Managing rule project versions: Introducing branches in WebSphere Operational Decision Management V7.5

Pierre Berlandier (pberland@us.ibm.com)
Senior Technical Staff Member
I.B.M.

14 December 2011

Using three common development situations that require working on concurrent rule project versions, my goal in this article is twofold: first, I'll present strategies to best address these situations with the tools available in WebSphere® JRules V7.1, and second, I'll introduce the new project branches feature of WebSphere Operational Decision Manager V7.5 Decision Center, and show you how this feature drastically simplifies rule project maintenance tasks.

Use cases for rule project versioning

Managing multiple versions of a rule project is usually not needed until the first cut of the ruleset has been deployed in a production environment and the rule project goes into its maintenance phase. After this point, project versions are needed to address the exceptional cases of bug fixes and major project refactoring, and to support ongoing concurrent enhancements to the rule service. The details of these scenarios are the following:

- **Bug fixes**: In this scenario, a set of rules have been updated and promoted to the QA environment (or maybe even up to the production environment), and it turns out that these updates are introducing a defect in the decision service. Between the time when the ruleset is promoted and the defect is detected, another batch of rule updates may have already started in the rule authoring environment. Before they can start fixing the defect, the rule authors must get access to a workable version of the project from which the defective ruleset was promoted.
- **Rule project refactoring**: In this scenario, a significant technical update to the ruleset is implemented in the IT rule development environment -- in this case, Rule Studio in WebSphere ILOG JRules V7.1 (hereafter JRules) or Rule Designer in WebSphere Operational Decision Management V7.5 (hereafter Decision Management). This type of update includes, for example, a change in the ruleset parameters or in the rule flow, and most often a refactoring of the Business Object Model (BOM) resulting from a change in the eXecution Object Model (XOM). While this technical update is being prepared and tested on the IT side, the maintenance of the production ruleset must continue, thus requiring two concurrent active versions of the rule project.

- **Concurrent rule service enhancements**: Change requests to a complex rule-based decision service don't usually come in an orderly and sequenced fashion. They may come from different lines of business that participate to the decision, each of them with a specific schedule for their release, and each of them with different levels of complexity. Therefore, the rule authors need to be able to work concurrently on writing rules in different areas of the rule project, while only a portion of the changes are released at a given point in time.

Managing rule project versions with JRules V7.1

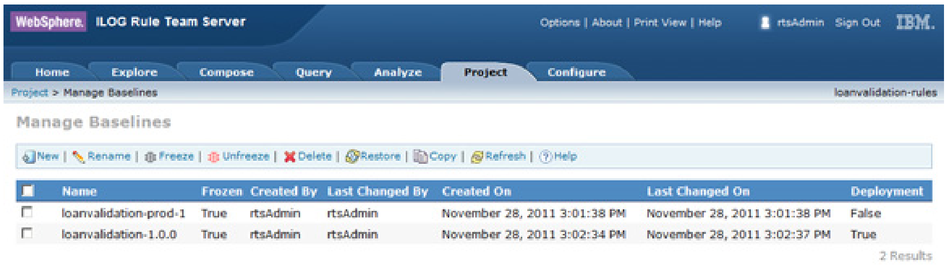
With Rule Team Server (RTS) V7.1, the rule repository maintains a single active state for a given rule project (the "current project state"). This makes the management of concurrent rule project versions a non-standard operation. However, JRules provides a palette of product features and techniques which, along with some well-defined human processes, allow users to emulate concurrent versions or work around the need for versions.

The sections below present these features and techniques and show how they can be applied to our three uses cases for rule project versions.

Using RTS baselines

RTS offers two different types of baselines: *development* baselines and *deployment* baselines. The former allows the user to create a snapshot of the current project state at any time. The latter is created as part of an optional step of the RuleApp deployment process. Figure 1 shows the baseline management page from RTS. The type of the baseline is indicated in the **Deployment** column of the table.

Figure 1. RTS baselines


WebSphere.
ILOG Rule Team Server
Options | About | Print View | Help
rtsAdmin Sign Out
IBM.
Home
Explore
Compose
Query
Analyze
Project
Configure
Project > Manage Baselines
loanvalidation-rules
Manage Baselines
New | Rename | Freeze | Unfreeze | Delete | Restore | Copy | Refresh | Help

	Name	Frozen	Created By	Last Changed By	Created On	Last Changed On	Deployment
☐	loanvalidation-prod-1	True	rtsAdmin	rtsAdmin	November 28, 2011 3:01:38 PM	November 28, 2011 3:01:38 PM	False
☐	loanvalidation-1.0.0	True	rtsAdmin	rtsAdmin	November 28, 2011 3:02:34 PM	November 28, 2011 3:02:37 PM	True

2 Results

After they've been created, baselines can be used as a historical reference and consulted to browse their content. But more interestingly for the purpose of managing multiple project versions:

- The content of existing baselines can be updated, and
- The content of development baselines can be restored.

