separate projects are used to illustrate each of the implementation options discussed in the next section. The projects associated with each implementation are listed in Table 1.

Table 1

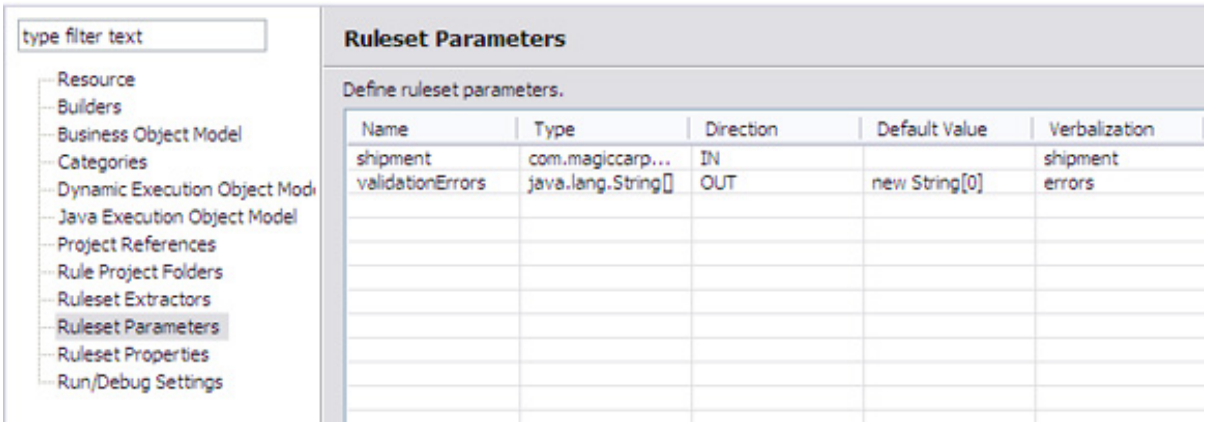
Inplementation option	XOM	Rule project
Option A: Clone rules	ShipmentJavaXOM	ShipmentRules_A
Option B: Common interface	ShipmentJavaXOM_B	ShipmentRules_B
	ShipmentJavaXOM	ShipmentRules_B2
Option C: Internal model	ShipmentJavaXOM_C	ShipmentRules_C
Option D: Separate project	ShipmentJavaXOM	ShipmentRules_D
	none	ExciseRules_D

Implementation options

As you can see from the Magic Carpet business rules above, there are some rules that apply at the shipment level while others apply at the package level. Four different options for implementing these requirements are presented here. Common to all the implementation options is the base Java™-based execution object model (XOM) containing the business entities shown in Figure 1. The rule project simply takes the shipment as the input parameter and returns an array of error messages

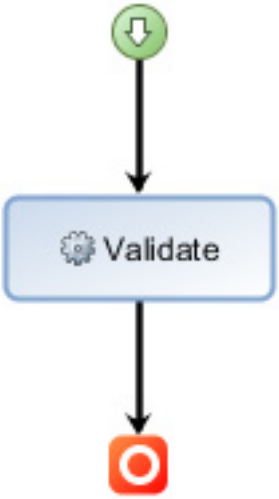
as the output parameter, shown in Figure 2.

Figure 2. Ruleset parameters



Unless specified otherwise, all the implementations use a simple ruleflow with one ruletask, as shown in Figure 3.

Figure 3. Validation ruleflow



Option A: Clone rules

The naïve approach is to clone the rules individually for shipments and shipment packages. For example, consider the business policy:

If length of excise code is 9, the fifth char must be B or Y.

This policy applies to excise codes in shipments and well as shipment packages. The shipment business rule that implements this policy is shown in Listing 1.

Listing 1

```
if
```

```
        the excise code of shipment is not empty and
        the length of the excise code of shipment is 9
        and the char at 5 in the excise code of shipment is not one of { "B", "Y"}
then
        add error: "Excise code " + the excise code of shipment +
                ": Fifth char must be B or Y";
```

