Chapter 8
IBM WebSphere ILOG JRules

Target audience

- *All*

In this chapter you will learn

- *The architecture of the JRules BRMS*
- *The concept of operations of the JRules BRMS*
- *An introduction into the various JRules rule artifacts*
- *An introduction into the various JRules modules*

Key points

- *JRules has a modular architecture, providing different components that support different tasks or cater to different users.*
- *Business users and technical users author and manage business rules using different environments, which are adapted to their tasks and skill sets.*
- *JRules provides a domain-specific language for rule authoring that has a modular/layered architecture.*
- *JRules embeds a powerful and easy to use rule unit testing framework.*
- *JRules offers a rich set of functionalities for managing rule execution.*

8.1 Introduction

This chapter presents the IBM WebSphere ILOG JRules Business Rule Management System, or *JRules* in short. JRules offers an important set of components and capabilities to enable business users and developers to manage business rules directly with various levels of implication, from limited review to complete control over the specification, creation, testing, and deployment of business rules.

The description included in this chapter is based on the JRules 7 version of the product. At the time of this writing, JRules is in its 14th year as a commercial product. JRules was initially developed by *ILOG Corp*, a software vendor founded in 1987, and acquired by IBM in 2009. JRules is the third family of rule engine (what they were called back then)/BRMS products, starting with a Lisp implementation, followed by a C++ implementation, in 1992. JRules has gone through a number of major architectural changes around each major version number (JRules 3, around 1999–2000, JRules 4, around 2002–2003, JRules 5, in 2005, JRules 6, in 2006, and JRules 7 in 2009). One of the most significant changes occurred around JRules 6 where the old proprietary ILOG rule IDE[1] was dropped, and its functionalities split between two components, one destined for technical users, packaged as an Eclipse/RAD plug-in, and one destined for business users, packaged as a Web application. It was also starting with JRules 6 that earlier components and libraries for EJB support and rule engine pooling were packaged into a JCA-compliant *Rule Execution Server* (RES). JRules 7 shares the same basic architecture.

Section 8.2 provides a very quick overview of JRules, including the main components, the way they are meant to work together (concept of operations), and an overview of rule artifacts. Section 8.3 talks about *Rule Studio* (RS), which the JRules module aimed at technical users. We provide a brief description of the structure of rule projects, and of the various artifacts that we can create within Rule Studio projects; Chaps. 10 and 11 will go into much greater detail. We talk about *Rule Team Server* (RTS), the Web application aimed at business users, in Sect. 8.4. The *Rule Execution Server* is described in Sect. 8.5; far more detail will be provided in Chap. 13, where we talk about JRules's support for ruleset deployment and execution. We talk about *Rule Solutions for Office* (RSO) in Sect. 8.6, which consist of Microsoft Office's Word™ and Excel™ plug-ins that enable business users to edit if–then rules and *decision tables* in Word documents and Excel spreadsheets, respectively. We provide a summary in Sect. 8.7.

8.2 Business Rule Management System Main Components

The JRules BRMS platform is a collection of modules that operate in different environments while working together to provide a comprehensive Business Rule Management System. BRMS helps to manage business rule independently of the business application.

As mentioned in Chap. 1, a BRMS enables business and IT to collaborate, author, manage, and execute business rules. In addition to working on different timelines, IT and business users need to work with different tools that reflect their different skill sets and views of the application. Figure 8.1 shows the different modules provided by JRules, along with the target users and the need they help to fulfill. In the remainder of this section, we will provide a short overview of the various components.

[1]Integrated Development Environment.

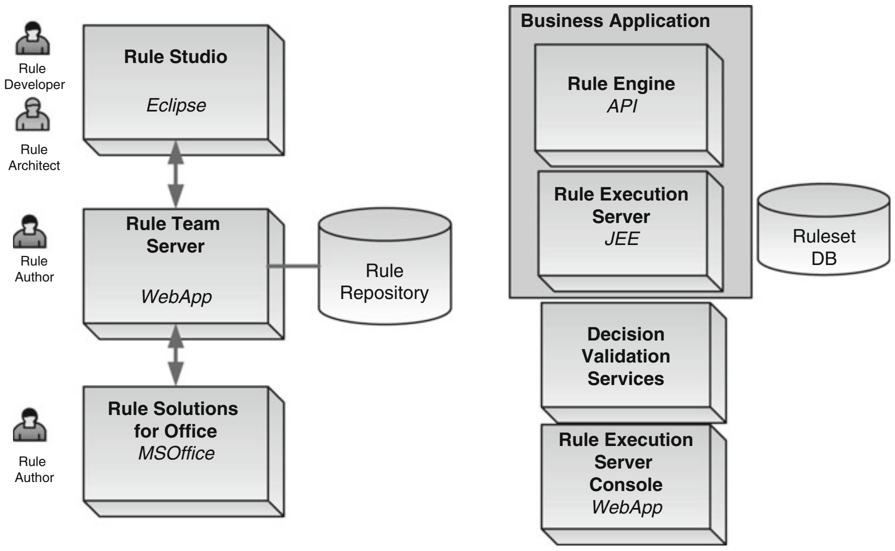


Fig. 8.1 IBM WebSphere ILOG JRules 7.1 Component View

- *Rule Studio (RS)*. Business rule application development starts in Rule Studio, an Eclipse[2]-based Integrated Development Environment (IDE). Working in Rule Studio, developers can set up the rule authoring environment, create business rules and rule templates, author more complex technical rules, and design the execution or rule flow. Rule Studio also provides tools for unit testing and deploying rules to the target execution environment. Developers can store the business rule artifacts using a source code control tool.
- *Rule Team Server (RTS)* is a Web-based application for rule authoring and management, aimed at business users. Business users can work in Rule Team Server both during application development and after the application is deployed to production, as part of the application maintenance (see ABRD cycle "Implementation and Enhancement" in Chap. 3). Rule Team Server stores rule projects in a *rule repository* which is typically persisted in a relational database.
- *Rule Solutions for Office (RSO)*. For business users and rule authors who prefer to work offline, Rule Solutions for Office supports rule authoring in Microsoft Office 2007 products. In particular, we can edit if–then rules in Microsoft Word™ documents and decision tables in Excel™ spreadsheets thanks to JRules plug-ins. We can export rule artifacts from Rule Team Server to Word™/Excel™ files, edit the Word/Excel files, and upload them back into Rule Team Server.
- *Rule Execution Server (RES)*. The rule engine module can be integrated into the core business application using a low level API, or can be deployed as a monitored

[2] www.eclipse.org.

component in a JEE container, the Rule Execution Server (see Sect. 7.4 for a general discussion, and Chap. 13 for JRules – specific deployment options). RES provides management, scalability, security, transaction support, and logging capabilities on top of the rule execution. Using this deployment the business application or more precisely the decision service within the application interacts with the rule engine using rule session API. RES maintains a pool of rule engines and ensures transaction propagation and security control. Coupled with RES is a Web-based application called Rule Execution Server Console, used to monitor and manage the rule sets deployed within RES.

- *Decision Validation Services (DVS)* is a unit testing framework that enables rule testing in Rule Studio and testing and simulation in Rule Team Server. Business users, developers, and QA testers can use DVS to verify the behavior of the rules in a rule set. The user can define *scenarios* in a data source such as Microsoft 2003 Excel spreadsheets. Scenarios can include the specification of expected test results (e.g., variable or object attribute values), as well as an enumeration of the rules or rule tasks that were executed in the process, to verify that the ruleset behaved as expected. Business users can also define key performance indicators (KPIs) to assess the rule set according to specific business metrics. With these capabilities a business user can perform "what–if" analyses, where a reference ("champion") ruleset is compared to the newly updated ruleset (the "challenger") during simulation testing. DVS is presented in Chap. 15.

8.2.1 The Concept of Operations

Once the first level of rule analysis is done, it is possible to quickly create rules using Rule Studio. The Rule Studio environment has all the wizards to develop object models, rule projects, rule flows, and other business rule artifacts. The object (or data) model is in fact built around two layers: the logical layer, called *Business Object Model* (BOM), is used as vocabulary for the rules, and one physical layer, called the *eXecutable Object Model* (XOM) which corresponds to the implementation of the application objects in Java or XML. XOM choices were discussed in Sect. 5.3, and will be discussed further in Chap. 13. BOM design issues were discussed *in general terms* in Sect. 9.2.1 and Sect. 10.3.

Rule Studio is used by rule developers, rule architects, software developers, and business analysts, depending on their technical skills. Note that while business analysts are not meant to be technical, it has been our experience that many business analysts have an IT background. At the very least, they are Excel specialists who can do wonders with a spreadsheet. We have also met many who are IT specialists, who are able to develop a simple database application, or even a simple Web application. For such users, an Eclipse-based environment is not an issue.

In those cases where rule authoring is performed by non-technical business analysts, developers need to *publish* rule projects created from within Rule Studio to Rule Team Server to make it available for business users to begin rule authoring.

This enables developers and business users to collaborate on the same rules while working in their separate environments. After the rule project has evolved in Rule Team Server, developers can synchronize their copy of the rules with those stored in the Rule Team Server rule repository. We recommend using this repository as the reference or master during rule maintenance, and use the synchronization to Rule Studio to work on deeper changes like the updates to object models, to the rule project structure or to the rule flow. An additional synchronization mechanism is available in Rule Team Server that allows business rules to be published as Microsoft Word or Excel documents. After working with the rules in Word™ or Excel™, rule authors can easily synchronize their updates back to the Rule Team Server rule repository.

Within Rule Team Server (RTS), business analysts manage the rule elements by using a set of capabilities to control the version, the configuration and the life cycle. In RTS all users collaborate within a shared workspace and use locking to control access to resources currently being edited. The definition of rule artifacts includes properties (metadata) like the rule *status* property, or the *effective date* to control the rule life cycle. RTS supports versioning of the various rule artifacts. Each time a rule artifact is saved, a separate version is created. RTS supports also the concept of a *baseline* which tags the current versions of all the artifacts within a rule project. Baselines freeze the state of a project at a given moment in time. They are used most often before a rule set deployment or to mark the end of a maintenance cycle.

Once the ruleset reaches a certain level of completeness, it can be deployed to a Rule Execution Server. Figure 8.2 illustrates this process.