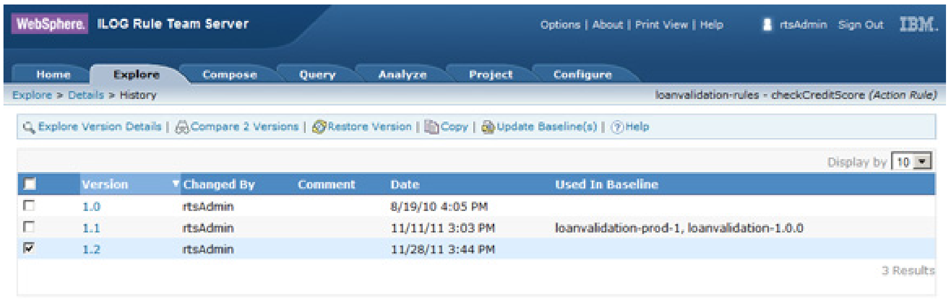
Updating baselines

The default RTS configuration is to create baselines in a "frozen" state, in which their content cannot be modified. Ideally, they should stay that way throughout their lifespan, since their original intent is to associate a tag with a known and certifiable project state. However, once unfrozen, the baseline content can be updated and thus can be considered as a workable version of the rule project.

To unfreeze a baseline, from the **Manage Baselines** page, select the desired baseline, then click **Unfreeze** (see Figure 1). Once a baseline is unfrozen, you can update its content on a rule-by-rule basis, by doing the following:

1. Start by exploring the history of a rule and select the rule version that should be copied to the baseline. In our example, the deployment baseline **loanvalidation-1.0.0** currently uses version **1.1** of the **checkCreditScore** rule. Assume that we want now to have the baseline use version **1.2**. To do that, select it from the history, as shown in Figure 2.

   Figure 2. Change baseline version

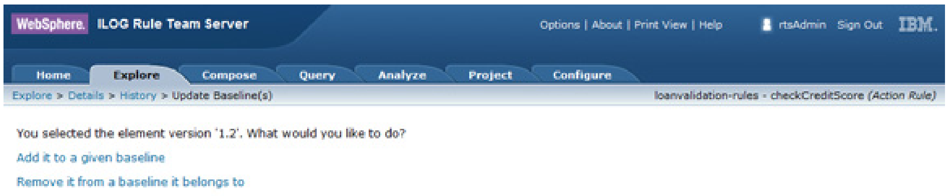
   
   WebSphere. ILOG Rule Team Server
   Options | About | Print View | Help
   rtsAdmin Sign Out IBM.
   Home Explore Compose Query Analyze Project Configure
   Explore > Details > History
   loanvalidation-rules - checkCreditScore (Action Rule)
   Explore Version Details | Compare 2 Versions | Restore Version | Copy | Update Baseline(s) | Help
   Display by 10

☐	Version	Changed By	Comment	Date	Used In Baseline
☐	1.0	rtsAdmin		8/19/10 4:05 PM	
☐	1.1	rtsAdmin		11/11/11 3:03 PM	loanvalidation-prod-1, loanvalidation-1.0.0
☑	1.2	rtsAdmin		11/28/11 3:44 PM	

   3 Results

2. Click **Update Baseline(s)** select that you want to add the rule version **1.2** to a given baseline, as shown in Figure 3.

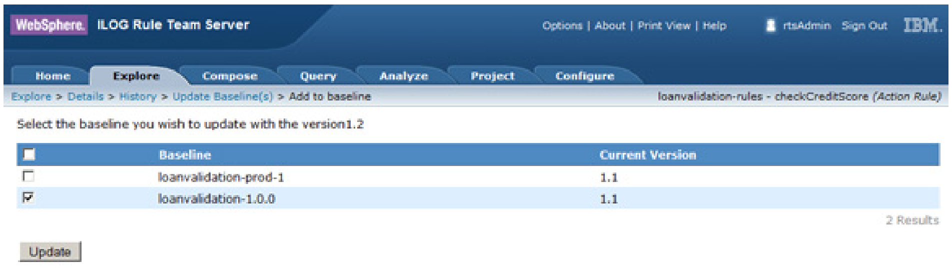
   Figure 3. Add new rule version to a baseline

   
   WebSphere. ILOG Rule Team Server
   Options | About | Print View | Help
   rtsAdmin Sign Out IBM.
   Home Explore Compose Query Analyze Project Configure
   Explore > Details > History > Update Baseline(s)
   loanvalidation-rules - checkCreditScore (Action Rule)

   You selected the element version '1.2'. What would you like to do?
   Add it to a given baseline
   Remove it from a baseline it belongs to

3. Select the **loanvalidation-1.0.0** baseline, as shown in Figure 4.

   Figure 4. Select the baseline to update

   
   WebSphere. ILOG Rule Team Server
   Options | About | Print View | Help
   rtsAdmin Sign Out IBM.
   Home Explore Compose Query Analyze Project Configure
   Explore > Details > History > Update Baseline(s) > Add to baseline
   loanvalidation-rules - checkCreditScore (Action Rule)
   Update