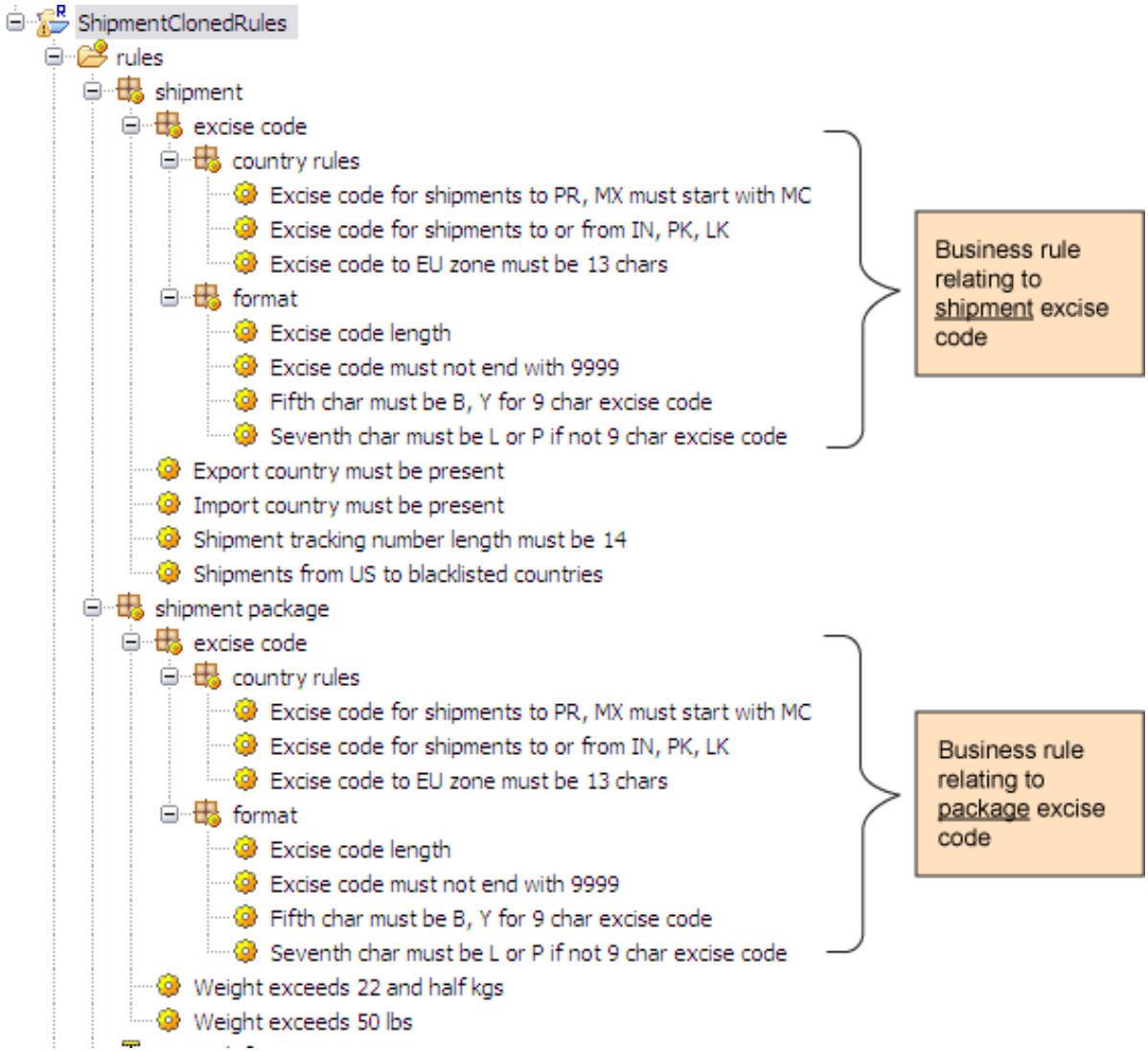
The same business policy is implemented for shipment packages with the rule variant in Listing 2.

Listing 2

```
definitions
        set package to a shipment package in the packages of shipment ;
if
        the excise code of package is not empty and
        the length of the excise code of package is 9
        and the char at 5 in the excise code of package is not one of { "B", "Y"}
then
        add error: "Excise code " + the excise code of package +
                ": Fifth char must be B or Y";
```

Even though the basic logic is the same, multiple variants of the business rule are required to implement the logic for different contexts. As shown in Figure 4, all the excise code business policies are implemented separately for shipments and shipment packages.

Figure 4. Rule project structure

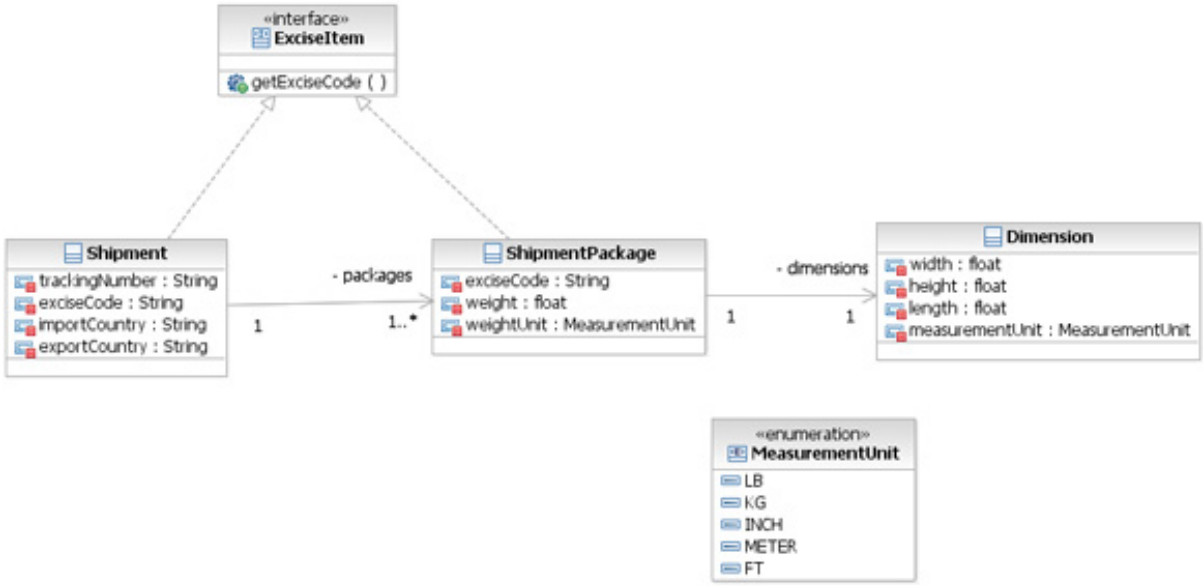


This approach causes maintenance issues, since any change to the business logic implies code changes in multiple rules. Additionally, this can lead to rule-bloat, which has negative implications for authoring and execution performance.