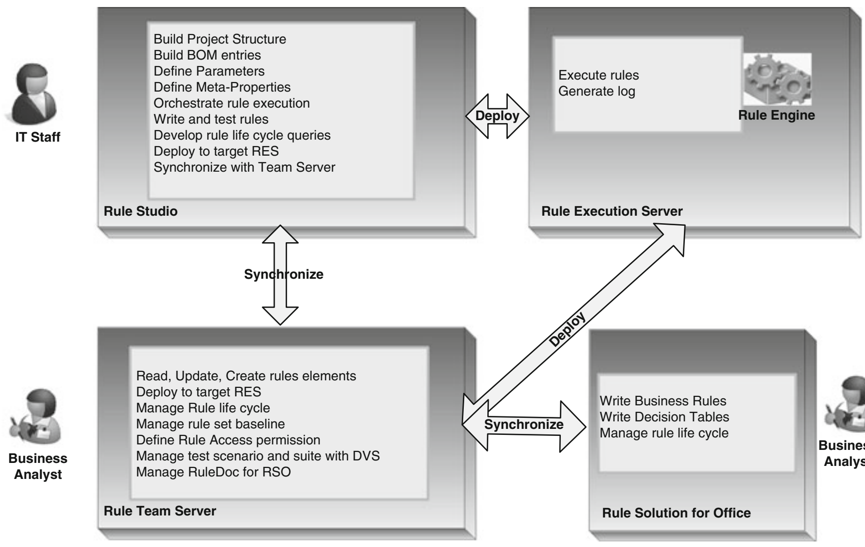


Fig. 8.2 Concept of operations

We detail later in this chapter how we build our claim validation rule projects using Rule Studio, then how rule writer can leverage the features of Rule Team Server to maintain the rule set. Let us start by looking at Rule Studio.

8.2.2 Rule Artifacts

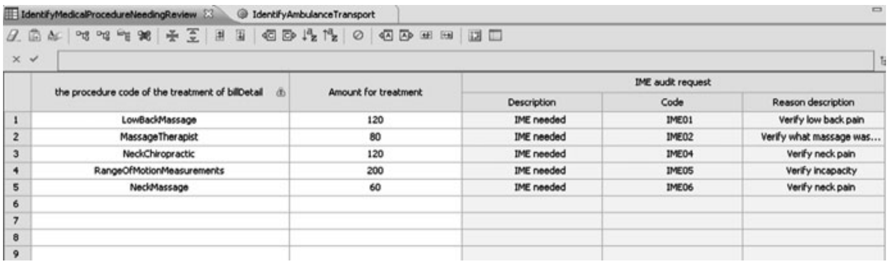
JRules enables us to create different types of rule artifacts, depending on the complexity of the business logic, on the regularity of its structure, and on its specific use. Most business-oriented rule artifacts are based on a business-oriented, natural language-like *Business Action Language* (BAL) . The BAL and the various artifacts that use it are described at length in Sect. 11.2. For the purposes of this section and chapter, we will give a preview of two BAL-based artifacts, *action (if–then) rules,* and *decision tables*. Figure 8.3 shows an example of an *action rule*. Action rules have four parts: *definitions* part, *if* part*, then* part*,* and *else* part. The *Definitions* part is used to define variables local to the current business rule. The conditions of the business rule are listed in the "*if*" part, and the actions to be performed are listed in the *then* and *else* parts. As we later see (Sect. 11.2), all the parts are optional except for the *then* part. The sample rule of Fig. 8.3 has no *else* part. The meaning of this rule should be self-explanatory.

Another very useful format for representing rules is the decision table which presents all the rules with similar conditions and similar actions in a tabular format: columns represent conditions and actions, and each row represents an individual rule. Decision tables provide an efficient representation when the rules need to test ranges of possible values, enumeration values, and numerical attributes (Fig. 8.4). Among other features, the decision table editor also helps identify gaps and overlaps within rule conditions. In the table below, columns with a clear (white) background represent conditions on the medical treatment procedure code, and the amount invoiced for that treatment. Columns with a grey background represent action columns.[3] In this example, there is a single action that creates an audit request to

```
Definitions
set policy to the policy of 'the claim' where the status of this policy is not
closed;
if
  the day of loss of 'the claim' is after the expiration date of policy
then
  add to 'the result' the issue : "claim date error" with a code "R01" and a descrip-
tion : "claim is after expiration date of the policy";
  set 'the claim' has issue;
```

Fig. 8.3 A sample action rule: *definitions* <conditional binding> *if* < conditions> *then* <actions>

[3]The decision table editor enables us to edit the graphical attributes of condition and action columns.


IdentifyMedicalProcedureNeedingReview
IdentifyAmbulanceTransport

	the procedure code of the treatment of billDetail	Amount for treatment	IME audit request		
			Description	Code	Reason description
1	LowBackMassage	120	IME needed	IME01	Verify low back pain
2	MassageTherapist	80	IME needed	IME02	Verify what massage was...
3	NeckChiropractic	120	IME needed	IME04	Verify neck pain
4	RangeOfMotionMeasurements	200	IME needed	IME05	Verify incapacity
5	NeckMassage	60	IME needed	IME06	Verify neck pain
6					
7					
8					
9					

Fig. 8.4 Decision table

evaluate the accuracy for the treatment. The three action subcolumns ("Description", "Code", and "Reason description") correspond to different parameters of the audit request to be created.

The reader may have noticed a warning icon near the header of the first condition column (on procedure code of the treatment). This is warning the user that there are some values from all the enumeration of procedure codes that are not listed/tested in this table. This is a special case of a more general *gap detection* feature that decision table authors can enable or disable. A related feature also detects *overlaps* between the values listed in different rows. Depending on the business logic, this *could* be problematic and may need to be fixed. The decision table editor supports other features, discussed in Sect. 11.2.

Rules can also be expressed as *decision trees* which embody an asymmetric structure using a tree of conditions, with the leaf (bottom) nodes representing the action part. A path from the root/top of the tree to a leaf node represents a complete if–then rule. Decision trees, scorecards, and technical rules will be discussed in more detail in Chap. 11.

8.3 Rule Studio

Figure 8.5 presents the different activities and tasks each user role can execute within Rule Studio (RS) and Rule Team Server (RTS).

Rule Architects use Rule Studio to design the structure of the Rule Project. A rule project is a type of Eclipse project dedicated to the development of business rules. Designing the rule project structure includes:

- The definition of its input/output parameters
- The definition of the different data models to use, namely, the Executable Object Models (XOMs), and the Business Object Models (BOMs)
- The definition of the different queries used to search for rule elements in the current project/workspace
- The creation of the hierarchy of rule packages, and finally
- The creation of the rule flow(s)

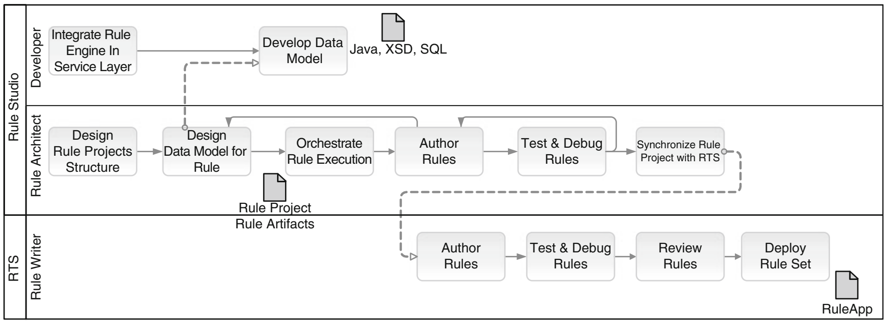

Rule Studio
Developer
Integrate Rule Engine In Service Layer
Develop Data Model
Java, XSD, SQL
Rule Architect
Design Rule Projects Structure
Design Data Model for Rule
Orchestrate Rule Execution
Author Rules
Test & Debug Rules
Synchronize Rule Project with RTS
Rule Project Rule Artifacts
RTS
Rule Writer
Author Rules
Test & Debug Rules
Review Rules
Deploy Rule Set
RuleApp

Fig. 8.5 Rule Studio and RTS rule authoring activities

As mentioned above, the XOM corresponds to the "physical" data model of the application objects manipulated by the rule engine, i.e., either Java classes (objects) or XML schemas (materialized as java objects). The Business Object Model (BOM) embodies the business view of the data, and provides the vocabulary/domain of discourse for writing the business rules, and is constructed as a *view* of the executable object model (XOM). Chapter 9 explores the (different) requirements placed on the XOM and BOM, in general. Chapter 10 explores in depth the BOM to XOM mapping in JRules.

Figure 8.6 shows the main window of Rule Studio. On the left, the Rule Explorer view shows the different projects within the current workspace. The R above a project icon/folder indicates that it is a rule project.

Rule Studio being Eclipse-based, the center view is used to display the different editors. In this case, we have the *rule flow* editor. The view "*Rule Project Map*" at the bottom helps to guide the developer through the various activities needed to create and complete a rule project. The selectable items/links are shortcuts to various rule project actions. A greyed out link (all but "Import XOM" and "Create BOM") represents an action whose prerequisites have not been fulfilled, and reflects dependencies between the various components of a rule project. Some tasks are optional, and a rule developer/architect can take many paths through the project map.

The different rule artifacts are represented by a *definition* which can be edited in the central view, and *properties* or *metadata* used for its management. Section 5.4.3 showed examples of rule properties, and what they may be used for. Rule architects can define custom rule properties through *rule metamodel extensions*, which are defined in a two XML files.

In the remainder of this section, we will give a brief overview of the major tasks performed within Rule Studio; the underlying design *issues* and best practices will be discussed in far more detail in subsequent chapters. Specifically, we will discuss:

- *Designing the rule project structure, in Sect. 8.3.1*. This will be thoroughly discussed in Sect. 9.4, in general terms, and in Sect. 10.2, for the case of JRules.
- *Designing the business rule (meta) model, in Sect. 8.3.2*. Rule properties support many processes, including rule deployment, rule testing, and rule governance.