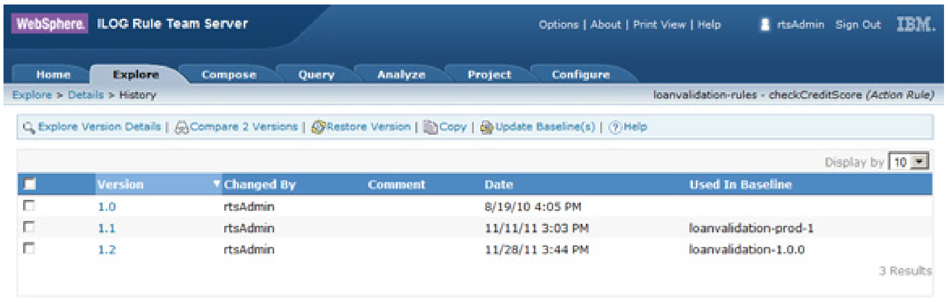
   Select the baseline you wish to update with the version1.2

☐	Baseline	Current Version
☐	loanvalidation-prod-1	1.1
☑	loanvalidation-1.0.0	1.1

   2 Results

4. You now have **loanvalidation-1.0.0** using version **1.2** of the rule instead of the original **1.1**, as you can see in Figure 5.

Figure 5. Updated version


WebSphere. ILOG Rule Team Server
Options | About | Print View | Help
rtsAdmin Sign Out
IBM.
Home
Explore
Compose
Query
Analyze
Project
Configure
Explore > Details > History
Explore Version Details | Compare 2 Versions | Restore Version | Copy | Update Baseline(s) | Help
Display by 10

loanvalidation-rules - checkCreditScore *(Action Rule)*

	Version	Changed By	Comment	Date	Used In Baseline
☐	1.0	rtsAdmin		8/19/10 4:05 PM	
☐	1.1	rtsAdmin		11/11/11 3:03 PM	loanvalidation-prod-1
☐	1.2	rtsAdmin		11/28/11 3:44 PM	loanvalidation-1.0.0

3 Results

By repeating this sequence, you can update a baseline with the desired version of the rules. You can use this approach to perform quick bug fixes on a deployment baseline: for each rule that needs a fix, you restore the rule version from the baseline, edit the rule as needed, then update the baseline again with the edited rule. Once the fix is completed and tested, the baseline can be frozen and redeployed to the QA environment.

This approach is valid for quick, well-defined patches that are applied to a limited number of rules and that mostly involve a simple change of value in the rules. The impact on the ongoing rule authoring is limited because only the rules that need fixing are restored from the baseline.

Restoring development baselines

A main feature of development baselines is that they can be restored, which means that the content of the baseline overwrites the content of the current project state. You can use this feature to work on and edit previous states of the project by doing the following:

1. Create a development baseline that captures the current project state.
2. Restore the desired development baseline and perform the desired updates, unit tests and deployments.
3. When finished, create a new baseline that captures the updated project state.
4. Finally, restore the baseline associated with the current project state, created in step 1.

As with updating baselines, you can use this approach to tackle bug fixing. In this case, there is no unfreezing involved, and since you're using the current state of the project, all the rules are available at once, which facilitates the update process. Therefore, this approach may be better suited for more significant bug fixes.

The major downside is that, while defects are being fixed, no new rule authoring can be performed since the current project state reflects a historical version of the project.

Using RuleStudio synchronization

The synchronization mechanism in Rule Studio is the bridge between the rule project in Eclipse and the one that resides in the RTS repository. During the life cycle of a rule project, the synchronization mechanism is usually used to perform either a full publish (from Rule Studio to

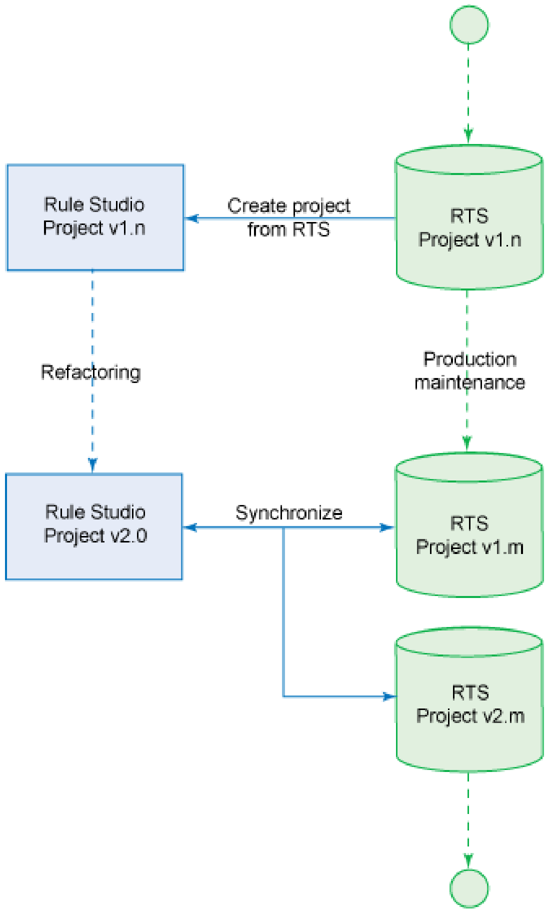
RTS) or a full update (from RTS to Rule Studio) of all the project elements, and will rarely involve any active selection or merging of specific rule elements. The main synchronization scenarios are as follows:

- After a rule project's initial design has been completed in Rule Studio, the project is published to RTS in order to make it available to the business users.
- When the rule project technical elements need to be refactored (for example, BOM update, rule flow change, and so on), the project is created in Rule Studio from RTS, the desired refactoring operations are performed, and finally the project is published back to RTS.