Option B: Apply common interface in domain model

The second option for implementing this relies on modifying the XOM to provide a common underpinning to the different contexts for which the rules apply. In this scenario, this option takes the form of creating an interface called ExciseItem with a getExciseCode() method defined. Shipment and ShipmentPackage implement this interface, as shown in Figure 5.

Figure 5. Modified shipment model with ExciseItem interface



With this modified shipment model, it is possible to write the excise code rules based on ExciseItem. A single rule can be written to implement the business policy:

If length of excise code is 9, the 5rd char must be B or Y.

The business rule implementation that works on both Shipment and ShipmentPackage is shown in Listing 3.

Listing 3

```
definitions
        set exciseItem to an excise item ;
if
        the excise code of exciseItem  is not empty and
        the length of the excise code of exciseItem  is 9
        and the char at 5 in the excise code of exciseItem  is not one of { "B", "Y"}
then
        add error: "Excise code " + the excise code of exciseItem  +
                ": Fifth char must be B or Y";
```

These rules rely on having instances of the ExciseItem in the rule engine working memory. This is handled by the initial action of the ruleflow, where you insert the shipment and all the packages into the working memory (Listing 4).

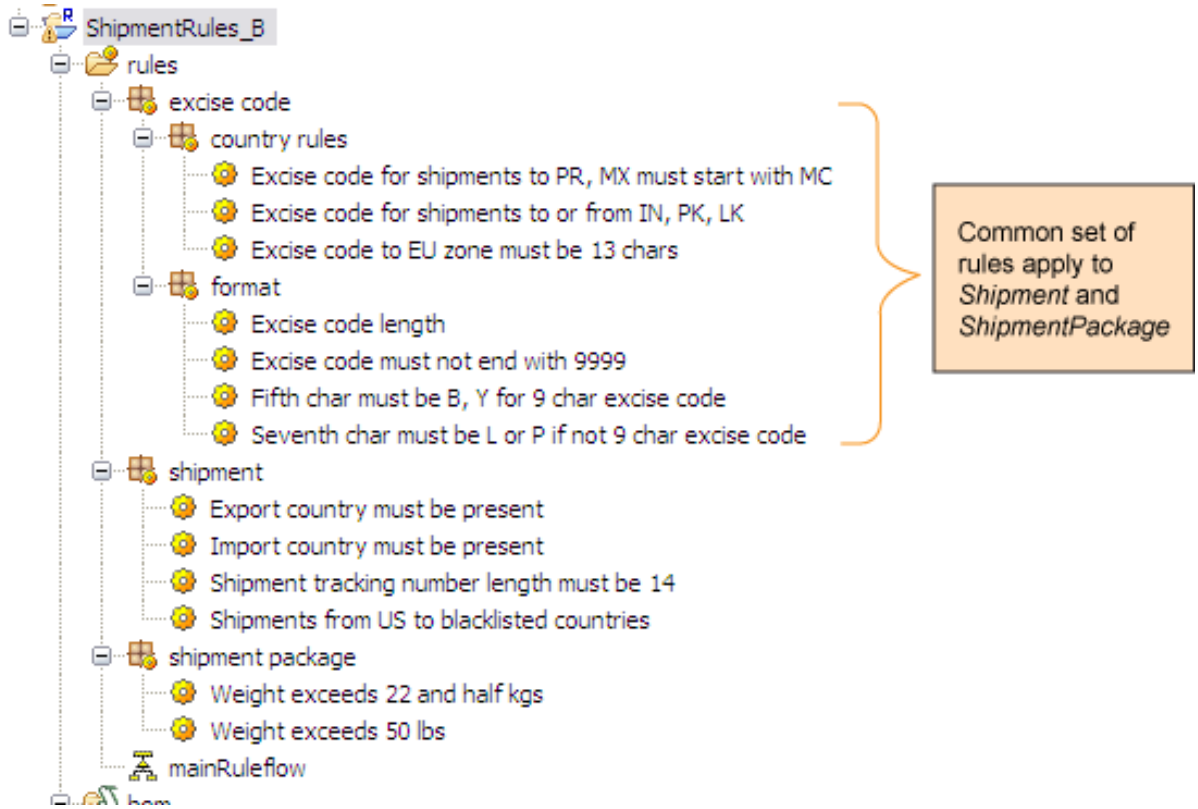
Listing 4

```
insert(shipment);
java.util.Iterator it = shipment.packages.iterator();
while (it.hasNext()) {
        insert (it.next());
}
```