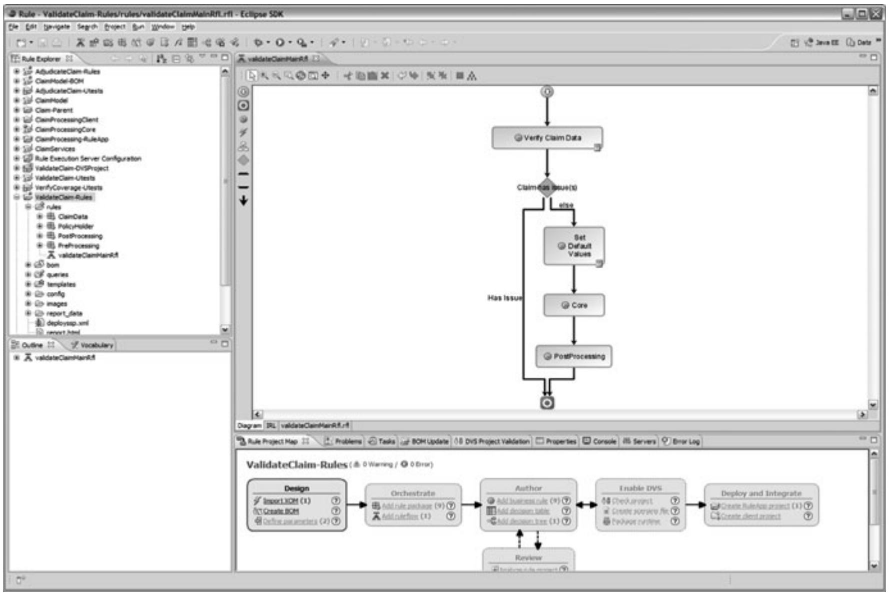


Fig. 8.6 Rule Studio with ruleflow editor

The use of these properties will be discussed in more detail in Chap. 12 (deployment issues), Chap. 14 (testing issues), and Chap. 16 (governance), in general, and for the case of JRules in particular, in Chap. 13 (deployment *with JRules*), Chap. 15 (testing *with JRules*), and Chap. 17 (rule governance *with JRules*).

- *Designing the object models, in Sect. 8.3.3*. This will be discussed in more detail in Sect. 9.2.1, in general, and in Sect. 10.3, for the case of JRules.
- *Orchestrating rule execution, in Sect. 8.3.4*. This will be discussed more thoroughly in Sects. 11.3 (fundamentals) and 11.4 (best practices).
- *Rule testing and deployment, discussed in Sect. 8.3.5*. Chapters 12 and 14 will explore deployment and testing issues, in general, and Chaps. 13 and 15 will discuss deployment and testing within the context of JRules.

8.3.1 Designing the Rule Project Structure

A rule project is a container for rule artifacts, and the artifacts needed to create them, execute them, and debug them. Section 10.2 will explore JRules rule project structure in detail, where we go over the different contents of a project. General, vendor-independent best practices for rule project organization will be discussed in Sect. 9.4. Additional best practices that take advantage of the JRules-specific

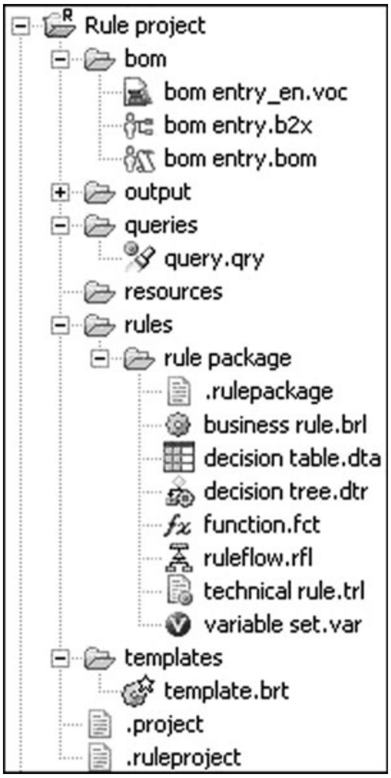


Fig. 8.7 Rule project files and folders

features will be presented in Sect. 10.4.1. For the purposes of this section and chapter, we will provide a brief overview of all of the above.

The various types of rule project elements are illustrated in Fig. 8.7. They include the Business Object Model (BOM), different types of rule artifacts (if–then BAL rules, decision tables, decision trees, technical rules, etc.), artifacts for rule execution orchestration (ruleset parameters, ruleflows), rule queries, and rule templates. Each element is persisted in a single file, or in a combination of files (e.g., the BOM). The elements can thus be version-controlled using a file-based version control software plugged into Eclipse (e.g., Subversion or CVS). BOM design will be briefly introduced in Sect. 8.3.3 and explored in detail in Sect. 10.3. The artifacts for rule execution orchestration will be introduced in Sect. 8.3.4 and explored in detail in Sects. 11.3 and 11.4. In the remainder of this section, we will talk briefly about the organization of rule artifacts with a rule project, give an example of rule project organization, and talk briefly about rule queries and templates.

Within a rule project, rules are organized within a hierarchy of packages. The package hierarchy is typically designed to reflect the structure of the business domain (e.g., product family, business process structure) and to accommodate the execution logic (e.g., a simple mapping to a ruleflow). As a good practice, it is recommended that rule packages include rule artifacts only at the leaf level. This has several advantages, including understandability, an easy mapping to the execution structure (rule flow, more on this in Sect. 11.3), and even more responsiveness of Rule Studio during rule authoring.[4]

Templates enable us to define fill-in-the-blank rule artifacts. Indeed, it is possible to define *rule templates* and *decision table templates* which are used to freeze some parts of a rule, or decision table, and leaving only a few prompts (or cell values) open for input/modification. Templates help enforce rule structure and rule consistency. They come in handy in those cases where the people who are tasked with rule entry and maintenance, are either non-IT savvy, or have a partial view/incomplete context of the rules.

Another important element of a rule project is the *rule query*. JRules supports a rich querying facility that uses a language similar to the *Business Action Language* (called *Business Query Language*) that enables us to search on rule *properties*, rule *definitions*, and rule *semantics*. The following query identifies the *business rules* (i.e., all kinds of rules, except *technical* rules) that have status "*Deployable*." This query can be used as part of a *ruleset extraction and deployment* process (see Sect. 13.5.2, and Chap. 17).

```
Find all business rules
such that the status of each business rule is Deployable
```

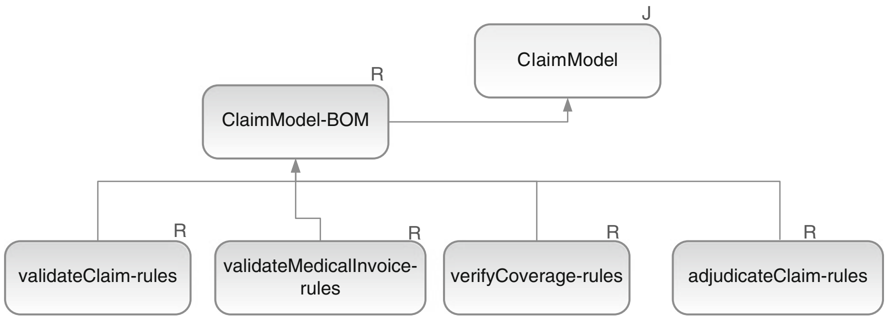
The following query shows an example of a query on the *definition* of rules: it looks for rules that refer to a specific BOM class; rule R "is using" BOM class C, if the rule reads (definitions part, conditions part, or action parts), or modifies an object of type C (action parts). This type of query could be part of some *impact analysis*, e.g., to assess the impact of refactoring a class; more on BOM and XOM *refactoring* in Sect. 10.3.4.

```
Find all business rules
       such that each business rule is using the BOM class
"abrd.claim.injury.Treatment"
```

As for the rule project organization, as discussed in Chap. 5, we recommend mapping one decision point to a rule set and one rule set to a rule project. Further, while a rule project is supposed to contain both *rules* and the BOM needed to write them, we recommend separating the definition of the BOM from the rules that use it so that different rule projects can refer to the same BOM. The project hierarchy below shows what the project structure might look like for our case study. The ClaimModel-BOM rule project includes the definition of the

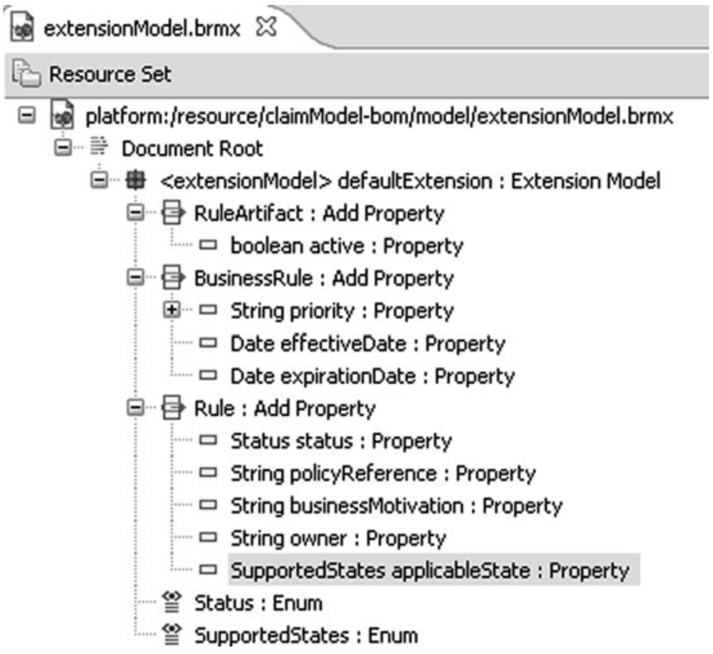
[4]If the tool is configured to perform on-demand loading, this will reduce the number of rules uploaded into the developer's workspace as they navigate the project structure.

BOM used by the rules, and it depends on the “ClaimModel” Java project (XOM). The “validateClaim-rules”, “validateMedicalInvoice-rules”, “verify-Coverage-rules”, and “adjudicateClaim-rules” rule projects contain the rules needed for the tasks “validateClaim”, “validateMedicalInvoice”, “verifyCoverage”, and “adjudicateClaim”, respectively. Sections 10.2 and 10.4 will go over the design issues and rationale in more detail.