This type of refactoring assumes a quick operation, where all business policy maintenance work can be put on hold while it is completed. Unfortunately, this is not always the case, especially on large and complex rule projects: refactoring can be a lengthy operation that includes the design and test of a new XOM and BOM, and production maintenance has to continue in parallel.

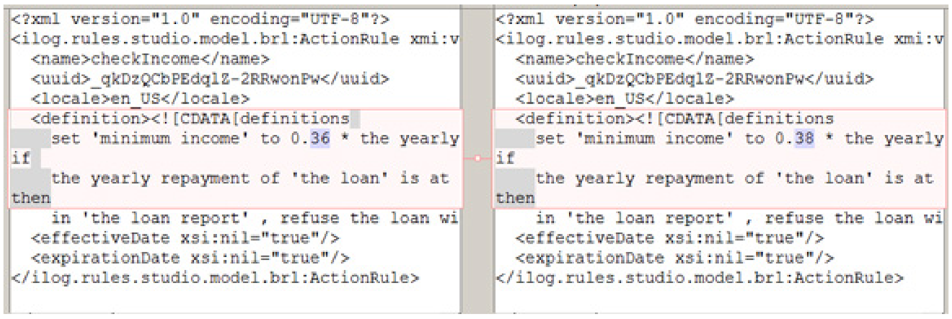
So, instead of stopping maintenance when refactoring is needed, the rule project in Rule Studio can be allowed to live independently from the RTS version up to the point when the refactored project is ready to be deployed, as shown in Figure 6.

Figure 6. Managing rule project versions using Rule Studio



Note that the main issue with this approach is that the comparison operation in Rule Studio is based on the XML serialized form of the rules, as shown in Figure 7, which makes the merging operation delicate and usually requires both an IT and a business resource. To alleviate the burden of the merge, it may be a good practice to perform the synchronization incrementally instead of all at the end.

Figure 7. Comparison of BRL in Rule Studio



```
<?xml version="1.0" encoding="UTF-8"?>
<ilog.rules.studio.model.brl:ActionRule xmi:v
  <name>checkIncome</name>
  <uuid>_qkDzQCbPEdqlZ-2RRwonPw</uuid>
  <locale>en_US</locale>
  <definition><![CDATA[definitions
    set 'minimum income' to 0.36 * the yearly
if
    the yearly repayment of 'the loan' is at
then
    in 'the loan report' , refuse the loan wi
  <effectiveDate xsi:nil="true"/>
  <expirationDate xsi:nil="true"/>
</ilog.rules.studio.model.brl:ActionRule>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ilog.rules.studio.model.brl:ActionRule xmi:v
  <name>checkIncome</name>
  <uuid>_qkDzQCbPEdqlZ-2RRwonPw</uuid>
  <locale>en_US</locale>
  <definition><![CDATA[definitions
    set 'minimum income' to 0.38 * the yearly
if
    the yearly repayment of 'the loan' is at
then
    in 'the loan report' , refuse the loan wi
  <effectiveDate xsi:nil="true"/>
  <expirationDate xsi:nil="true"/>
</ilog.rules.studio.model.brl:ActionRule>
```