Using this approach, only one variant of the business rule suffices for both

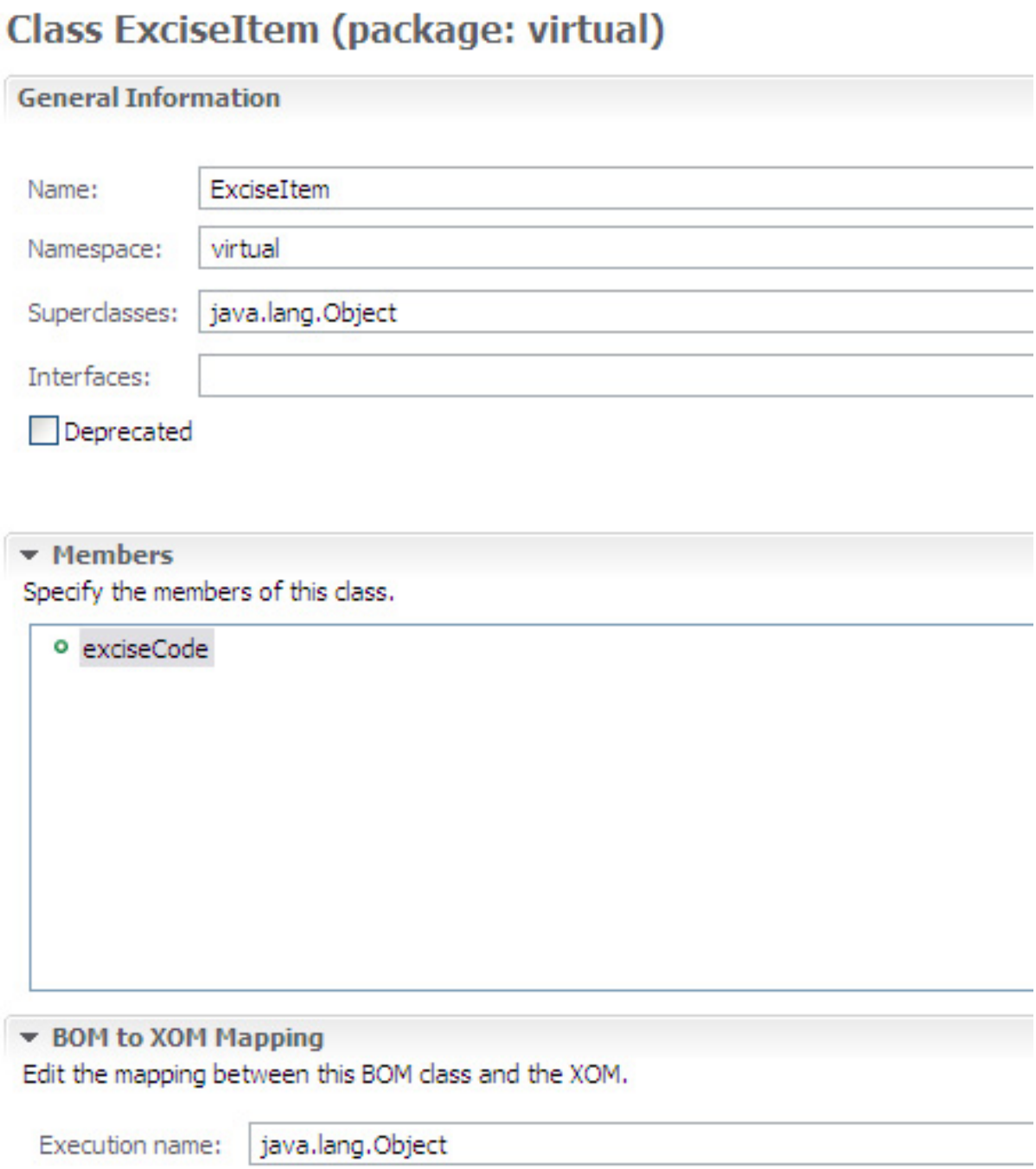
shipments and shipment packages. This results in far fewer rules, as seen in the product structure in Figure 6.

Figure 6. Project structure with common rules using ExciseItem



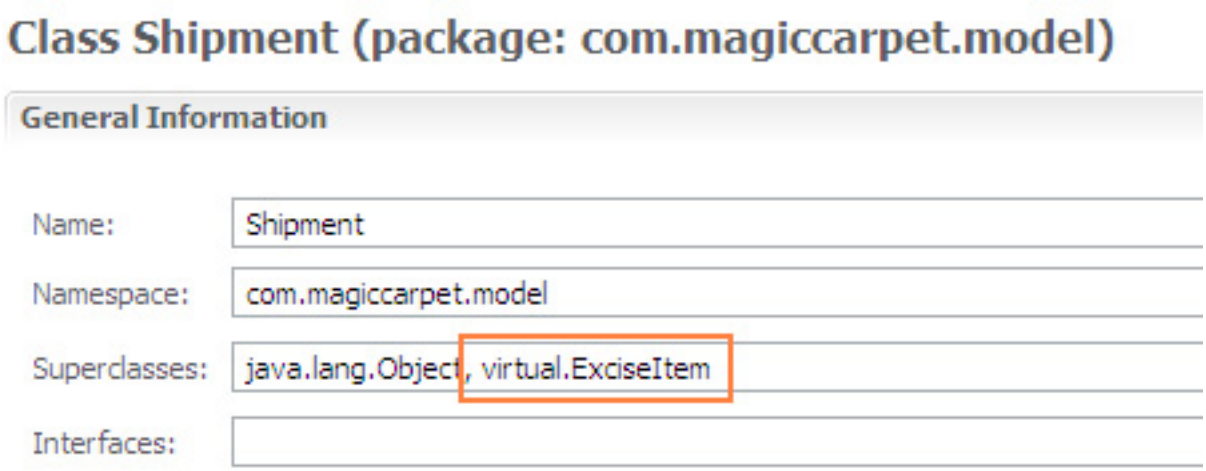
This option, as described, is only feasible when the rule developer is permitted to change the domain model. In situations where the XOM is controlled by an external group, you take a slightly different tack (Option B2 in the downloadable workspace). Instead of creating an ExciseItem class in the XOM, a virtual class is created in the BOM with one member called exciseCode, as shown in Figure 7.

