Develop decision services, Part 2: Rules development process

Skill Level: Introductory

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

Sandeep Desai (sandeep@us.ibm.com)
Enterprise IT Architect
IBM

26 Jul 2011

Part 1 introduced you to the smarter city scenario and the use case requirements. We described the role of the decision subsystem and the rationale for selecting IBM WebSphere® ILOG® JRules as a business rule management system (BRMS) to implement the decision subsystem. Part 2 walks you through the how-to of the rules development process.

Rule application development

Technical developers use Rule Studio, an Eclipse IDE-based tool, to create the foundational rule artifacts, including Java™ development and rule project development. This article is not intended as a detailed tutorial, but instead a walk through of the development process. You can follow along using the complete workspace, see Download. As a prerequisite, you should have IBM WebSphere ILOG JRules Rule Studio 7.1.1 installed on your workstation.

As noted in Part 1 of this series, rule application development occurs after rule discovery and analysis of the initiation phase. At a high level, rule application development consists of the following steps:

1. Create projects

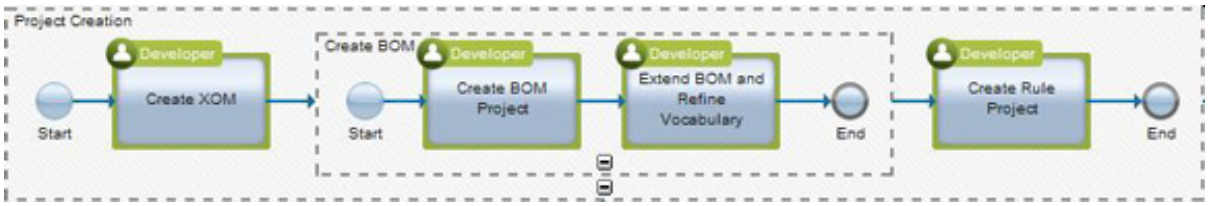
© Copyright IBM Corporation 2011

2. Design rules structure, including rule packages and ruleflows
3. Write business rules

Create projects

Using Rule Studio, the developer first creates a regular Eclipse workspace and then creates projects of various types within it, including Java projects and Rule projects. These projects constitute the Execution Object Model (XOM), the Business Object Model (BOM), and the rulesets. Figure 1 depicts project creation process.

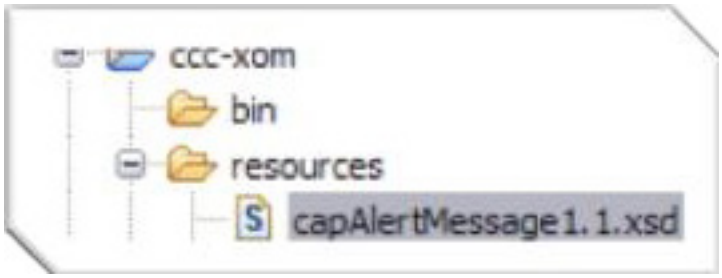
Figure 1. Project creation process



Create XOM project

The XOM, which can be a set of Java classes or XSD files, represents the physical data model – that is, the actual data that is passed to and from the decision service during execution. In our scenario, the Common Alerting Protocol (CAP) XSD (see OASIS CAP October update under Resources), constitutes the XOM. A simple project called "ccc-xom" is created to contain the XSD as shown in Figure 2.