Using RTS data sources

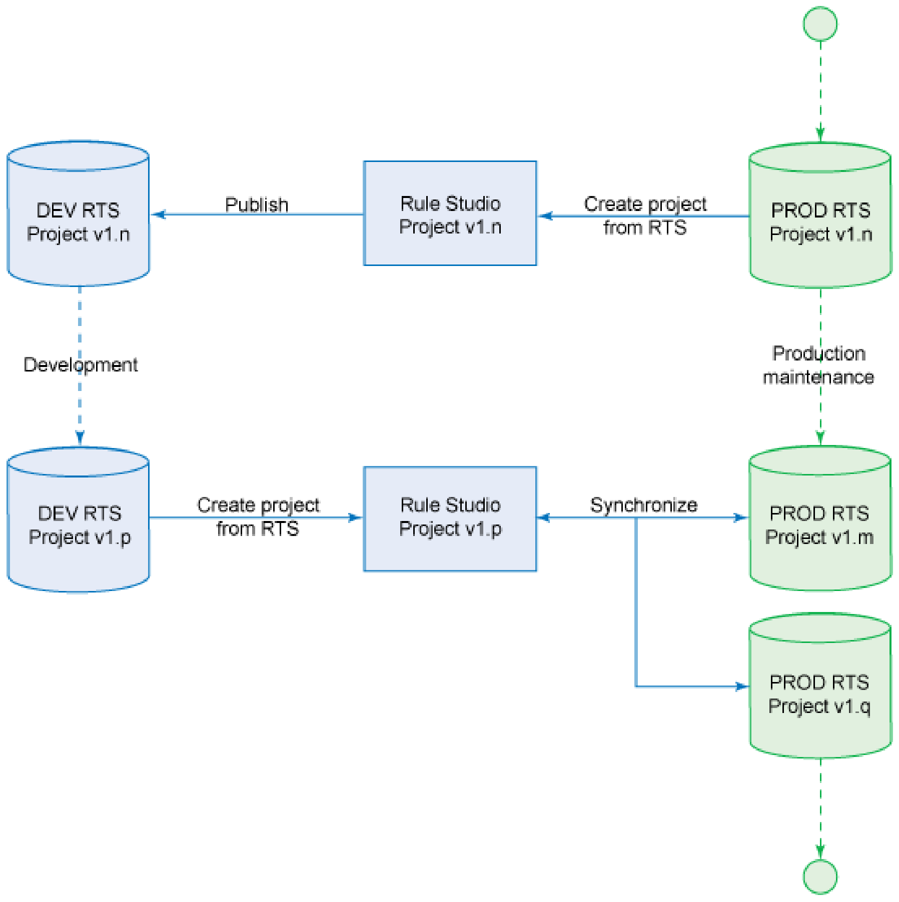
The RTS web application uses a data source to connect to the rule repository database. You can change the data source name (which by default is `jdbc/ilogDataSource`)using a configuration variable, which means that you can easily associate the RTS application with different rule repository instances.

You can thus have two or more copies of the same rule project that can live independently in two separate rule repositories. The relationship between the projects is established and maintained through the UUID of each project element, which will not change throughout their life cycle.

You can use this approach to manage concurrent rule project enhancements implemented by business users. In the example shown in Figure 8, PROD RTS represents the rule repository that is considered as the source of truth. At one point, the business has a need to bring a significant change to their business policy, which will take time to analyze, define and test, and may impact many of the existing rules.

To support this new development, a separate rule repository, DEV RTS, is used and the current state of the rule project is exported to it via Rule Studio synchronization. Once the new development is completed, it is published back to the PROD RTS, again using Rule Studio synchronization. Contrary to the refactoring presented in the previous section, merging of the two projects is usually simpler because there is no underlying change to the project BOM or other technical artifacts.

Figure 8. Leveraging multiple RTS datasources



The downside of this approach to managing concurrent rule project enhancements is that each different project instance needs to be supported by a different RTS database.

Note that if the requirement is only to have multiple copies of the same rule project that will be used as sandboxes by the business users in RTS, but that content will never actually be merged to the production project, you can just create copies of the project in Rule Studio and publish these copies to RTS. The copy operation in Rule Studio will take care of changing the UUIDs of the project elements and, therefore, there will be no element conflict from having multiple copies of the project in RTS.

Managing rule project versions with WebSphere Operational Decision Management V7.5

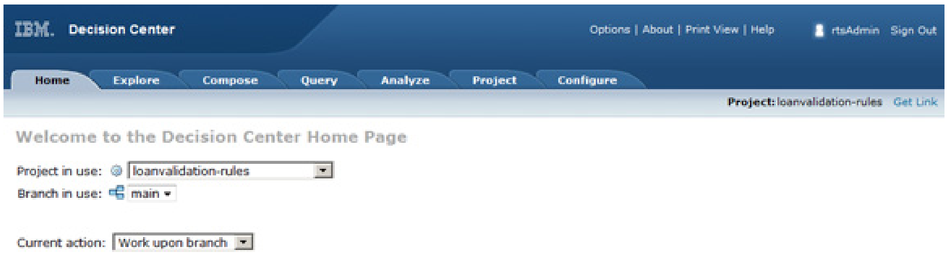
In the previous section, you learned about several strategies to deal with rule project versions, each adapted to a different purpose. However, each one has a downside and entails complex and error-prone sequences of operations, which points to one fundamental issue: RTS does not support the concept of branching on its projects.

WebSphere Operational Decision Management V 7.5 introduces a native branching capability in the Decision Center (ex-RTS) that provides intuitive solutions to the various project versioning requirements.

Project branching in Decision Center

Branches are an integral part of working with the Decision Center. Before starting any work, you need to select the target rule project as well as the branch you want to use in that rule project. Figure 9 shows the home page of the Decision Center where you make this selection.