8.3.2 *Designing the Business Rule Model*

Rule artifacts have *definitions* (their contents), and a bunch of *properties* (metadata) attached to them. These properties can be used for rule management (mostly) but also for rule execution. The set of properties associated with rule artifacts constitute the *business rule model* – sometimes also referred to as *rule metamodel*. Out of the box, rules artifacts come with a set of predefined properties. Rule architects can *add* more properties to the default rule (meta) model. In Sect. 5.4.2 (prototyping), we presented a list of commonly useful rule properties, including properties that trace rules back to their business motivation, or that restrict their applicability (e.g., jurisdiction, effectiveness period, etc.). Rule Studio supports the *edition* of the *business rule model* using two XML files, a *business rule model extension* file (*.brmx extension), which contains the definition of the new properties (name, type, initial values, behavior upon copy, etc.), and a *data extension file* (*.brdx), which is used to provide property values, for those properties that have an enumerated set of values. Figure 8.8 below shows the Rule Studio wizard for editing the rule model extension file. The figure shows that we are adding properties to three different categories (classes), **RuleArtifact**, **Rule**, and **BusinessRule**. Properties added to **RuleArtifact** will also be added to **Rule**, which is a kind of **RuleArtifact** (the other kind being a **Function**, see Chaps. 10 and 11), and to **BusinessRule**, which is a kind of **Rule** – the other kind being **TechnicalRule**. In turn, the (metamodel) class **BusinessRule** groups ***action*** **(if–then)** rules, decision tables, and decision trees (see Sects. 8.2.2, and 11.2). The reader may also notice that many of the properties



```
extensionModel.brmx
Resource Set
platform:/resource/claimModel-bom/model/extensionModel.brmx
  Document Root
    <extensionModel> defaultExtension : Extension Model
      RuleArtifact : Add Property
        boolean active : Property
      BusinessRule : Add Property
        String priority : Property
        Date effectiveDate : Property
        Date expirationDate : Property
      Rule : Add Property
        Status status : Property
        String policyReference : Property
        String businessMotivation : Property
        String owner : Property
        SupportedStates applicableState : Property
      Status : Enum
      SupportedStates : Enum
```

Fig. 8.8 Rule model extension

have predefined Java types (boolean, java.lang.String, java.util.Date), whereas the “status” property has type **Status** and the “applicableState” property (as in United *States* of America) has type **SupportedStates**. These two types are enumerations, and their values are defined in the *.brdx file – not shown here.

We recommend defining such properties during the prototyping phase before we embark on wholesale rule authoring, because adding property values later can be anywhere from tedious to problematic.[5] Also, it is during the prototyping phase that we start thinking of the rule lifecycle, and of the properties needed to manage it. Finally, we should take advantage of the intense and close communication between business and IT during the prototyping phase to identify and address issues in a timely fashion. Beware of the proliferation of properties, however: too many properties for rule management mean an overly complex/over-engineered rule lifecycle, and too many properties to control rule execution mean poorly contextualized rules, and brittle rule orchestration.

[5]For example, a property such as “author” needs to be initialized at rule creation time, and must not be modifiable. Adding it after a rule has been created can create headaches.

8.3.3 Designing the Business Object Model

Recall that the Business Object Model (BOM) provides a business view of the application object model specifically designed to write the application's business rules. The high-level language (Business Action Language) used to write the rule uses a verbalized view of the BOM, called the business *vocabulary*. The BOM is actually made of three layers that stand between the rules and the executable object model (XOM) – for example, a set of Java classes:

- The BOM data model or interface, which is the middle layer, is stored in a file with *.bom extension, contains the definition of BOM classes, with their public attributes and functions, in a Java-like syntax.
- The vocabulary, which is a verbalization of the elements of the BOM data model, puts a natural language-like coating on top of the BOM data model that stands between the BOM data model and the rules. The vocabulary is built so that rules can refer to a **MedicalInvoice** object as "a medical invoice" and to the attribute dateOfCreation of a **Claim** as "the date of creation of the claim." The vocabulary is stored in a file with extension *.voc, and can be *localized*, i.e., we can have different vocabularies associated with the same BOM data model.
- The BOM to XOM mapping (stored in a *.b2x) file, shows how elements of the BOM data model map to the underlying XOM/Java classes.

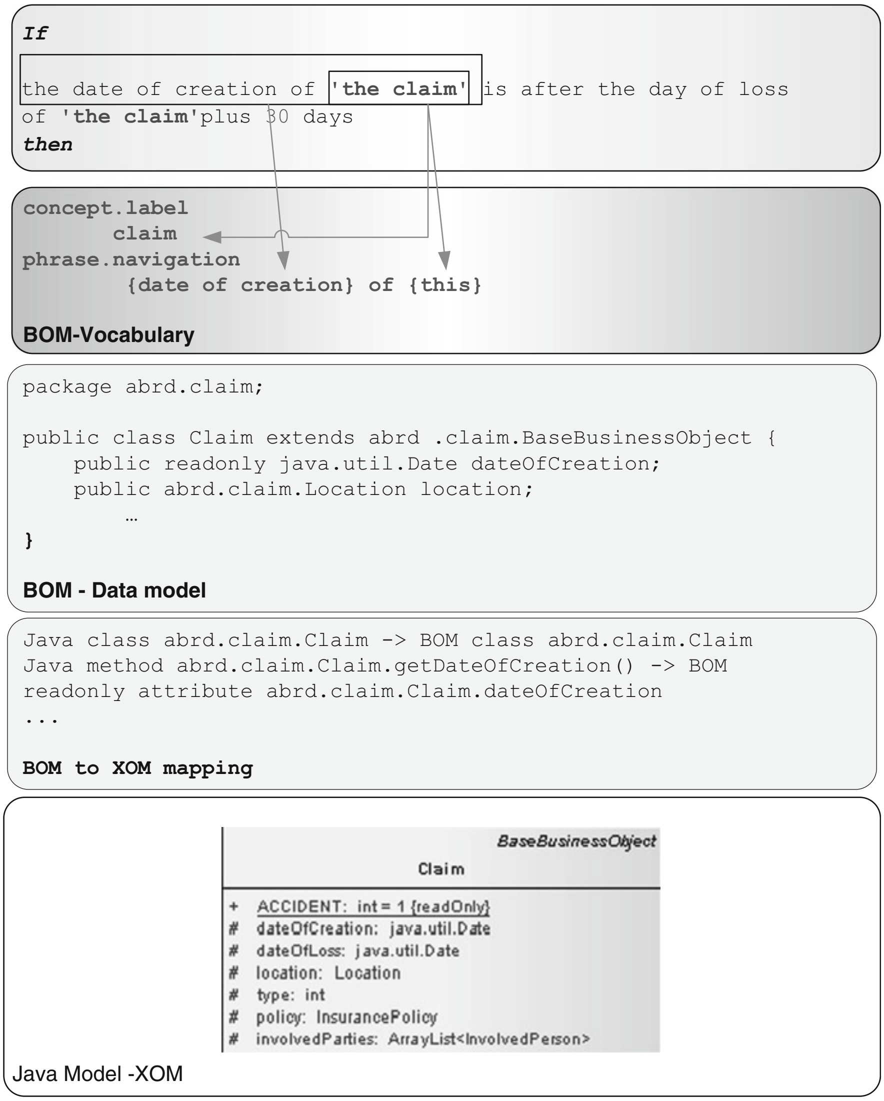
Figure 8.9 *illustrates* the three components of the BOM and their relationships to rules (on top) and to the Java classes/XOM (bottom). Truth be told, only the representation of the BOM data model is accurate. For presentation purposes, we simplified the representation of the vocabulary and of the BOM to XOM mapping to illustrate the concept. In fact, a BOM to XOM mapping such as the one represented in Fig. 8.9 is assumed by default, and *not* explicitly stored; (much) more on this in Sect. 10.3.

Rule Studio provides a BOM editor, which enables us to edit the three components (BOM model, vocabulary, and BOM to XOM mapping) in a unified and synchronized fashion. Figure 8.10 shows a partial view of the wizard (we do not see the BOM to XOM editing prompts). A thorough explanation is provided in Sect. 10.3.

There are two ways to build a BOM and to link it to a XOM:

- A *bottom-up* approach, where we start with the XOM (a Java project or a Java jar file), and build a *default* BOM from it, using a default BOM to XOM mapping. Roughly speaking, the default BOM to XOM mapping generates one BOM class for each *public* XOM class (Java class or XSD complex type), and maps all of the *public* members of the XOM class to corresponding members of the BOM class.[6] This mapping is also pretty good at coming up with reasonable verbalizations,

[6]With the exception of getters and setters, which are mapped by default to attributes that are read only (only getter present), write only (only setter present) or read/write (both accessors present).



```
If
the date of creation of 'the claim' is after the day of loss
of 'the claim'plus 30 days
then
```

```
concept.label
       claim
phrase.navigation
        {date of creation} of {this}
```

BOM-Vocabulary

```
package abrd.claim;

public class Claim extends abrd .claim.BaseBusinessObject {
    public readonly java.util.Date dateOfCreation;
    public abrd.claim.Location location;
        …
}
```

BOM - Data model

```
Java class abrd.claim.Claim -> BOM class abrd.claim.Claim
Java method abrd.claim.Claim.getDateOfCreation() -> BOM
readonly attribute abrd.claim.Claim.dateOfCreation
...
```

BOM to XOM mapping

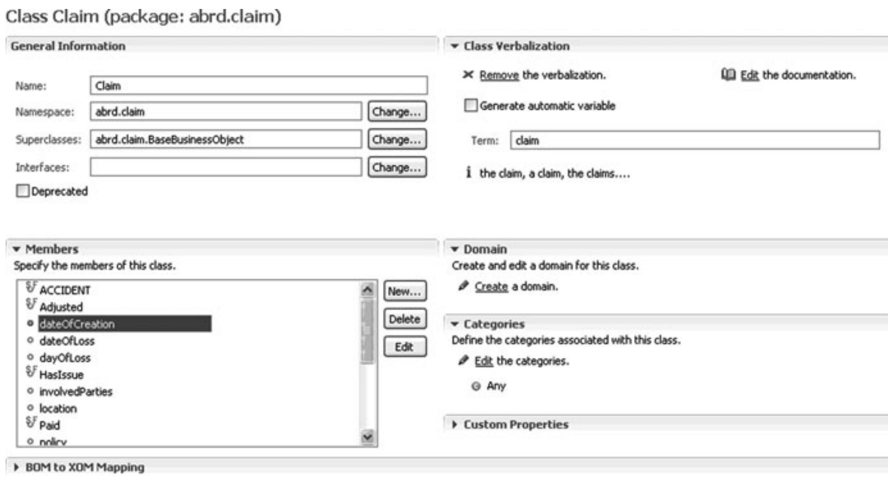
```
BaseBusinessObject
Claim
+ ACCIDENT: int = 1 {readOnly}
# dateOfCreation: java.util.Date
# dateOfLoss: java.util.Date
# location: Location
# type: int
# policy: InsurancePolicy
# involvedParties: ArrayList<InvolvedPerson>
```

Java Model -XOM

Fig. 8.9 The BOM is a three-layer structure that bridges natural language-like (BAL) rules to Java classes (XOM)

provided that developers have followed standard coding/naming practices on the Java (or XSD) side. The so-generated BOM can later be edited to modify the defaults or to add new elements.