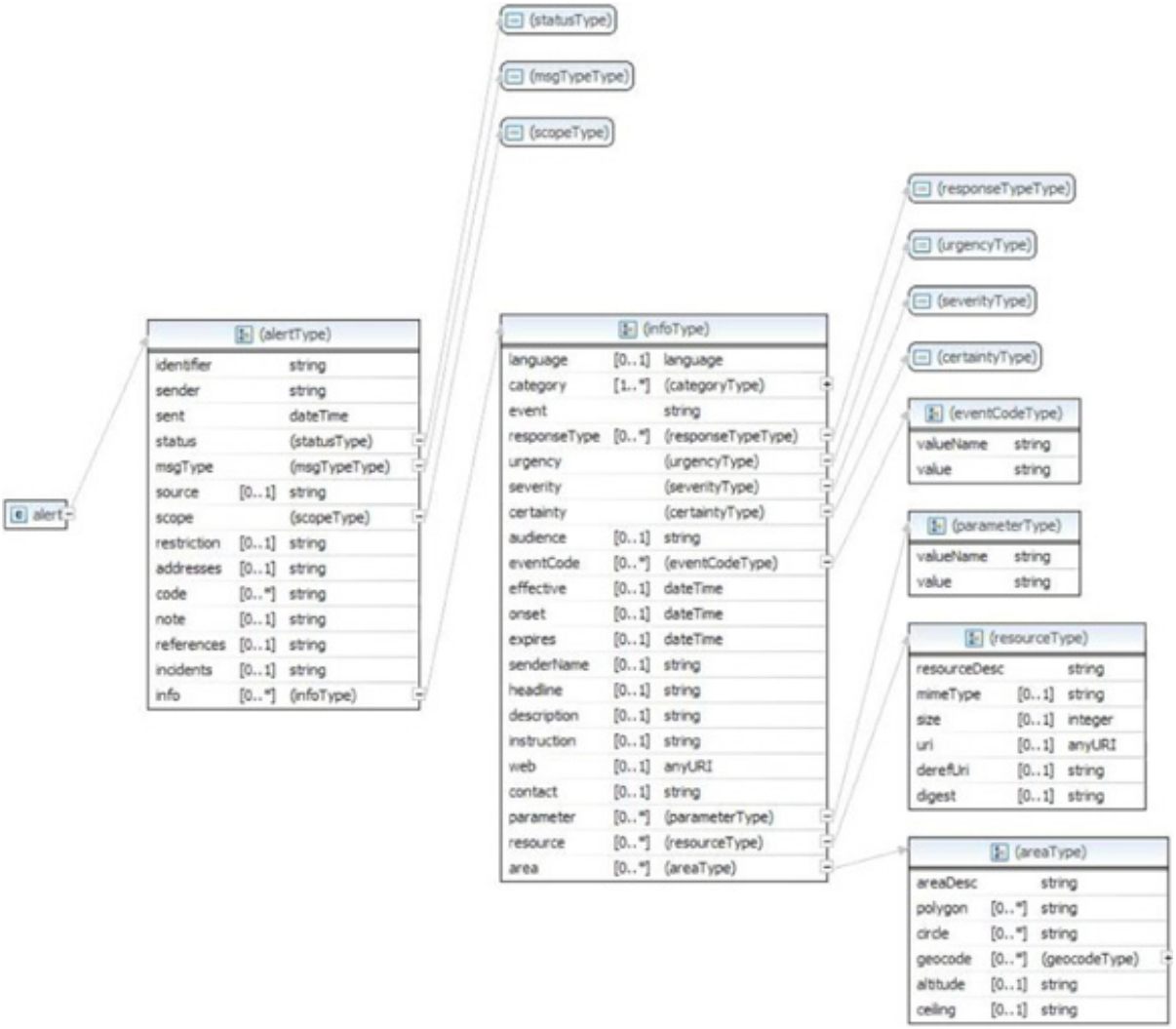
Figure 2. XOM project in the package explorer



In our use case, we create a BOM from the XOM. In this bottom-up approach, it is important to fully understand the structure and elements of the XOM as they form the basis for the BOM and rule vocabulary. In the CAP XSD, an `Alert` contains multiple `InfoTypes`, which in turn contain one or more `ParameterTypes` and `EventCodeTypes`. *Parameters* specify data values, such as the observed rainfall over the last 12 hours. Figure 3 depicts these XOM elements.

Figure 3. XSD schematic diagram

© Copyright IBM Corporation 2011



(View a larger version of Figure 3.)

Create BOM project

The strength of WebSphere ILOG JRules BRMS is that it allows writing business logic in natural language based on business terms, rather than in a programming language. For this task, you create a BOM that is a logical model of the business domain and that describes the data on which your decision is based. Each BOM element that is visible to the rule author is assigned a natural language verbalization. All elements of the decision are written in terms of this vocabulary.

In Rules Studio, using a bottom-up approach, importing the XOM creates the BOM. Before creating the BOM project, switch to the Rule Perspective in Rule Studio. Then, create the BOM project using a wizard invoked through **File - New - Rule Project**. Specify `capAlertMessage1_1.xsd` as the dynamic binding for the BOM. Figure 4 illustrates the sequence of steps used in creating the BOM project from an

