Implementing advanced rule governance in IBM Operational Decision Management

Nigel T. Crowther

May 22, 2013

This article introduces a rule-based Rule Governance design pattern. The design pattern can be applied to all versions of IBM® Operational Decision Management to improve the agility and strength of Decision Center governance. You can build your own implementation based on this pattern, or can contact the author for details on the IBM Rule Governance Asset.

Introduction

This article introduces an IBM ODM governance framework for advanced governance solutions. It proposes a flexible alternative to the usual rule governance implementation based on configurable Java business logic, as illustrated by the rule governance product sample. We show that using rules rather than Java to govern the change process improves the power and agility of advanced governance within ODM.

For governance of change activities, we recommend you look at the new governance features in ODM V8.5.

The Rule Governance sample

The architecture of the governance extension provided in the product samples is shown in Figure 1. At the heart of the sample is the session controller, which enforces access permissions based on state. The Decision Center calls out to a custom session controller called WorkflowSessionController, which extends the IlrDefaultSessionController. The WorkflowSessionController overrides methods such as checkUpdate, checkCreate and checkDelete to perform governance operations.

The limitations of this architecture are:

- State base permissions are controlled by a text configuration file embedded within the Decision Center EAR. Configuration changes cannot be applied without redeploying the EAR.
- Custom business logic is implemented in Java. Java changes are expensive to implement and test and may need to be merged with each new version of ODM API. Also the Decision Center EAR needs to be redeployed with each change to the code.
• Project access is controlled by combining two different concepts of roles: functional and project roles. This can lead to a proliferation of roles. For more details on this, see the Assign roles section.

Figure 1. Architecture of the Rule Governance ODM product sample

Rule-based rule governance

Consider what happens if you delegate the governance logic from the EAR file to a decision service, as shown in Figure 2.

Figure 2. Rule governance decision service

The architecture in Figure 2 assumes Decision Center resides on the same application server as Decision Server to allow local POJO (Plain Old Java Object) invocation. An alternative approach would be for the J2SE Rule Engine to be embedded in the Decision Center EAR. This approach would be used if Decision Center does not have access to a Decision Server.

The parameter interface between the Governance Decision Service and the Governance Session Controller is implemented with the Generic Ruleset Signature Pattern. This pattern enables the rule model (.brmx) to be changed without having to redeploy the Decision Center EAR. The
object model is built using Excel dynamic domains which map to the rule model. For details on the Generic Ruleset Signature Pattern, see Introducing the Generic Ruleset Signature pattern for WebSphere Operational Decision Management V7.5, Part 1.

Governance rule flow

The rule flow inside the Governance decision service is shown in Figure 3. It is divided into six sub-flows that largely correspond to the session controller methods.

Figure 3. Governance rule flow

![Governance rule flow diagram]

The purpose of the six subflows are as follows:

- **Roles** - return the functional roles
- **Status** - return the status transitions given an artifact's current status
- **Permissions** - determine the create/update/delete permissions for a rule artifact
- **Committed** - sends notification messages given pre- and post commit status
- **Validation** - validates rules and rule metadata and returns errors.
- **StyleCheck** - Style-checks rules and returns warnings.

In the following sections, we'll take a look at each of these operations.

Roles

Role permissions in Decision Center are applicable across all projects. The way to define project access permissions is to create combined roles for each project and function. For example, a function of Tester and a project of LoanValidation would produce the TesterLoanValidation role. Two projects called LoanValidation and Annuity and two functions called Tester and Author would produce the following combined roles:

- AuthorLoanValidation
- TesterLoanValidation
- AuthorAnnuity
- TesterAnnuity

A tester accessing the LoanValidation project would be assigned the TesterLoanValidation role. Similarly, a tester accessing the Annuity project would be assigned the TesterAnnuity role. A tester allowed access to both projects would be assigned both the TesterLoanValidation and TesterAnnuity roles.
Adopting this approach for many projects can lead to a proliferation of roles. For example, six functional roles and ten projects would require sixty roles.

A simpler approach is to split project roles and functional roles, then restrict the Decision Center security model to operate only on project roles and restrict the Governance decision service to operate only on functional roles. Figure 4 illustrates the separation of functional and project roles.

**Figure 4. Separation of roles**

The implementation of project and functional roles is simple. Just assign project roles in Decision Center as shown in Figure 5 and Figure 6.

**Figure 5. Assign project role to LoanValidation project**

**Figure 6. Assign project role to Annuity project**

Assign functional roles in Rule Governance as shown in Figure 7. Now project access is determined by Decision Center and functional access is determined by the Rule Governance permission rules described in the next section.
Figure 7. Assign functional roles in Rule Governance

Permissions

As described in the previous section, we use the Decision Center security model to determine project access and delegate fine-grained access permissions to Rule Governance. In doing so, not only does this allow us to separate functional roles from project roles, but it allows fine-grained access permissions. For example you could restrict access by rule state, folder, rule name and expiration date.

In Figure 8, the author is allowed to change decision table attributes in new, defined and cancelled state, but can only change the rule body (definition) in new state.

Figure 8. Permissions decision table

Status

The diagram in Figure 9 shows state transitions for a rule author. The transitions are enforced in the decision table shown in Figure 10.

Figure 9. State transition diagram

Figure 10. State transitions defined in a decision table
You can see that in row 1 in Figure 10, a newly created rule is set to new state. In row 2 the author is restricted to moving from new to defined or cancelled state.