- A *top-down* approach, whereby the rule developer constructs the BOM class by class using the BOM editor, adding data members, function members, and the like, but with no corresponding XOM. Such a BOM can be used to author rules,


Class Claim (package: abrd.claim)

General Information

Name: Claim
Namespace: abrd.claim Change...
Superclasses: abrd.claim.BaseBusinessObject Change...
Interfaces: Change...
☐ Deprecated

Class Verbalization

Remove the verbalization. Edit the documentation.
☐ Generate automatic variable
Term: claim
the claim, a claim, the claims....

Members
Specify the members of this class.

ACCIDENT
Adjusted
dateOfCreation
dateOfLoss
dayOfLoss
HasIssue
involvedParties
location
Paid
policy

New... Delete Edit

Domain
Create and edit a domain for this class.
Create a domain.

Categories
Define the categories associated with this class.
Edit the categories.
Any

Custom Properties

BOM to XOM Mapping

Fig. 8.10 BOM editor

but naturally, not to execute them. Once we have a XOM that we can hook up to, we can associate it with the BOM, and synchronize the two.

The bottom-up approach is the more common of the two. Indeed, in most of the projects we were involved with, the business rules approach is introduced as part of a re-engineering effort – in which case the XOM already exists. Even with new projects, it is often the case that by the time we have rules we can code, the XOM will have already been built. However, we have used the top-down approach in a number of projects where hesitant managers wanted a proof of concept/to see what rules would look like, before embarking on business rules. In general we prefer build the XOM from the conceptual data model representing the business entities and their relationships in scope for the rule expression. The XOM is built only for the rules component, in which case, the BOM *was* in some ways, the requirements for the XOM, and needed to be built first. Whichever model gets built first, both will evolve, and Rule Studio provides functionality for keeping them in sync (see Sect. 10.3).

BOM design is a critical activity in business rule development. A well-designed BOM results into an intuitive, unambiguous, and easy to use business rule vocabulary. This helps make rule authoring, rule reviewing, and rule maintenance much easier and much less error-prone. A complex BOM, one that exposes all the complexity and relationships of an enterprise model, will make rule authoring difficult and error prone. Similarly, a BOM that mirrors too closely the idiosyncrasies of the corresponding XOM will result in awkward and hard to understand rules. Chapter 10 will go into the details of BOM design, and BOM to XOM mapping and will present best practices for both.

8.3.4 *Orchestrate Rule Execution*

As explained in Sect. 5.4.3, whereas the rule project structure is concerned with the development time organization of rules, *rule orchestration* is concerned with the run-time execution sequence of rules. Also, while a rule engine (and the production system paradigm) can deal with a "flat" ruleset that is a "bag of rules," the decision embodied in a ruleset can often be broken into a set of more elementary, and stable sub-decisions. This is embodied in a *ruleflow*. Simply put, a ruleflow organizes rule execution in terms of a flow of *rule tasks*, with transitions between them. The transitions (flow links) can be *conditional* on some boolean expression being true. Figure 8.6 showed what the rule editor looks like. Figure 8.11 shows a basic example of a ruleflow for the data validation rule set for the Claim processing application. The ruleset execution starts a task that verifies claim data. If the claim has an issue, the processing terminates. Else, we go through three steps: (1) completing data values, (2) performing the core validation rules, and (3) performing some post-processing (e.g., preparing a validation report).

The ruleflow editor enables us to design the task flow, and to specify the task *bodies,* i.e., specify which rules execute in each task. The recommended practice is to assign one rule package to a rule task. This has several advantages, including: (1) providing execution context for the rules being authored, and (2) simplifying rule-flow maintenance. This also has an impact on the structuring of rules within packages: the mapping to executable rule tasks add one more dimension that we need to consider when we design the package structure. Section 9.4 will go into rule package organization principles and drivers. Section 11.3 will go into a far more detailed discussion of rule execution orchestration.

8.3.5 *Ruleset Testing and Deployment*

Continuing our process of Fig. 8.5, once the rule development infrastructure (rule projects, business rule model, BOM, and orchestration) is completed, we can start entering rules and unit-testing them. Rule testing can be performed using either Junit, or the *Decision Validation Services* (DVS) component of JRules. Chapter 14 will explore testing issues in general. Chapter 15 will explore testing functionalities of JRules, including DVS.

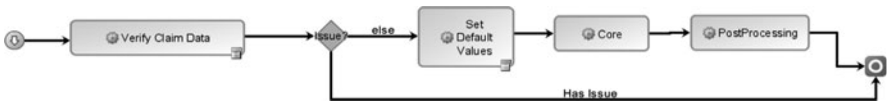


Fig. 8.11 Example of ruleflow

The last recurring activity with Rule Studio is the deployment of the rule sets to the target execution environment. Rule Studio enables us to package/extract rulesets, and deploy them. JRules supports two execution patterns for rulesets, and Rule Studio offers functionalities for both:

- *The embedded execution pattern using the rule engine API*. this is the case where the business application manages the rule engine object on its own, from creation, to population with a ruleset, to invocation, to disposal. For this execution pattern, we need to generate a *ruleset archive* by applying a *ruleset extractor* that builds the ruleset archive from the contents of a rule project.[7] The default extractor grabs all of the rules of the project, but we can develop custom extractors that use rule queries to filter which rules to include in the ruleset archive. For example, we can extract only those rules that have status *deployable*.
- *The decision service execution pattern*. In this case, rulesets are bundled within *RuleApps* and deployed to a *Rule Execution Server* that acts as a central rule execution service for various parts of an application (different decision points in a use case, different tasks of a workflow, or different activities of a BPEL process) or various applications. Rule Studio supports a number of project templates for (1) specifying ruleset bundles/RuleApps (which rulesets to include, and for each ruleset, which project and which extractor), and (2) for specifying Rule Execution Server configurations (host application server, URL, admin credentials, etc.). RuleApps/ruleset bundles can be created and deployed directly, using a live connection, to a rule execution server.

Chapter 13 will go into the details of rule deployment and execution functionalities of JRules.

8.4 Rule Team Server

Rule Team Server (RTS) is a Web-based rule management application that provides a collaborative environment for authoring, managing, validating, and deploying business rules. This is the workspace for business users with an intuitive point-and-click interface that helps support the major use cases for business rule management. Figure 8.12 shows the various activities that different user roles can perform within Rule Team Server. We will provide a brief overview of the underlying functionalities in this chapter. More details will be provided in subsequent chapters. Rule project synchronization with Rule Studio will be presented in Sect. 10.2.3. Access control and permission management within Rule Team Server will be discussed in Sect. 10.2.4 and Chap. 17. Deployment functionalities will be discussed in Chap. 13. Testing functionality will be discussed in Chap. 15. Governance functionality will be discussed in Chap. 17.

[7]And the rules projects it depends on ... more on this in Chaps. 10 and 13.

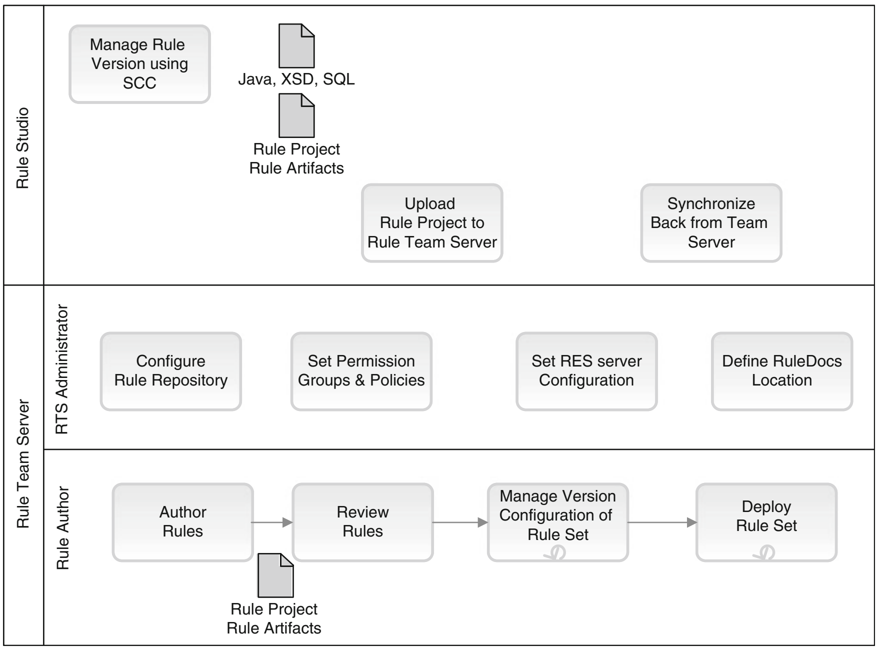
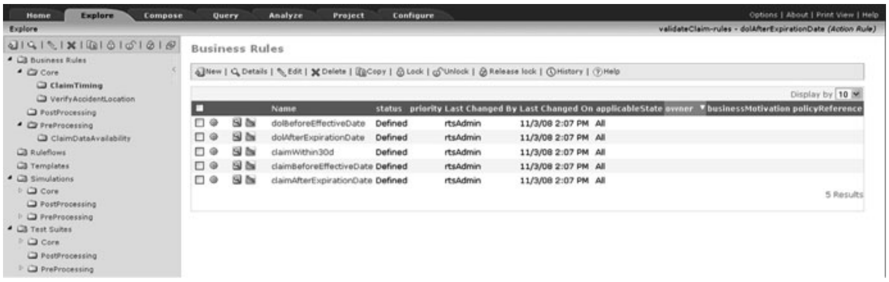

Rule Studio
Manage Rule Version using SCC
Java, XSD, SQL
Rule Project Rule Artifacts
Upload Rule Project to Rule Team Server
Synchronize Back from Team Server
Rule Team Server
RTS Administrator
Configure Rule Repository
Set Permission Groups & Policies
Set RES server Configuration
Define RuleDocs Location
Rule Author
Author Rules
Review Rules
Manage Version Configuration of Rule Set
Deploy Rule Set
Rule Project Rule Artifacts

Fig. 8.12 Rule Team Server activities

Each installation of Rule Team Server manages a *rule repository*, consisting of a bunch of rule projects persisted in a relational database. Recall from Sect. 8.2.1 that rule projects are first “born” in Rule Studio and are brought into Rule Team Server through Rule Studio’s publication and synchronization functionality. When we populate Rule Team Server with Rule Studio projects, it is important to respect project dependencies and publish a project before publishing the projects that refer to it. As mentioned in Sect. 8.2.1, if rule authoring is to be done by non-technical users who use Rule Team Server, its repository should be considered as the copy of record. In this case, developers should regularly synchronize their Rule Studio projects with Rule Team Server to update *their local copy*.