Figure 7. Virtual ExciseItem in the BOM



In the Shipment and ShipmentPackage BOM classes, add this virtual ExciseItem as a superclass, as shown in Figure 8.

Figure 8. ExciseItem added as a superclass



The member exciseCode in ExciseItem is marked as read-only in the BOM and is supplied with a BOM to XOM mapping that retrieves the value from either a Shipment or a ShipmentPackage.

Listing 5

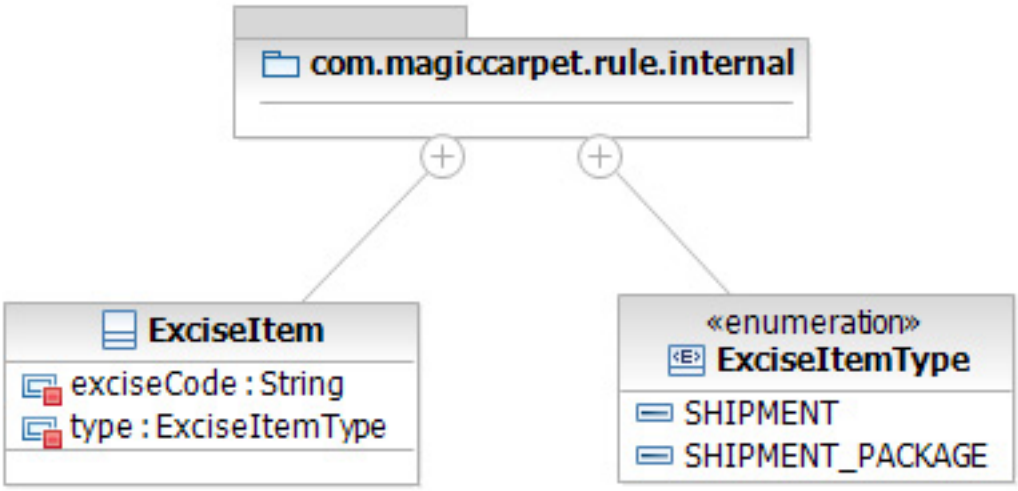
```
if (this instanceof Shipment)
        return ((Shipment)this).exciseCode;
else
        return ((ShipmentPackage)this).exciseCode;
```

The rest of the code, including the business rules using this approach, are identical to those described earlier with the ExciseItem defined in the XOM.

Option C: Use internal model

This approach is similar to option B, in that you add to the object model to achieve your goal of not having multiple rule variants. In this case, however, instead of modifying the original domain model, you add some internal classes strictly to be used by rules. This can be defined in its own Java project if necessary, although for the purpose of this article, it is defined in the same XOM as the other domain classes. An internal class called ExciseItem is added to the com.magic.rule.internal package, as shown in Figure 9. This class basically contains an excise code and an attribute indicating the source of this excise code.

Figure 9. Model addendum with internal classes used only by the rules



During the initial action of the ruleflow, instances of ExciseItem are created from the shipment and shipment packages and inserted into the working memory of the rule engine. The code is excerpted in Listing 6.