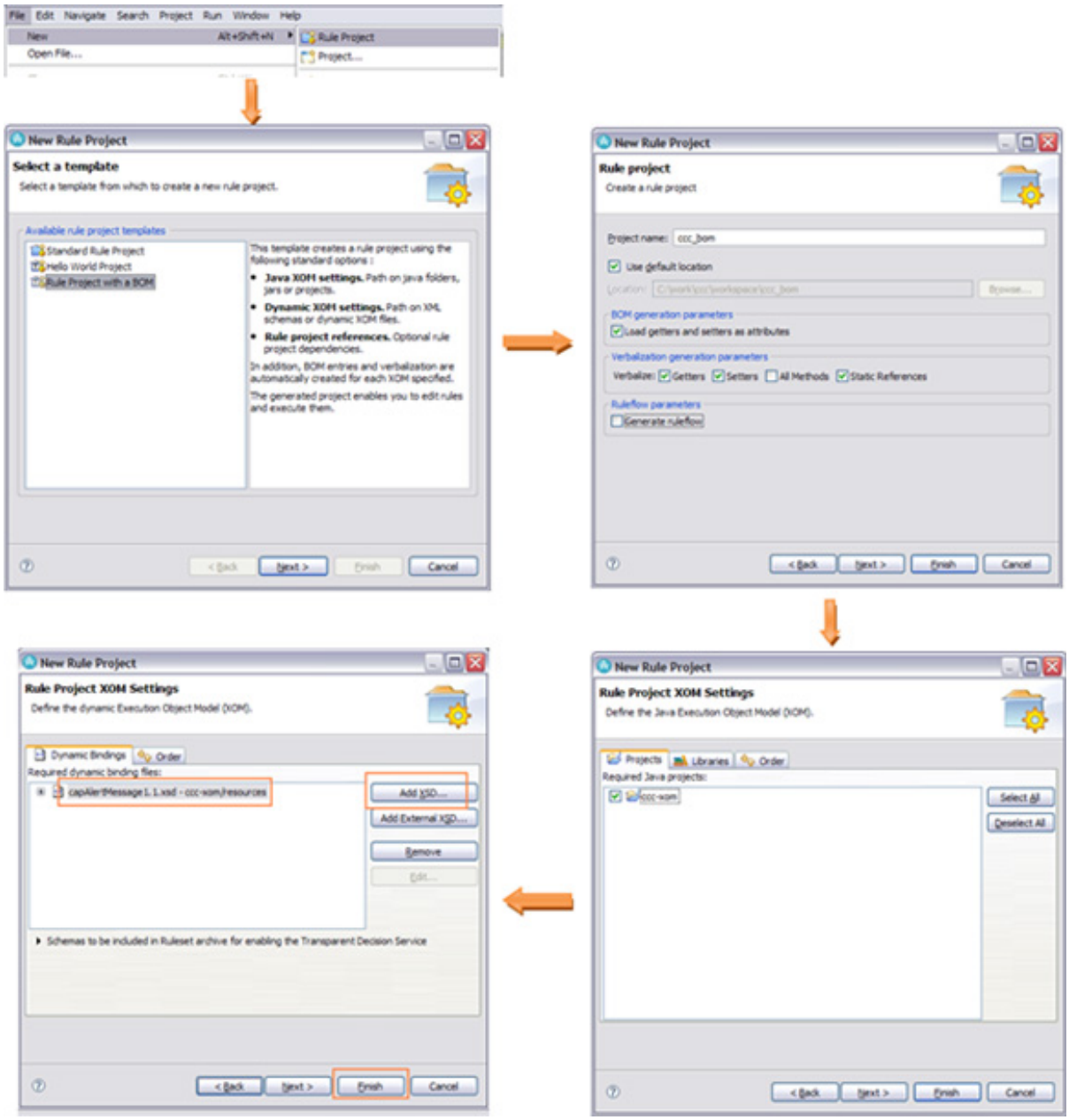
© Copyright IBM Corporation 2011

XSD XOM. To create the BOM project, follow these steps:

1. Select **File – New – Rule Project**.
2. In the window that opens, select **Rule Project with a BOM** as the template and click **Next**.
3. In the following window, specify “ccc-bom” as the project name and click **Next**.
4. Click **Next**.
5. Select `capAlertMessage1_1.xsd` using the **Add XSD** button in the **Dynamic Bindings** tab in **Rule Project XOM Settings** and click **Finish**.