Figure 9. Decision Center home page



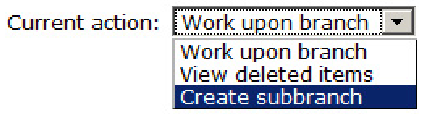
Project: loanvalidation-rules

Welcome to the Decision Center Home Page

Project in use: loanvalidation-rules
Branch in use: main
Current action: Work upon branch

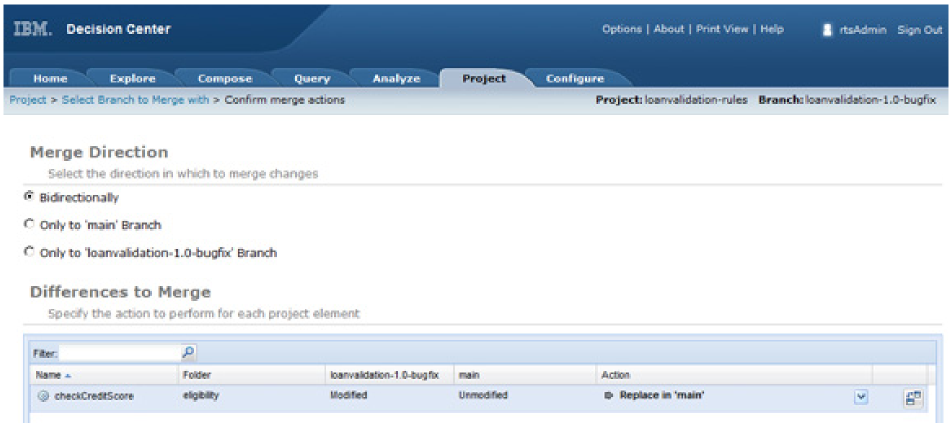
By default, a rule project exists with a single branch, the "main" branch. At any time, you can create a sub-branch from the "branch in use" by selecting the **Create subbranch** action in the **Current action** menu, as shown in Figure 10.

Figure 10. Create a subbranch



When you need to join concurrent project branches, the **Merge Branches** action provides a business user friendly way to perform the merging operation. After designating the branch to perform the merge with, Decision Center displays the set of differences to merge and proposes possible actions to be performed (replace in one branch or in the other, or do nothing), as shown in Figure 11. You can choose the desired direction for the merge in order to limit the set of differences that get displayed.

Figure 11. Merge branches actions



Project > Select Branch to Merge with > Confirm merge actions
Project: loanvalidation-rules **Branch:** loanvalidation-1.0-bugfix

Merge Direction
Select the direction in which to merge changes
[✓ Bidirectionally]
Only to 'main' Branch
Only to 'loanvalidation-1.0-bugfix' Branch

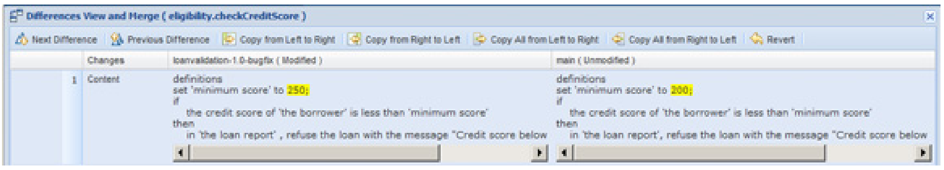
Differences to Merge
Specify the action to perform for each project element

Filter:

Name	Folder	loanvalidation-1.0-bugfix	main	Action
checkCreditScore	eligibility	Modified	Unmodified	Replace in 'main'

A detailed view of the differences, as shown in Figure 12, is provided for each artifact to help with the merge task, and to allow fine grain difference management.

Figure 12. Detail of merge differences


Differences View and Merge (eligibility.checkCreditScore)
Next Difference
Previous Difference
Copy from Left to Right
Copy from Right to Left
Copy All from Left to Right
Copy All from Right to Left
Revert

	Changes	loanvalidation-1.0-bugfix (Modified)	main (Unmodified)
1	Content	definitions set 'minimum score' to 250; if the credit score of 'the borrower' is less than 'minimum score' then in 'the loan report' , refuse the loan with the message "Credit score below	definitions set 'minimum score' to 200; if the credit score of 'the borrower' is less than 'minimum score' then in 'the loan report', refuse the loan with the message "Credit score below

This is about the extent of the operations on branch management in Decision Center. By deliberately limiting the set of operations, branch management is kept as non-technical as possible, so that it can be easily used and understood by business users.

The concept of baselines is still present in Decision Center, but baselines are now read-only snapshots of the project state (a baseline cannot be unfrozen anymore). The role of baselines is to represent:

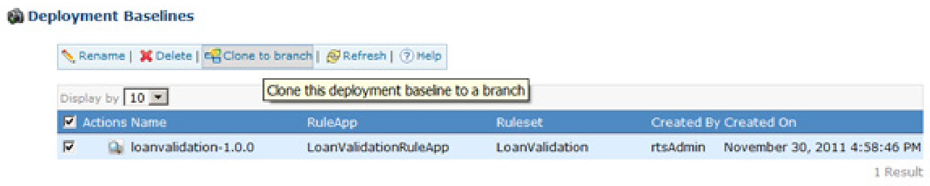
- A designated restore point for development baselines, similar to a tagged version for an SCCS.
- A reliable deployment state that can be used for clean promotion through the different rule environments, from testing to production.

Managing bug fixes

The branching capability of Decision Center makes bug fixes straightforward. Assume that the promotion of a new ruleset version was initiated from the **main** branch by creating a deployment baseline named **loanvalidation-1.0.0**.