Upon logging into RTS, the “Home” tab presents the user with the list of projects available in the repository. The user can select a project, and explore its contents in the “Explore” tab. Figure 8.13 shows a example of the Explore tab for the “validate-Claim-rules” project. The top-level folders of the project show “Business Rules”, with the package hierarchy underneath, “Ruleflows”, “Templates”, “Simulations”, and “Test Suites”, the latter two with a folder hierarchy that mirrors the rule package hierarchy.[8] The central view of the “Explore” tab shows the list of rules in the “**ClaimTiming**” package, under the “**Core**” package.

[8]Note that this view of a rule project is customizable and is referred to as *smart view*. RTS users can create *smart views* to present project elements in any order they want.


Home
Explore
Compose
Query
Analyze
Project
Configure
Options | About | Print View | Help
Explore
New | Details | Edit | Delete | Copy | Lock | Unlock | Release lock | History | Help
Display by 10

validateClaim-rules - dolAfterExpirationDate (Action Rule)

- Business Rules
  - Core
    - ClaimTiming
    - VerifyAccidentLocation
  - PostProcessing
  - PreProcessing
    - ClaimDataAvailability
- Ruleflows
- Templates
- Simulations
  - Core
  - PostProcessing
  - PreProcessing
- Test Suites
  - Core
  - PostProcessing
  - PreProcessing

Business Rules

Name	status	priority	Last Changed By	Last Changed On	applicableState	owner	businessMotivation	policyReference
dolBeforeEffectiveDate	Defined		rtsAdmin	11/3/08 2:07 PM	All			
dolAfterExpirationDate	Defined		rtsAdmin	11/3/08 2:07 PM	All			
claimWithin30d	Defined		rtsAdmin	11/3/08 2:07 PM	All			
claimBeforeEffectiveDate	Defined		rtsAdmin	11/3/08 2:07 PM	All			
claimAfterExpirationDate	Defined		rtsAdmin	11/3/08 2:07 PM	All			

5 Results

Fig. 8.13 Rule Team Server Explore Tab to navigate a rule project

To view or edit the contents of a rule, the user can select the rule in a table such as the one shown in the central view of Fig. 8.13, and then select the desired action in the tool bar. Alternatively, the rule table includes iconic shortcuts to viewing (magnifying glass) and editing a rule (pencil), left of the rule name in Fig. 8.13. Note that RTS supports user and role-based access control/permission management. By default, all users/roles can view and update project elements within the projects of the repositing. When we activate permission management for a given project, we can define user and role-based permissions to different users, or user roles, to create, view, edit, and delete project elements. More on this in Sect. 10.2.4.

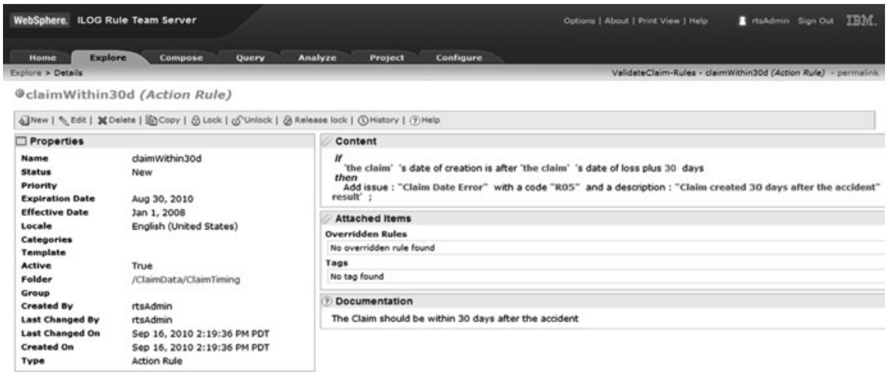
It is important to note that some project elements cannot be edited within RTS, including *rule flows*, and the BOM, and for slightly different reasons:

- *Safety*. Both the BOM and ruleflows represent central infrastructure elements, and we should not make them modifiable by users who may not have the skill or authority to modify them.
- *Partial information*. The RTS manages a *partial* representation of the BOM.[9] Accordingly, we are not able to assess the impact of BOM changes within RTS, or to propagate them. Hence, any BOM changes need to take place within Rule Studio, where they can be synchronized with the corresponding XOM, and propagated to the vocabulary – and the rules.

This is one example of a situation where developers need to synchronize their Rule Studio project versions with those in RTS, make the needed changes and refactorings in Rule Studio, and then publish the modified project(s) back to RTS.

Figure 8.14 shows the detailed view of a rule ("claimWithin30d"), with its properties/metadata shown on the left, and its "**Content**" (definition), "**Tags**", and "**Documentation**" shown on the right. The JRules 7 RTS offers two rule editors: a single-form editor which edits rule contents and change documentation in a single form with a "save" and "cancel" button, and a six-form wizard which enables us to change everything about a rule (including metadata/properties, and versioning policy). Note that each time the contents or the properties of a rule

[9]The BOM to XOM mappings needs both the BOM and . . . the XOM! Which is not available in RTS.


WebSphere. ILOG Rule Team Server
Options | About | Print View | Help
rtsAdmin Sign Out
Home
Explore
Compose
Query
Analyze
Project
Configure
Explore > Details
ValidateClaim-Rules - claimWithin30d (Action Rule) - permalink
New | Edit | Delete | Copy | Lock | Unlock | Release lock | History | Help

claimWithin30d *(Action Rule)*

Properties

Name	claimWithin30d
Status	New
Priority	
Expiration Date	Aug 30, 2010
Effective Date	Jan 1, 2008
Locale	English (United States)
Categories	
Template	
Active	True
Folder	/ClaimData/ClaimTiming
Group	
Created By	rtsAdmin
Last Changed By	rtsAdmin
Last Changed On	Sep 16, 2010 2:19:36 PM PDT
Created On	Sep 16, 2010 2:19:36 PM PDT
Type	Action Rule

Content

```
if
   'the claim'  's date of creation is after 'the claim'  's date of loss plus 30  days
then
   Add issue : "Claim Date Error"  with a code "R05"  and a description : "Claim created 30 days after the accident"
result' ;
```

Attached Items

Overridden Rules

No overridden rule found

Tags

No tag found

Documentation

The Claim should be within 30 days after the accident

Fig. 8.14 Rule Team Server rule details view

artifact are edited and saved, a new version of the artifact is created, according to the versioning policy; the default policy increments the minor version number, but we can force it to increment the major version number.

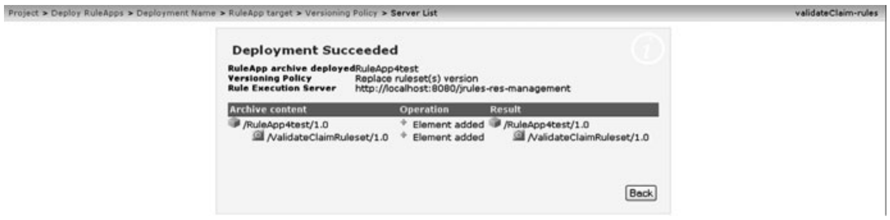
The Query Tab is used to create, edit, and run queries against the rule repository. Recall from, Sect. 8.3.1 that JRules supports queries on rule artifacts, which can search against the properties, contents (definition), and semantics of rules. Similar querying facilities are available within RTS. There are small differences, however, between the types of queries that we can run in RTS versus Rule Studio. For example, some rule properties are only available in RTS, including the login name of the RTS user who created or modified the rule.

Rule Team Server also supports the same ruleset deployment functionalities that are available in Rule Studio, namely:

- Ruleset extraction and archival
- RuleApp definition and generation
- Rule Execution Server configuration management
- RuleApp deployment

These functionalities are accessible to users having the role *RTS Configurator*. Regular RTS users (rule authors) can only deploy existing RuleApps to existing server configurations. Figure 8.15 shows the result screen for a successful deployment of a RuleApp to a given Rule Execution Server installation (e.g., http://localhost:8080/res).

Finally, when Decision Validation Services are installed, additional functionality is enabled in Rule Team Server to allow policy managers to run tests and simulations against their rules. Testing is based on *test scenarios*, which represent the test data and the expected results. Test scenarios can be edited by business users, as they are entered in Excel spreadsheets, where columns represent data elements (object attributes), and each row representing a test case/scenario. The format/layout of the Excel workbook is generated by DVS functionality based on the ruleset parameters. Scenarios can be combined into test suites. Users can also



Project > Deploy RuleApps > Deployment Name > RuleApp target > Versioning Policy > **Server List** — **validateClaim-rules**

Deployment Succeeded

RuleApp archive deployed	RuleApp4test
Versioning Policy	Replace ruleset(s) version
Rule Execution Server	http://localhost:8080/jrules-res-management

Archive content	Operation	Result
/RuleApp4test/1.0	Element added	/RuleApp4test/1.0
/ValidateClaimRuleset/1.0	Element added	/ValidateClaimRuleset/1.0

Fig. 8.15 RuleApp deployment from Rule Team Server

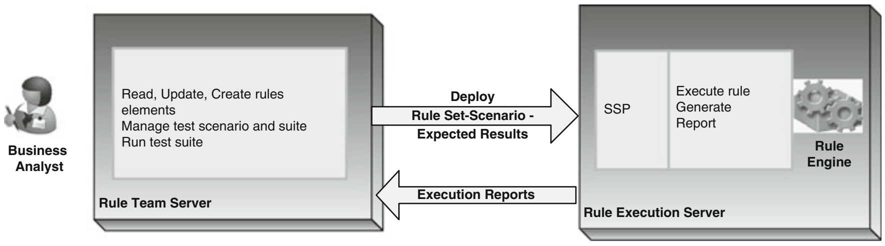


Fig. 8.16 DVS Components View

define *key performance indicators* (KPIs) that are assessed along with the expected results.