Figure 4. Create BOM project

© Copyright IBM Corporation 2011

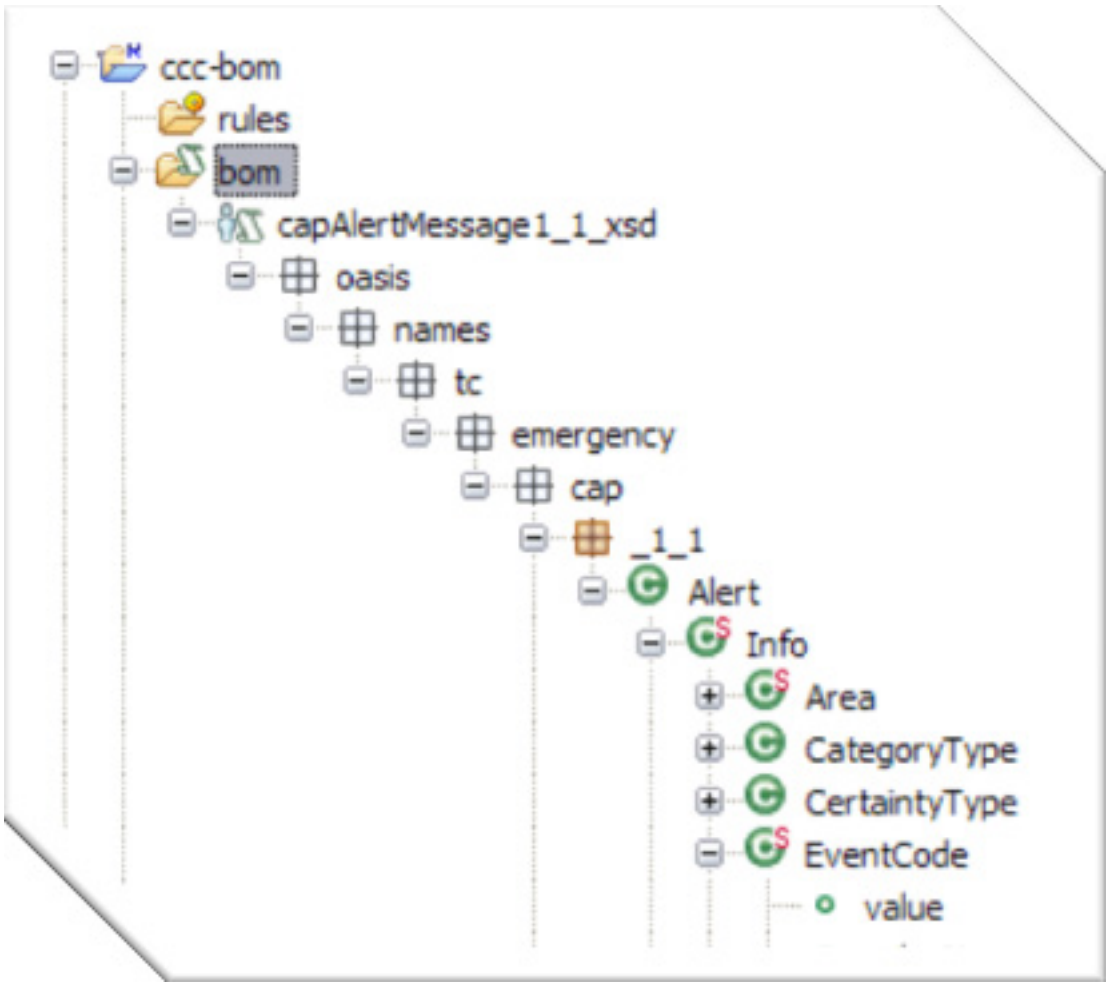


(View a larger version of Figure 4.)

This set of steps creates BOM entries for all the elements in the XOM, with the default verbalizations to access and set each of the BOM elements. Figure 5 shows some of the BOM elements and packages in the Rule Explorer.

Figure 5. BOM project in the Rule Explorer

© Copyright IBM Corporation 2011



The default verbalizations are sufficient in most cases. For example, the generated verbalizations for setting and getting the *value* of an *EventCode* can be seen in the BOM editor by double-clicking the BOM element, as shown in Figure 6.

Figure 6. Verbalization in BOM editor

(View a larger version of Figure 6.)

Extend BOM and refine vocabulary

© Copyright IBM Corporation 2011

Rule vocabulary is the set of terms that can be used to precisely define a business rule. These terms are verbalizations of the BOM. Often, we need to enhance the BOM by adding elements or methods to ease rule writing. For example, the *value* of a *Parameter* is present in the input model as a string. To compare this value against numeric values in rules, we define a “virtual” or “synthetic” method *getNumericValue* in the BOM class *Parameter*, which extracts a double value from the string, and give it a verbalization. This step enables us to write rules such as this one:

```
if the numeric value of this parameter is at least 15
then …
```

Every BOM element needs to eventually map back to the executable XOM. For "virtual" methods that do not have a corresponding XOM implementation, this mapping is done through the BOM-to-XOM mapping. At runtime, the rule engine parses the ruleset and uses BOM-to-XOM mapping to access the XOM.

Figure 7 shows the virtual method in the BOM editor. To define this virtual method, follow these steps in the BOM editor:

1. The **Name** is set to *getNumericValue*.
2. The **Type** is set to java.lang.Double.
3. In the **Member Verbalization** section, **Navigation** is set to {numeric value} of {this}
4. In the BOM to XOM Mapping section, enter this:

```
Double d = 0;
try {
        d = Double.parseDouble(this.value);
} catch (Exception e) { }
return d;
```