1. If a defect is found in the rules while going through QA, the rule authors can simply clone the deployment baseline to a branch as shown in Figure 13, and start using this branch to fix the defect.

   Figure 13. Deployment baseline section from the Manage Subbranches and Baselines page

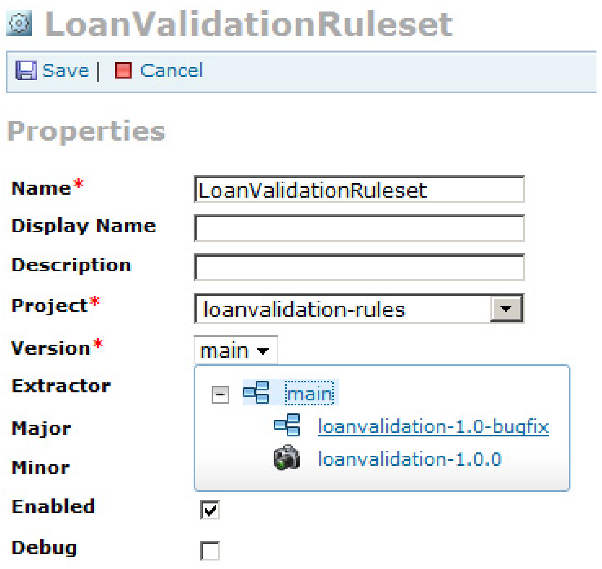
   

✓ Actions	Name	RuleApp	Ruleset	Created By	Created On
✓	loanvalidation-1.0.0	LoanValidationRuleApp	LoanValidation	rtsAdmin	November 30, 2011 4:58:46 PM

   1 Result

2. Once the fix is completed, you need to define a new RuleApp in order to deploy from the new branch instead of **main**. The definition of rulesets in Decision Management V7.5 includes the specification of the project version from which the ruleset should be extracted. As shown in Figure 14, the version can be either a baseline (such as **loanvalidation-1.0.0**) or a branch (such as **loanvalidation-1.0-bugfix**). For our example, we'll select the latter.

Figure 14. Select the project version



LoanValidationRuleset

Save | Cancel

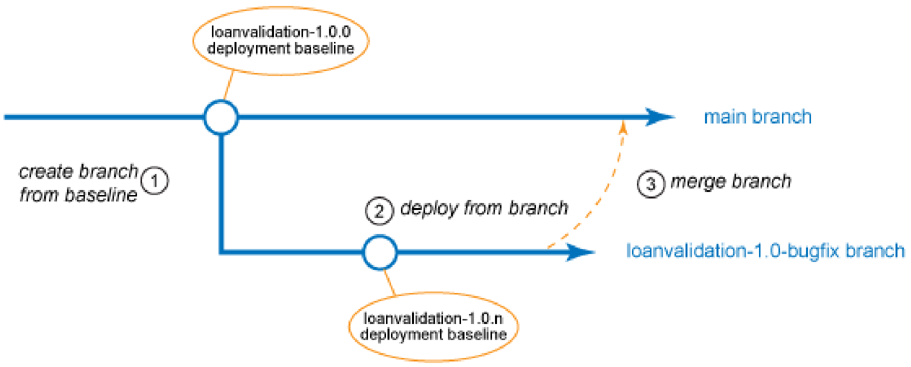
Properties

Name*	LoanValidationRuleset
Display Name	
Description	
Project*	loanvalidation-rules
Version*	main
Extractor	
Major	
Minor	
Enabled	☑
Debug	☐

3. You can make additional fix iterations on the bug fix branch until the version is ready to be released in the production environment. At this point, the last task to complete is to merge the changes from the bug fix branch to the main branch. You can do this with the **Merge Branches** action, previously shown on Figure 11. In this case, we'll select the **Only to 'main' Branch** option for the direction.

Figure 15 illustrates these three steps.

Figure 15. Managing bug fixes process

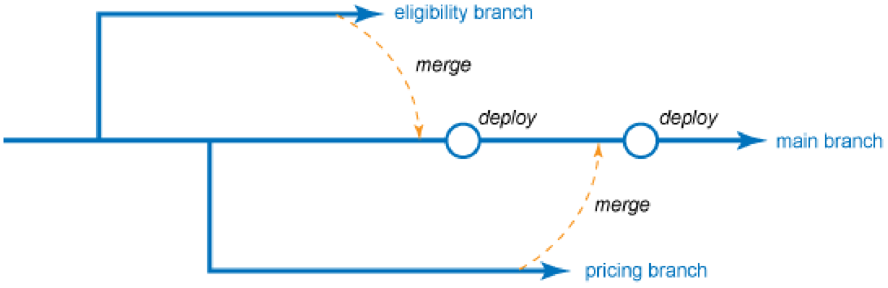


loanvalidation-1.0.0 deployment baseline
main branch
create branch from baseline ①
② deploy from branch
③ merge branch
loanvalidation-1.0-bugfix branch
loanvalidation-1.0.n deployment baseline

Managing concurrent rule service enhancements

Concurrent enhancements to a rule service are also made straightforward through project branches. When a change to the business policy is submitted, you can create a branch to implement and unit-test the change request. When you're ready to deploy the change, you can merge the updated project elements back to the main branch, and a deployment baseline is created as a result of deploying the updated RuleApp.