Listing 6

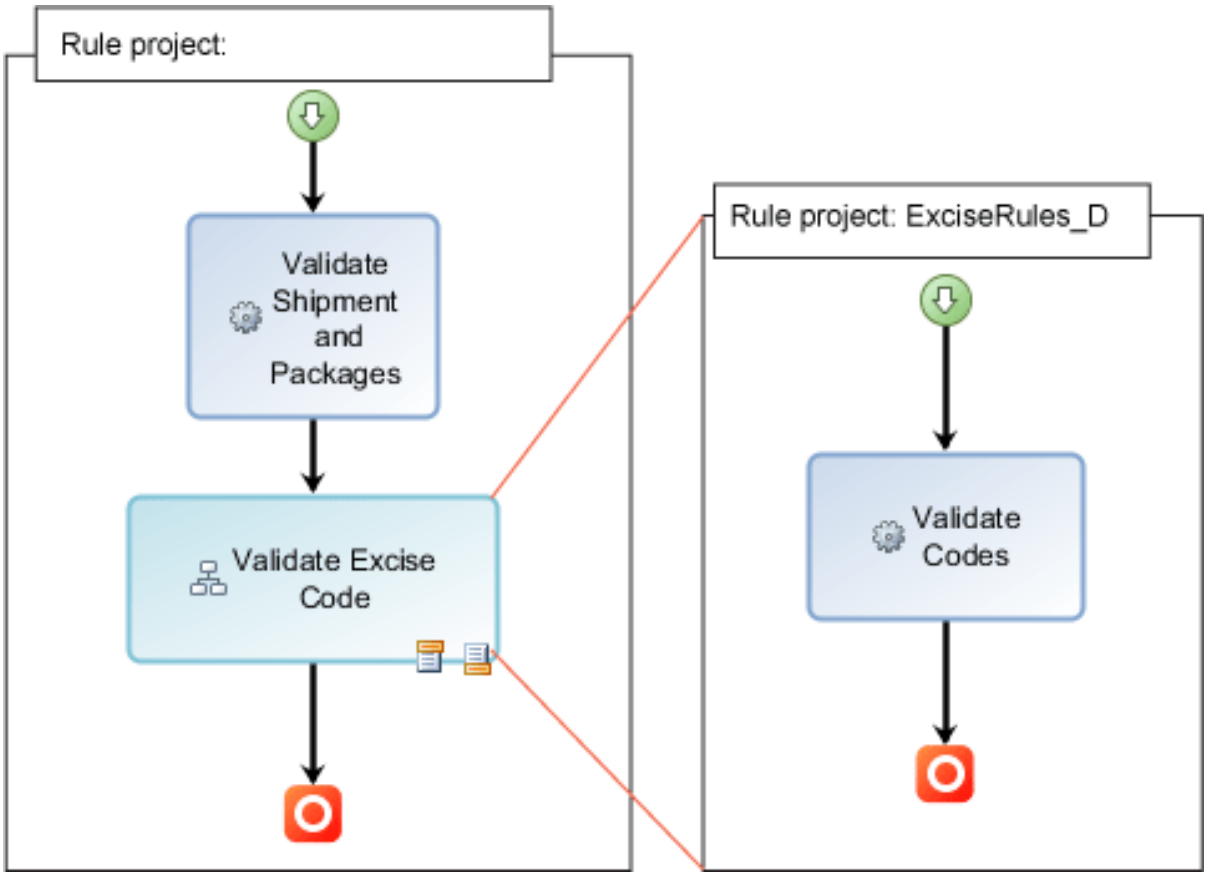
```
insert (new ExciseItem(shipment.exciseCode, ExciseItemType.SHIPMENT));
java.util.Iterator it = shipment.packages.iterator();
while (it.hasNext()) {
        ShipmentPackage pkg = (ShipmentPackage)it.next();
        insert (new ExciseItem(pkg.exciseCode, ExciseItemType.SHIPMENT_PACKAGE));
}
```

The rule structure and rules are identical to that which you've seen in Option B. Therefore, you achieve the same benefits outlined earlier with the additional advantage that the external domain model is not modified. While not necessary for this scenario, know that it is possible to define more complex internal requests using this technique.

Option D: Create separate rule project

Yet another alternative is to isolate the excise code validation rules into its own rule project and use them from other rule projects. In this implementation approach, you add a separate subflow task to validate the excise codes in the main ruleflow of the shipment rules. This subflow task uses the ruleflow defined in the excise code validation rule project, as illustrated in Figure 10. This figure shows the ruleflow of ShipmentRules on the left and the ruleflow of the "embedded" rule project ExciseRules on the right.

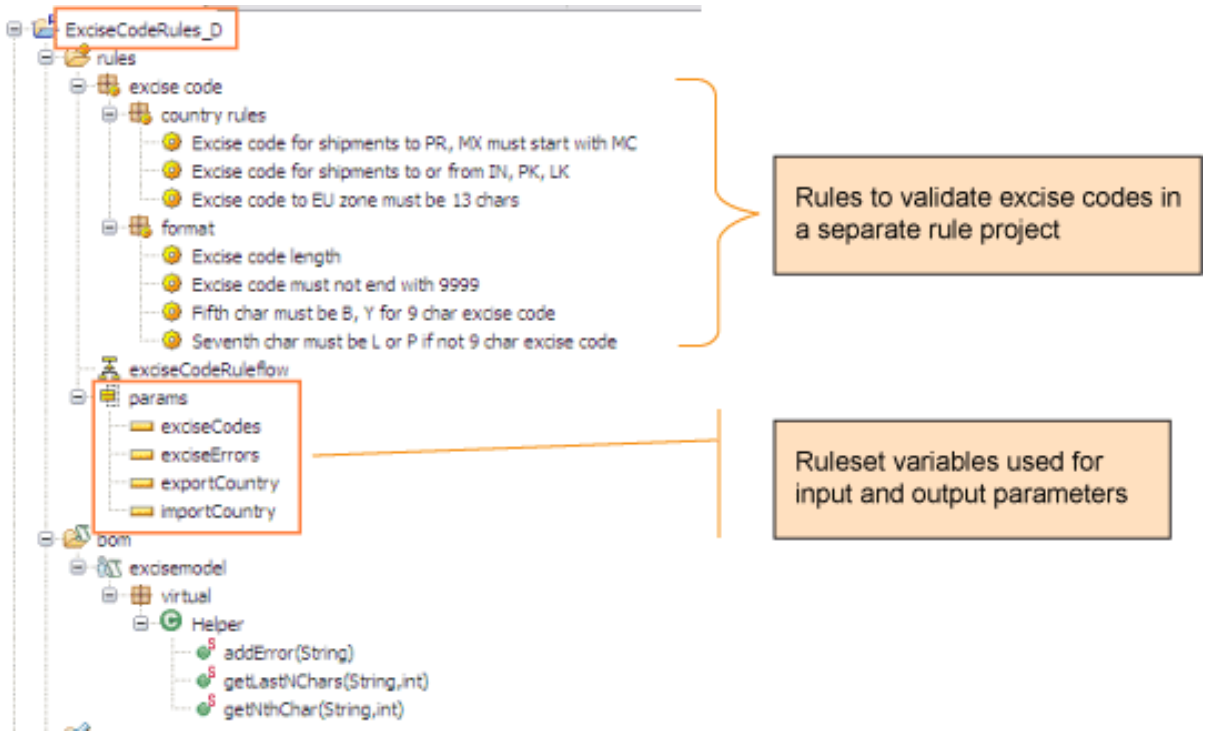
Figure 10. Shipment ruleflow with validate excise codes subflow