Figure 7. Adding a virtual method

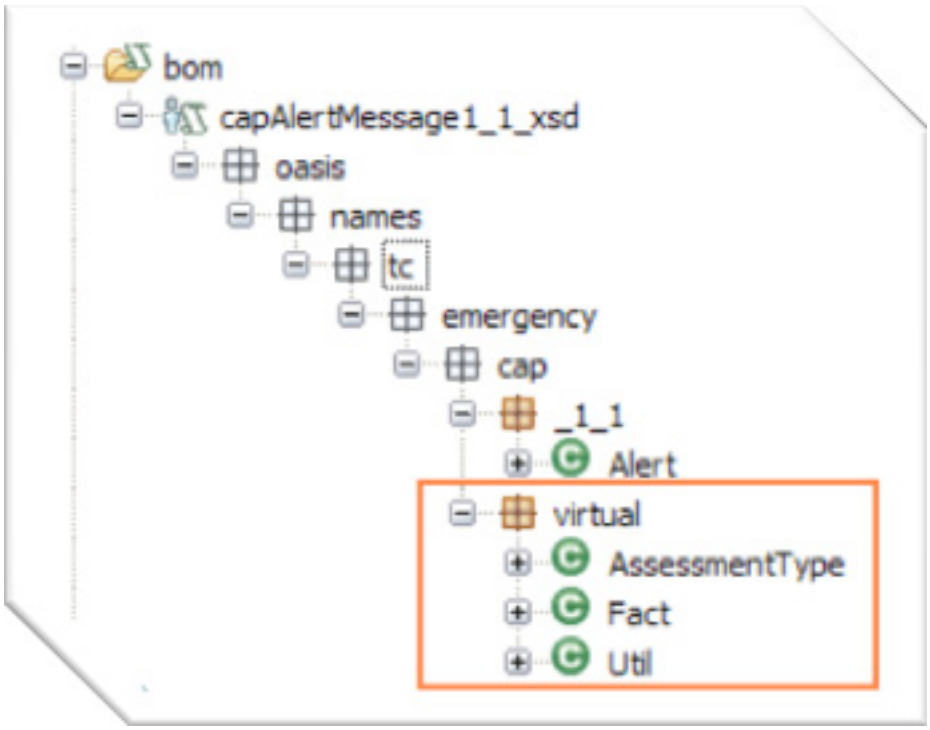
© Copyright IBM Corporation 2011

(View a larger version of Figure 7.)

Besides virtual methods, we add some virtual classes to implement synthetic objects. Figure 8 highlights these virtual classes. (See Resources for more details.)

Figure 8. Virtual classes to implement synthetic objects

© Copyright IBM Corporation 2011



Create a rule project

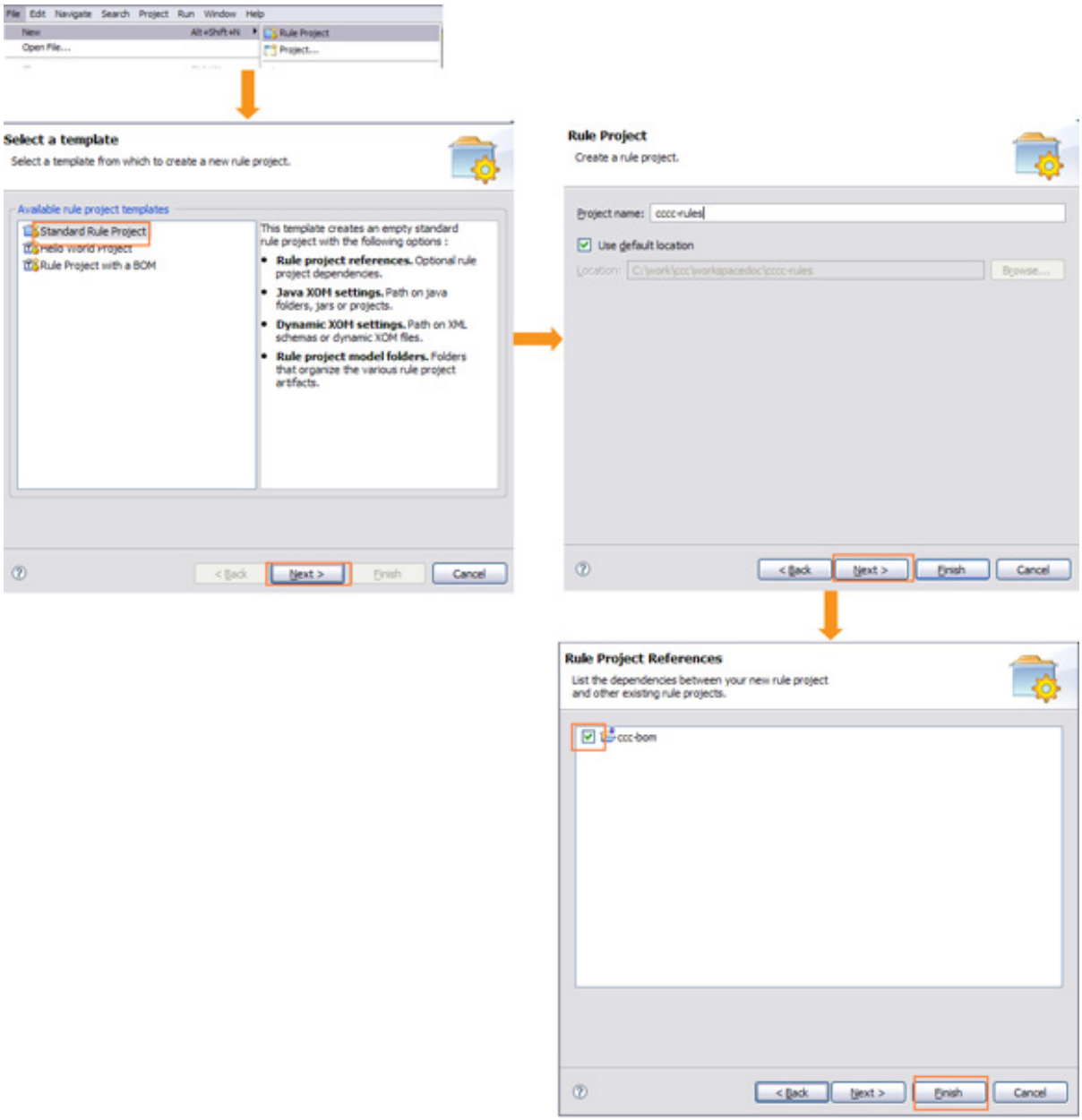
Technically, the *ccc-bom* project is a rule project that can contain business rules as well. However, we recommend that the BOM be placed in its own project, with business rules in a separate project. This approach allows future reuse of the BOM for other rulesets. Therefore, we use the new rule project wizard to create a separate rule project called *ccc-rules*.