The DVS includes functionality to upload test data (Excel spreadsheet) and rulesets to an execution environment called SSP (Scenario Service Provider). The outcome of test execution is a report sent back to Rule Team Server, and displayed in HTML format for review. Figure 8.16 illustrates the process. We will show how to create test scenarios and test suites in Chap. 15.

8.5 Rule Execution Server

The Rule Execution Server (RES) is a managed, monitored execution environment for deployed rulesets. Rule Execution Server handles the creation, pooling, and management of rule sets in order to make rule invocation from the application code more scalable. It natively supports ruleset sharing and rule engine pooling, with the possibility to update rules at runtime. RES provides a management console, from which we can deploy, manage, and monitor RuleApps.

Figure 8.17 shows the architecture of Rule Execution Server (RES). RES can be thought of as two distinct components/stacks that share a database, and that communicate via JMX:

- An *execution stack*, which includes the server-side components to invoke ruleset execution (called *rule sessions*, along with factory/helper classes), an *execution unit* (XU), which knows how to load a ruleset from the database, parse it, and manage a pool engine pool to execute on behalf of business applications.

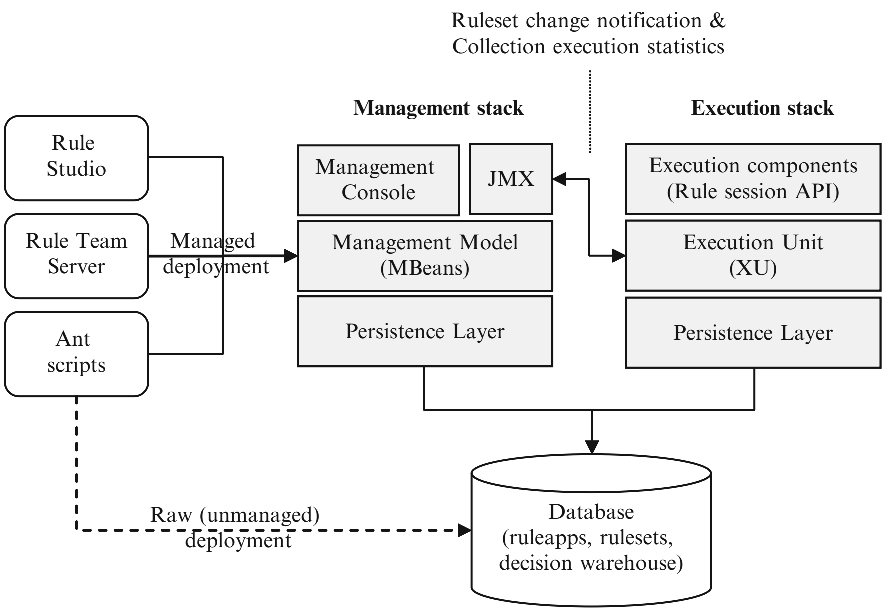

Ruleset change notification &
Collection execution statistics
Management stack
Execution stack
Rule Studio
Rule Team Server
Ant scripts
Managed deployment
Management Console
JMX
Management Model (MBeans)
Persistence Layer
Execution components (Rule session API)
Execution Unit (XU)
Persistence Layer
Raw (unmanaged) deployment
Database (ruleapps, rulesets, decision warehouse)

Fig. 8.17 An overview of the architecture and concept of operations of Rule Execution Server

- A management stack, which knows how to deploy rule apps/rulesets, persist them in the database, manage their versions, *and*, (a) notify the execution stack of the deployment of new or new versions of rulesets, and (b) collect execution statistics from the execution unit (XU). The management stack includes a *web console*, which enables us to perform all of these tasks, and to view management information.

The two components are fairly distinct, and are packaged as separate archives. In fact, in a cluster environment, they will not even be deployed to the same server instances: the execution units will be "clustered", whereas a single managed component is deployed on a server outside of the cluster. Further, business applications that need to execute rulesets will only interact with the execution stack via client-side *execution components* (i.e., rule sessions and rule session factories/ providers), unless they need to collect management information or execution statistics.

Rule Execution Server supports *hot deployment* of rulesets. This means that we can deploy new rulesets/RuleApps, or new versions of existing rulesets/RuleApps, *while the server is running* and *executing the current version of the rulesets*, and ensure that the new versions of the rulesets will be used for subsequent calls. This is made possible thanks to the JMX communication between the management stack (the JMX box in Fig. 8.17) and the execution stack (the execution unit (XU) box in Fig. 8.17). In fact, there are two flavors of this hot deployment:

- What we might call an "eager" hot deployment, which will immediately parse new versions of rulesets, and block any *new* incoming requests for that ruleset until the new version is "installed".
- What we might call a "lazy" hot deployment, which will not block new incoming requests, and let them run with the current version of the ruleset, until the new version is parsed and "installed".

Most business contexts can live with the lazy approach, but some mission-critical applications might require an eager approach.

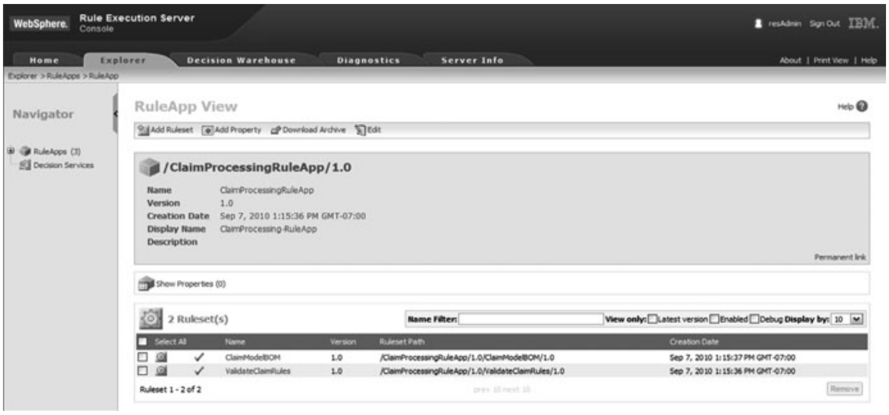
Figure 8.17 also shows the different deployment paths. As mentioned earlier, we can deploy RuleApps from Rule Studio or from Rule Team Server, using live connections to the management component of RES. We can also execute batch ant scripts to do the same thing. Again, there are two flavors:

- Scripts that communicate with the management model. This will ensure that proper versioning is used, and that the proper notifications are sent to the execution units.
- Scripts that access directly the database, without going through the management model. These are simpler to set-up, and more efficient to execute, but may leave the execution components in an inconsistent or out of date state.

Finally, note that there different *deployment flavors* of the Rule Execution Server. While the general architecture suggests a full-fledged J2EE deployment, we can have more lightweight configurations to accommodate the variety of execution contexts that we can encounter:

- Full J2EE deployment, including the cluster deployment mentioned above. In this case, the execution component is a full J2EE application, a JCA resource adapter, where the execution components are EJB or POJO session beans. This is the most scalable configuration, and one in which we can configure the pool size, and the like.
- Web container deployment (e.g., Tomcat 6.x), in which case, we still enjoy the services for JDBC data source management, JMX, and JNDI, and a full management stack deployed as a Web application. The execution component is using a J2SE session.
- Pure Java SE deployment in which RES executes within the same JVM as the calling application, and we need to embed a RES execution JAR within our application. The rule engine pooling is available, but no transaction support and security control.

Figure 8.18 shows the "Explorer" tab of the RES Web management console. It requires a servlet container such as Tomcat 6.0 or other JEE application server like WebSphere Application Server. The RES Console supports different user profiles like *administrator*, *monitor*, or RuleApp *deployer*. *Monitor* role can update already deployed RuleApp and access the reporting data. The RES console includes features to manage RuleApp and ruleset, to run server diagnostics, to view server logged events, and to manage reporting data in the Decision Warehouse. Decision



/ClaimProcessingRuleApp/1.0

Name	ClaimProcessingRuleApp
Version	1.0
Creation Date	Sep 7, 2010 1:15:36 PM GMT-07:00
Display Name	ClaimProcessing-RuleApp
Description	

2 Ruleset(s)   Name Filter: ☐   View only: ☐ Latest version ☐ Enabled ☐ Debug   Display by: 10

Select All			Name	Version	Ruleset Path	Creation Date
☐		✓	ClaimModelBOM	1.0	/ClaimProcessingRuleApp/1.0/ClaimModelBOM/1.0	Sep 7, 2010 1:15:37 PM GMT-07:00
☐		✓	ValidateClaimRules	1.0	/ClaimProcessingRuleApp/1.0/ValidateClaimRules/1.0	Sep 7, 2010 1:15:36 PM GMT-07:00

Ruleset 1 - 2 of 2   prev 10 next 10

Fig. 8.18 Rule Execution Server console – exploring the deployed RuleApp

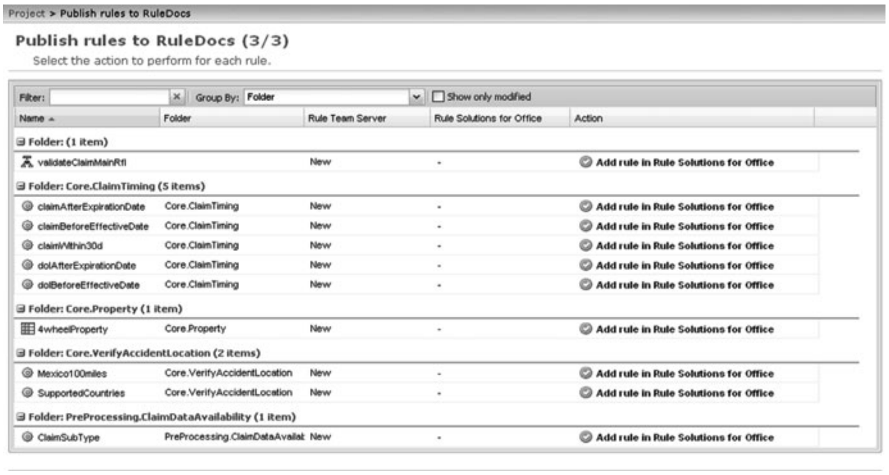
Warehouse delivers a set of features to trace, store, and view and query rule execution activities. Indeed, each ruleset execution can generate traces, which when persisted forms a decision history that can be consulted and used for auditing purposes.