Because the ruleflow in ExciseRules is only used as a subflow, the main flow task property must be set to false.

The rule project for validation of excise codes does not use the shipment domain model at all. It uses a simpler model by leveraging the fact that validation of an excise code only depends on the import country and the export country. These are simply represented as strings in the "params" variable set. This is shown in Figure 11, which depicts the new rule project to validate excise codes.

Figure 11. Project structure of rules to validate excise codes



The excise code validation rules use these variables in the rule conditions and actions. For example, the rule that checks the format of a 9-character excise code is shown in Listing 7. Notice that there is no reference to a shipment or a package, but merely to variables such as ‘excise codes’. Errors are added to the exciseErrors variable.

Listing 7

```
definitions
        set exciseCode to a string in 'excise codes';
if
        exciseCode is not empty and
        the length of exciseCode is 9
        and the char at 5 in exciseCode is not one of { "B", "Y"}
then
        add error: "Excise code " + exciseCode + ": Fifth char must be B or Y";
```

So the question that arises is: where do these input variables get set? Well, these are set in the invoking ruleflow (in ShipmentRules). The initial action of the Validate Excise Code rule task sets the import country, export country, and the list of excise codes to validate, as shown in Listing 8. Here, virtual.Helper.addExciseCode is simply a virtual method that adds an excise code to the ruleset variable exciseCodes.

Listing 8

```
virtual.Helper.addExciseCode(shipment.exciseCode);
importCountry = shipment.importCountry;
exportCountry = shipment.exportCountry;
```

```
java.util.Iterator it = shipment.packages.iterator();
while (it.hasNext()) {
        ShipmentPackage pkg = (ShipmentPackage)it.next();
        virtual.Helper.addExciseCode (pkg.exciseCode);
}
```

In the final action of Validate Excise Code, the errors found during processing of the excise codes (that is, exciseErrors) are added to the output (Listing 9).

Listing 9

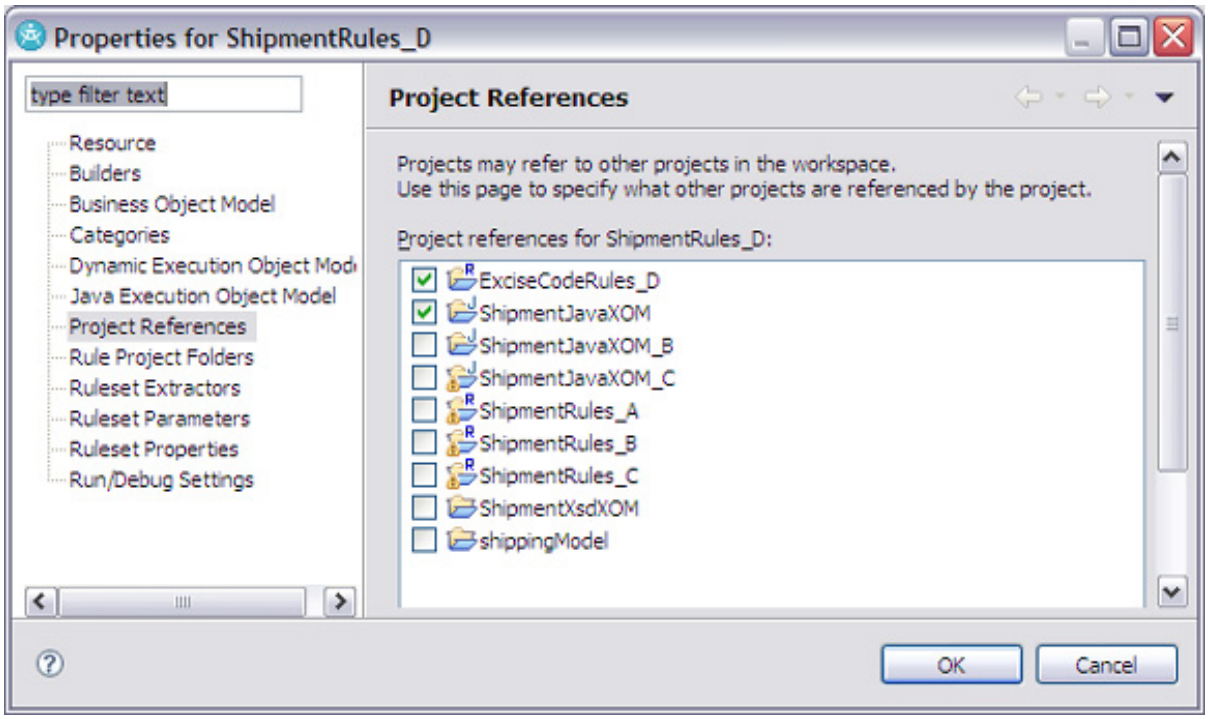
```
int i;
for (i=0; i<exciseErrors.length; i++) {
        virtual.Helper.addError(exciseErrors[i]);
}
```