Figure 9 depicts the sequence of steps used in creating the rule project as a standard rule project referencing the *ccc-bom* rule project. To create the rule project, follow these steps:

1. Select **File – New – Rule Project.**
2. In the window that opens, select **Standard Rule Project** as the template and click **Next**.
3. In the following window, specify “ccc-rules” as the project name and click **Next**.
4. Select `ccc-bom` in the **Rule Project Reference** and click **Finish**.

Figure 9. Create rule project

© Copyright IBM Corporation 2011



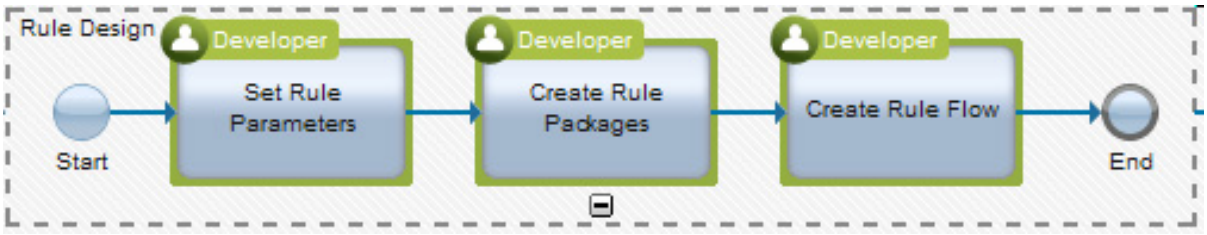
(View a larger version of Figure 9.)

Rule design

Design of a ruleset includes identifying the input and output of a ruleset, identifying how rules are organized within a rule project, and creating ruleflows to orchestrate the execution of rules within the ruleset. Figure 10 shows the rule design process, which we detail in subsequent subsections.

Figure 10. Rule design process

© Copyright IBM Corporation 2011

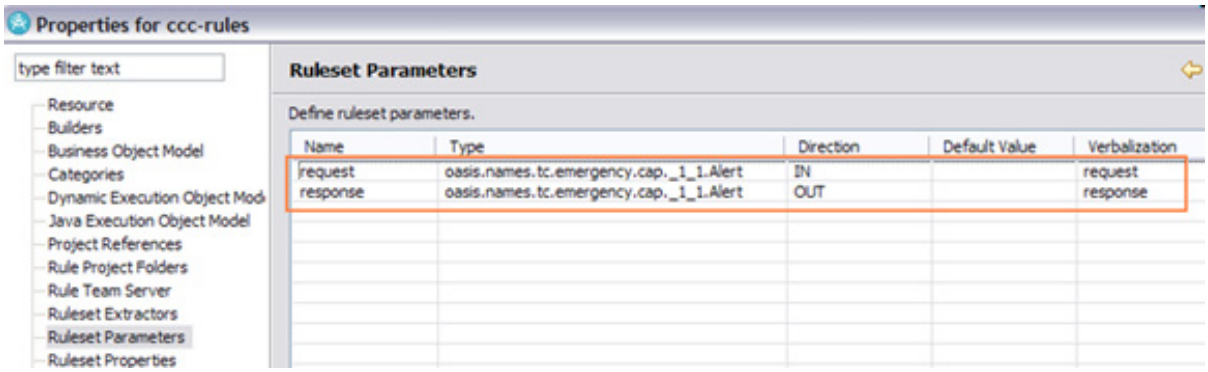


Set ruleset parameters

Ruleset parameters specify the inputs and outputs to and from the decision service and therefore define the interface between the decision service and the invoking client.