Chapter 13 will go into the details of RES API and explore the different views that deployed rulesets can present to calling business applications. It will also explore in more detail the functionalities of the decision warehouse.

8.6 Rule Solutions for Office

Rule Solutions for Office, or RSO, consists of a set of functionalities that enable us to (1) publish rule artifacts from Rule Team Server to Microsoft Office 2007 Word and Excel documents, (2) edit those rule artifacts with Word and Excel, and (3) upload the new versions back into Rule Team Server to integrate them with the main editing stream within Rule Team Server. Rule Solutions for Office leverages the capabilities of the OpenXML-based file format used by Microsoft Office 2007. We will provide a brief overview of the various functionalities.

To be able to publish Rule Team Server (RTS) projects into Microsoft Office documents, an RTS administrator needs to create a so-called RuleDocs location. This location is a URL to a shared disk on a server or a local directory on user workstation. Once this is done, a rule author can publish a project to the predefined location. They do so through the "*Publish rules to RuleDocs*" action in the "Project" tab of RTS. This will prompt the user for, (a) the RuleDocs location, and (b) the (sub) set of rules to publish. By default, that set/subset consists of all of the action (if–then) rules, and all of the decision tables of the project. However, the user can select which rules to include/exclude, individually, or by specifying a query. Figure 8.19 shows the rule selection form in RTS. The user may choose to publish



Project > Publish rules to RuleDocs

Publish rules to RuleDocs (3/3)

Select the action to perform for each rule.

Filter: Group By: Folder ☐ Show only modified

Name	Folder	Rule Team Server	Rule Solutions for Office	Action
Folder: (1 item)				
validateClaimMainRfl		New	-	Add rule in Rule Solutions for Office
Folder: Core.ClaimTiming (5 items)				
claimAfterExpirationDate	Core.ClaimTiming	New	-	Add rule in Rule Solutions for Office
claimBeforeEffectiveDate	Core.ClaimTiming	New	-	Add rule in Rule Solutions for Office
claimWithin30d	Core.ClaimTiming	New	-	Add rule in Rule Solutions for Office
dolAfterExpirationDate	Core.ClaimTiming	New	-	Add rule in Rule Solutions for Office
dolBeforeEffectiveDate	Core.ClaimTiming	New	-	Add rule in Rule Solutions for Office
Folder: Core.Property (1 item)				
4wheelProperty	Core.Property	New	-	Add rule in Rule Solutions for Office
Folder: Core.VerifyAccidentLocation (2 items)				
Mexico100miles	Core.VerifyAccidentLocation	New	-	Add rule in Rule Solutions for Office
SupportedCountries	Core.VerifyAccidentLocation	New	-	Add rule in Rule Solutions for Office
Folder: PreProcessing.ClaimDataAvailability (1 item)				
ClaimSubType	PreProcessing.ClaimDataAvailat	New	-	Add rule in Rule Solutions for Office

Fig. 8.19 Publish rules to RuleDocs

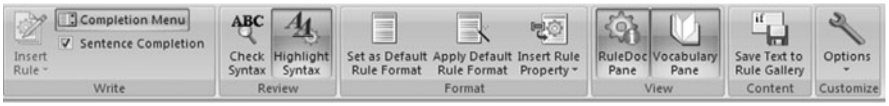


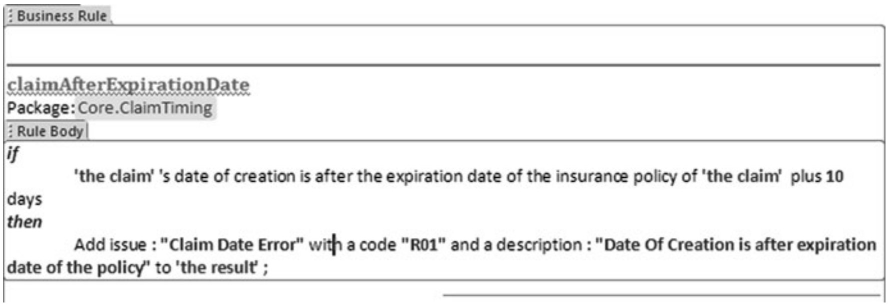
Fig. 8.20 RSO word *Rule Ribbon*

all of the action rules of a project to a single Word document, or to publish each *package* to a separate Word document. The same is true for decision tables and Excel documents.

The installation of RSO will modify Word and Excel, by adding new menu bar buttons, new actions, and anew behavior. Figure 8.20 shows the so-called *Rule-Ribbon* added to Word.

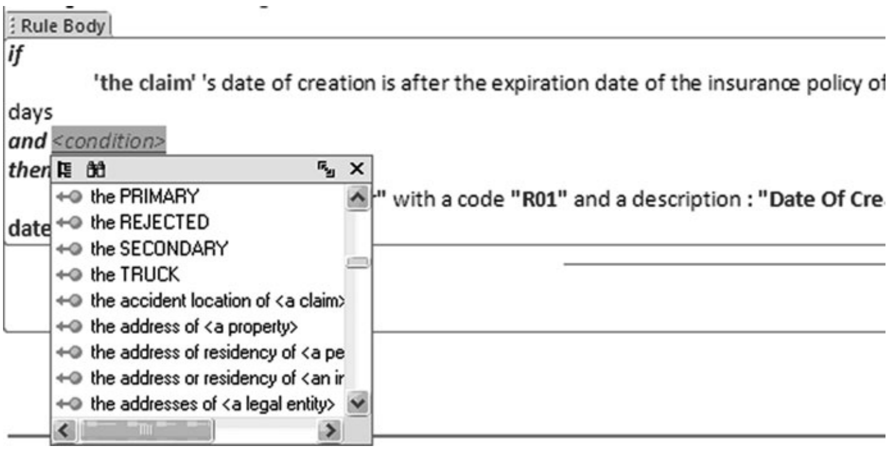
In Word, a *RuleDoc* includes a *Write pane* (the main editing frame in Word), a *View pane*, and a *Review pane*. The Write pane is used to insert new rules or edit existing ones. This pane supports a number of custom completions modes activated with CTRL-SPACE, SPACE, TAB, and so forth. The review pane is used to check the rules for errors and to activate or deactivate automatic syntax highlighting. The View pane is used to show or hide the RuleDoc pane and Vocabulary panel. Each business rule in a RuleDoc is stored within a *content* control that separates the contents of the business rule from the surrounding text. In Fig. 8.21, the top-level content control includes the name of the rule, its package hierarchy, while the Rule Body content control includes the text of the rule.

Content controls react to user actions and act as custom editors within Word. Completion assistants provide guidance while writing business rules by showing the applicable vocabulary terms that can be used at this particular location of the rule. This is illustrated in Fig. 8.22. They are similar to the code completion


Business Rule

claimAfterExpirationDate
Package: Core.ClaimTiming
Rule Body
if
'the claim' 's date of creation is after the expiration date of the insurance policy of 'the claim' plus 10 days
then
Add issue : "Claim Date Error" with a code "R01" and a description : "Date Of Creation is after expiration date of the policy" to 'the result' ;

Fig. 8.21 Two content controls for one rule


Rule Body
if
'the claim' 's date of creation is after the expiration date of the insurance policy of
days
and <condition>
then
" with a code "R01" and a description : "Date Of Cre
date

the PRIMARY
the REJECTED
the SECONDARY
the TRUCK
the accident location of <a claim>
the address of <a property>
the address of residency of <a pe
the address or residency of <an ir
the addresses of <a legal entity>

Fig. 8.22 BOM – term navigation

assistants available in Rule Studio's text-based rule editor (*Intellirule*). The content control includes also a rule syntax checker, similar to Word's standard spelling and grammar checker, that enables users to check and correct business rules.

Once authoring is done within Rule Solution for Office, the rule writer can use the Update rules from RuleDocs feature of Team Server to update RTS repository.

8.7 Summary

This chapter provided a brief overview of the IBM WebSphere ILOG JRules BRMS. We described the overall architecture of the product, the concept of operations, and then, for each component of the product, we identified the main workflows and activities supported by the component and gave a brief overview of those activities, throwing in some best practices along the way.

Many of these activities and workflows will be revisited in the subsequent chapters, namely:

- *The design of rule project structures*. General, product-neutral issues will be discussed in Chap. 9, and JRules specifics in Sect. 10.2.
- *The design of the Business Object Model (BOM)*. General issues, related to the different requirements between the BOM and XOM, and relationships between them, will be discussed in general terms in Sect. 9.2.1, and for the case of JRules in Sect. 10.3.
- The rule artifacts will be presented in a product-independent way in Sect. 9.2.2; JRules rule artifacts will be presented in Sect. 11.2.
- The orchestration of rule execution, will be discussed in Sects. 11.3 (foundations) and 11.4 (best practices).
- Ruleset deployment and execution will be discussed, in general terms, in Chap. 12, and with JRules specifics, in Chap. 13.
- Rule testing will be discussed in general terms, in Chap. 14, and with JRules testing functionalities, in Chap. 15.
- Finally, everything related to rule governance will be discussed in general terms, in Chap. 16, and for the case of JRules, in Chap. 17.

This is the only chapter of the book that is product/feature driven. The remainder of the book is *activity*- or *issue-driven*, even for those chapters that deal with JRules specifics. As the product evolves, some of the actions, menus, and screens shown in this chapter may change. However, the major issues and activities should remain the same, barring a profound change in the product architecture and concept of operations.

8.8 Further Reading

This being a product-driven chapter, the reader is referred to the product documentation for more technical information and tutorials, which is now public accessible at http://publib.boulder.ibm.com/infocenter/brjrules/v7r1/index.jsp.