### Validation rules

Governance validation rules enforce who can do what and prevent the rule from being committed if there is an error. **Figure 11** shows a governance validation rule preventing an author from testing his own rule. This can happen when a user is assigned both Tester and Author roles. An author testing his or her own rule is not desirable behavior because it breaks the “four eyes” principle.

**Figure 11. Rule preventing author from testing his or her own rule**

```plaintext
definitions
    set createdAuthor to the string createdBy ;
    set lastAuthor to the string lastChangedBy ;
    set thisUser to the string loginUser ;
if
    the string status is tested
    and ( createdAuthor is thisUser or lastAuthor is thisUser )
    and thisUser is not "admUser"
then
    set validationError as "You cannot test rules you have authored" ;
```

The governance validation rule in **Figure 12** prevents a user from submitting a rule without documentation.

**Figure 12. Rule preventing author from saving without documentation**

```plaintext
if
    the string documentation is empty
    and the string status is defined
then
    set validationError as "Please add documentation" ;
```

### Style check rules

Style check rules check for potential problems rather than errors. Unlike the validation rules, Style rules do not prevent the user from saving the rule, but simply create a warning message. It is up to the author and (ultimately the reviewer) to decide whether the warning advice should be taken. **Figure 13** shows a style rule enforcing naming standards. The standard requires rule names to start with an upper case character and to be followed by one or more lowercase characters or numbers.

**Figure 13. Style rule enforcing naming standards**

```plaintext
definitions
    set 'the rulename format' to "^[a-zA-Z-0-9]+" ;
    set ruleName to the string name ;
if
    it is not true that ruleName matches 'the rulename format'
then
    add warning "Invalid rule name: Please use format: " + 'the rulename format" to warnings ;
```
If a name breaks this convention, a warning message appears in the Warnings property field. In Figure 14, the user has entered an invalid rule name containing a dollar symbol. This is detected by the name validation rule in Figure 13, which writes to the Warnings property field.

**Figure 14. Warning message**

To highlight the issue further, a warning symbol appears in the Rule Explorer view and the warning message is in the tool tip, as shown in Figure 15.

**Figure 15. Warning indicator in Rule Explorer**
Other examples of style rules are:

- Naming standards for column names
- Decision table maximum row limits
- Decision table cell value maximum length limits
- Thresholds for numeric values in a decision table

Examples of these rules are shown in Figures 16, 17, 18 and 19, respectively.

**Figure 16. Style rule to check whether a decision table has invalid column names**

```plaintext
definitions
set 'the columnname format' to "^[A-Za-z]+$";
set 'columnNames' to get column names from editedDT;
set 'columnName' to a string in columnNames;
if
  the rule artefact type is DecisionTable
  and it is not true that columnName matches 'the columnname format'
then
  add warning "Invalid column name " + columnName + ". Please use format: " + 'the columnname format' to warnings;
```

**Figure 17. Style rule to check whether a decision table has too many rows**

```plaintext
definitions
set 'maxNumberRows' to 100;
set 'numRowsEdited' to get last edited number of rows
if
  the rule artefact type is DecisionTable
  and numRowsEdited is more than maxNumberRows
then
  add warning "Decision Table exceeds recommended " + maxNumberRows + 
  " rows. Current number of rows is " + numRowsEdited to warnings;
```

**Figure 18. Style rule to check whether a decision table cell value exceeds 16 characters**

```plaintext
definitions
set maxSize to 16;
set 'columnName' to "Fee Descriptor";
set 'oversize values' to check max length of maxSize
  of table editedDT column columnName;
if
  'oversize values' is not ""
then
  add warning "" + 'columnName' + 
  " contains oversize values: " + 'oversize values' + 
  "\nMaximum length is " + maxSize to warnings;
```
Figure 19. Style rule to check whether change is beyond a threshold limit

```
definitions
  set 'change threshold' to 25 ;
  set 'column' to "Per Item Rate";
  set 'percentage changes' to get all the absolute delta changes
  between the values of the editedDT and persistedDT
  for the DT column called 'column' ;
  set 'variance' to a number in 'percentage changes' ;

if
  the string rulePackage is "5-fees"
  and 'variance' above 'change threshold'
then
  add warning "Change variance " +
  'variance' + "; is beyond threshold of " + 'change threshold' +
  "; % for column " + 'column' to warnings;
```

Committed Rules

Committed rules are called on the onCommit event and determine whether a notification should be sent. A notification could be email, SMS or written to a database. Figure 20 shows a state transition from New to Defined triggering an email. The triggering rule is shown in 21.

Figure 20. Email notifications

Figure 21. Email notification rule

```
definitions
  set recipient to "testers@acme.com" ;
  set changeRequest to the string currentChangeRequestId ;

if
  the boolean Level2Approval is false
  and the string oldStatus is new
  and the string status is defined
then
  set sendNotification as true ;
  set notificationSubject as changeRequest + " implemented" ;
  set notificationBody as "Please test " + changeRequest ;
  set notificationBody as the string notificationBody + "\n\nURL: \n" + the string URL ;
```
The IBM ODM Rule Governance asset

All of the features described in this article and more are implemented in the IBM Rule Governance asset. This asset provides a customizable governance solution for advanced governance implementations. For more information and to obtain the asset, contact the author.

Conclusion

This article presented a rule-based Rule Governance design pattern that uses business rules rather than Java to improve the agility and power of governance implementations for Decision Center.

Acknowledgements

The author would like to thank Pierre Berlandier and Jonathon Carr for reviewing this article.
Related topics

- For more information about the IBM Rule Governance asset, contact Nigel Crowther
- IBM developerWorks Middleware

© Copyright IBM Corporation 2013
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)