To set the ruleset parameters, right-click *ccc-rules* and select *Properties*. In our scenario, we receive an *Alert* as input and return an enriched *Alert* as the output. They are verbalized as "*request*" and "*response,*" respectively, as shown in Figure 11.

Figure 11. Ruleset parameters



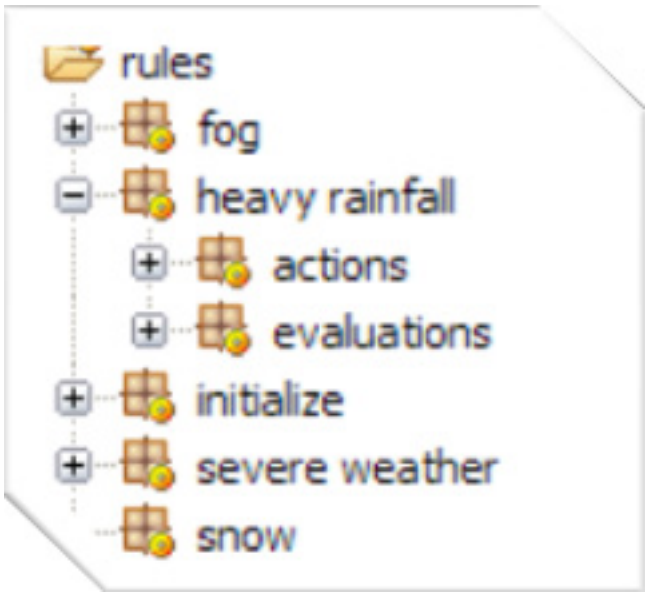
Create rule packages

Rule packages are simply folders and subfolders in which you can create rules. Rule packages help organize the rules in a hierarchical structure.

Based on our rule analysis, we determine that we have different rules for different weather patterns, such as "fog" and "heavy rainfall." This distinction forms our top-level rule packages. For each of them, we have subpackages for assessment rules and directive rules. Figure 12 shows a subset of the rule packages used in the rule project.

Figure 12. Rule packages

© Copyright IBM Corporation 2011



Create ruleflow

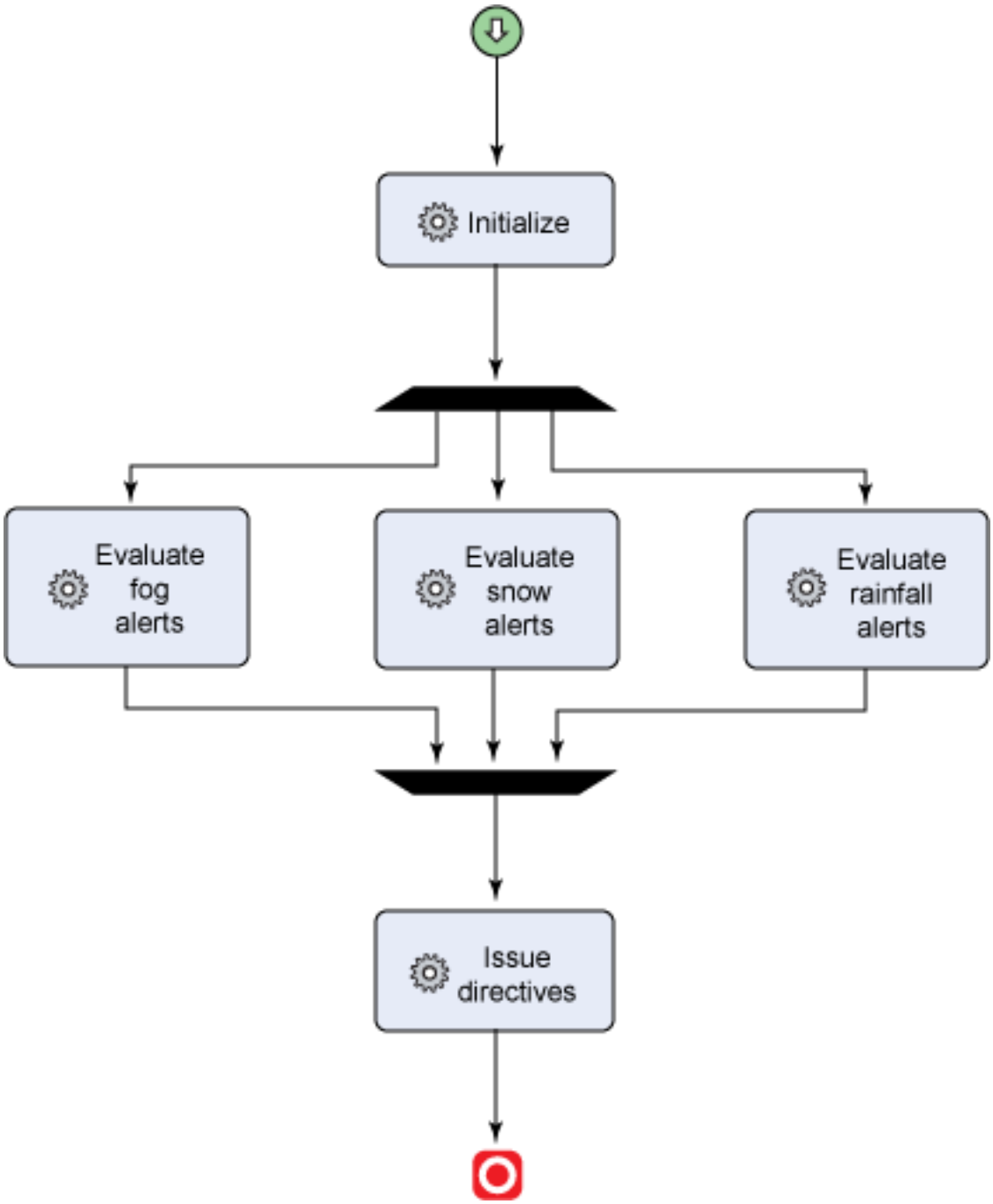
The ruleflow defines the execution order of rule artifacts within the context of a larger decision. To the calling application, the ruleset is invoked to make a single business decision. That business decision, however, typically breaks down into a series of smaller decisions. Using a ruleflow, you identify the smaller decisions as rule tasks and the routing logic as transitions.