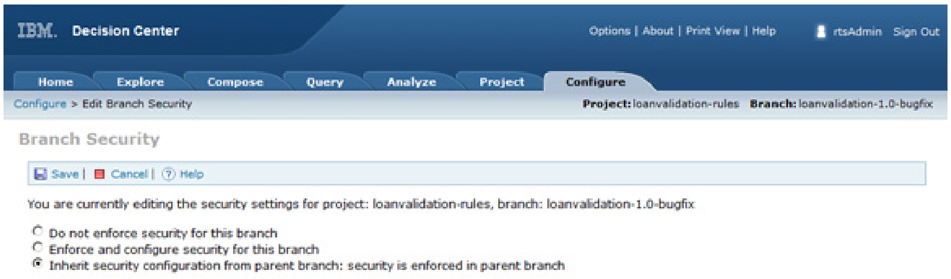
Figure 16. Concurrent rule service enhancement process



The concept of rule project security as proposed in RTS is now managed at the branch level. In large projects that include business policies coming from multiple LOBs or business groups, this allows for enforcement of fine-grain security by dedicating branches to certain groups (for example, **pricing** and **eligibility** in Figure 16).

Figure 17 shows the new branch security options in Decision Center.

Figure 17. Branch security options in Decision Center


IBM. Decision Center
Options | About | Print View | Help
rtsAdmin Sign Out
Home Explore Compose Query Analyze Project Configure
Configure > Edit Branch Security
Branch Security
Save | Cancel | Help

Project: loanvalidation-rules **Branch:** loanvalidation-1.0-bugfix

You are currently editing the security settings for project: loanvalidation-rules, branch: loanvalidation-1.0-bugfix

- ○ Do not enforce security for this branch
- ○ Enforce and configure security for this branch
- [✓ Inherit security configuration from parent branch: security is enforced in parent branch]

Managing rule project refactoring

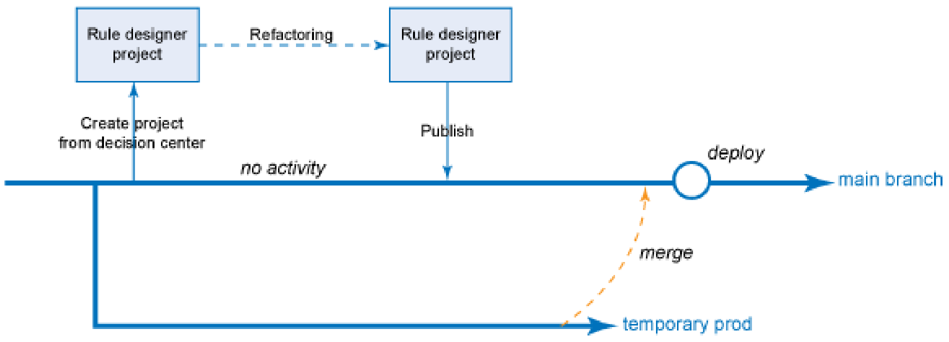
Rule project refactoring, involving changes to the BOM for example, remains a task that is shared between Rule Designer (formerly Rule Studio) and the Decision Center, similar to the refactoring process described for JRules V7.1.

The merge operation in Decision Center does not include the BOM or the ruleset parameters. Updates to these artifacts on a given branch must therefore come from a synchronization operation with Rule Designer.

The refactoring process is accomplished as follows:

1. Create a branch to be the temporary production branch, to support ongoing maintenance.
2. In Rule Designer, create a rule project from the Decision Center project's main branch.
3. Perform the rule project refactoring. Meanwhile, all project maintenance work will be handled through the temporary production branch. There should be no activity on the main branch until refactoring is completed.
4. When the refactoring is complete, synchronize the Rule Designer project with the Decision Center project's main branch, overriding the content of the main branch.
5. Merge the temporary production branch back to the main branch.

Figure 18. Refactoring process in Decision Center



The main benefit in this scenario is that the merging operation is using the rule specific diff instead of the generic textual diff based on the BRL format, as shown previously in Figure 7.

Conclusion

While the concept of branching is usually technical in nature and rooted in the software development culture, the Decision Center component of WebSphere Operational Decision Management V7.5 manages to hide its complexity to business users, while providing them the capabilities they need to achieve maximum agility in managing the life cycle of rule-based decision services.

Resources

- WebSphere ILOG JRules BRMS Information Center
- developerWorks BPM zone
- IBM BPM Journal

About the author

Pierre Berlandier

Pierre Berlandier has worked for ILOG services for the past 15 years, during which time he has helped ILOG's clients leverage the successive incarnations of the rules programming paradigm from expert systems to real-time intelligent agents, and now business rules applications.