To recap, validation occurs as follows:

1. The Validate Shipment and Packages rule task in ShipmentRules applies all validation rules with the exception of excise codes.
2. In the initial action of the Validate Excise Code rule task in ShipmentRules, the ruleset variables of ExciseRules (import country, export country, excise codes) are set.
3. Rules in ExciseRules are executed by the rule task Validate Excise Code.
4. The final task of Validate Excise Code takes the errors generated during excise code validation and adds it to the output.

Notice that the rule project, ShipmentRules, can access rules and variables defined in ExciseRules only because ExciseRules is marked as a project reference for ShipmentRules, as illustrated in the Project References property of ShipmentRules in Figure 12.

Figure 12. ExciseCodeRules used as a project reference in ShipmentRules



Now that the business rules for validating excise codes are moved to a different rule project, the ShipmentRules project structure is much simpler, as illustrated in Figure 13.

Figure 13. Rules in rule project ShipmentRules

Even though the excise code rules are moved to a different rule project, they are deployed as part of the shipment validation ruleset. In other words, you still have a single ruleset that includes rules from both the projects.

However, an interesting possibility arises – that of being able to deploy the excise code validation rules independently as a decision service. This service can then be

used by any application capable of invoking a web service, if it needs validation at that granular level. Considerations relating to performance and business suitability (which are beyond the scope of this article) determine if this service granularity is warranted.

In addition to avoiding redundant cloning of rules, this approach has the advantage that the excise code validation rules can be reused by other rule projects in the future, even if these rule projects have a very different domain model. This also means that the excise code validation rules are insulated from changes in the Shipment domain model. Moreover, this approach enables different groups to be responsible for different aspects of validation; one group can handle shipment validation rules, while another handles excise code validation. Of course, this approach by no means precludes a single group from being responsible for both rule projects.

On the flip side, because these rules are in two different rule projects, a rule author using Rule Team Server would have to switch between these projects to edit them. Furthermore, as excise code rules evolve, they might need to use additional parameters. For example, if in the future the excise code rules depend not just on import and export country, but also on a shipment type, then this would need to be passed in as a new variable. This will necessitate creation of new ruleset variables and changing the enclosing ruleflows to populate these variables.

Comparison of options

Table 2 summarizes the pros and cons associated with each of the implementation options discussed here.

Table 2

Option	Pros	Cons
Option A: Clone rules	• Straightforward implementation.	• Multiple rules implementing the same business logic leads to maintenance issues. • Rule bloat negatively impacts performance.
Option B: Common interface	• Overcomes need for multiple rule variants. Therefore, easier to maintain.	• If domain model is controlled by an external group, this requires somewhat complex BOM

		changes to implement virtual classes.
Option C: Internal model	• Overcomes need for multiple rule variants. Therefore, easier to maintain.	• Need to define additional internal classes in XOM.
Option D: Separate project	• Overcomes need for multiple rule variants. Therefore, easier to maintain. • Can be reused by other rule projects as well. • Excise code rules are insulated from changes in Shipment domain model. • Rule authoring responsibilities can be distributed to different groups.	• Need to define new rule project. • More complex ruleflows. • Using the Rule Team Server, a rule author will have to switch to a different rule project just to view or edit these rules. • May need to add more variables to support rule evolution (when they start using more inputs).

Conclusion

If you find yourself in the sticky, uncomfortable situation of creating multiple variants of a business rule simply because a business policy needs to be applied in different contexts, take a step back and know that you have several options for avoiding this. As discussed in this article, each of these options have their pros and cons, so be sure to evaluate these options in the context of your development environment and future needs and select the best one.

Downloads

Description	Name	Size	Download method
Code sample	exportedProjects.zip	564 KB	HTTP

Information about download methods

Resources

Learn

- WebSphere ILOG JRules BRMS Information Center
- WebSphere ILOG JRules product information
- WebSphere ILOG BRMS resources
- Series: Customizing WebSphere ILOG JRules for rule authoring
- IBM developerWorks WebSphere

Get products and technologies

- WebSphere ILOG JRules V7.1 trial download

Discuss

- Blog: Good Decision! a decision management blog

About the author

Raj Rao
Rajesh (Raj) Rao has been working in the area of expert systems and business rule management systems for over 20 years, during which time he has applied business rules technology to build diagnostic, scheduling, qualification and configuration applications across various domains such as manufacturing, transportation and finance. He has been with IBM for close to 2 years. With a background in Artificial Intelligence, his interests include Natural Language Processing and Semantic Web.