The first rule task in the ruleflow is *Initialization*, where a skeletal response `Alert` is created. Following that, assessments are made based on which directives are issued.

Figure 13 shows this ruleflow.

Figure 13. Main ruleflow

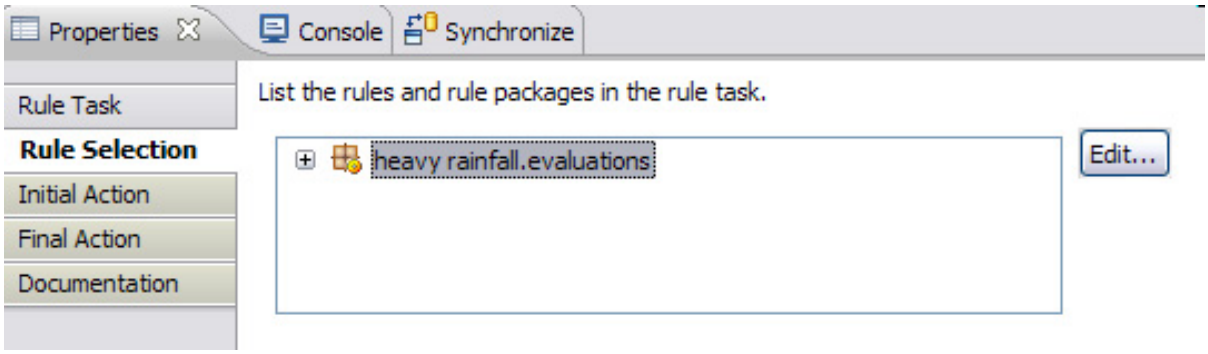
© Copyright IBM Corporation 2011



The rules associated with a rule task are specified in the properties of a rule task. For the rule task *heavy rainfall*, all the rules that are in the rule package *heavy rainfall.evaluations* are considered during execution of this rule task. Figure 14 shows the properties view of the rule task *heavy rainfall* where the associated rule selection is made.

Figure 14. Rule selection for heavy rainfall rule task

© Copyright IBM Corporation 2011



Note that ruleflows control the execution order among groups of rules, or rule tasks. Unlike the method used procedural programming, here you do not specify the order of individual rules. Many newcomers who have a background with procedural programming paradigms have a tendency to overuse ruleflows. For instance, the initial tendency might be to use control looping through the different `InfoType` objects associated with an `Alert` using ruleflows. That is not the recommended approach, though, and we shall see how rules can be used to naturally perform this looping in the following section.

Business rule authoring

The rule developer typically creates the initial set of rules and performs unit testing to ensure that the rule projects provide a solid foundation on which business users can build. In addition, a technical developer can create rule templates to ease authoring of new business rules by business users. Figure 15 depicts the rule authoring process.

Figure 15. Rule authoring process

Write initial set of rules

A rule developer uses Rule Studio to create the initial set of rules. As we've already seen in Part 1 of this series, in our scenario these rules fall under two categories: assessment rules and directive rules.

Assessment rules

Evaluation rules post assessments as intermediate facts based on incoming alerts. The assessment rules that handle rainfall events are created in the *heavy rainfall.evaluations* package. The example below posts a `HEAVY_RAINFALL`

© Copyright IBM Corporation 2011

assessment fact if over 150 mm of rainfall is observed in the last 12 hours. When all the conditions of the rule are met, the rule action, or the "then" part of the rule, is executed; this action creates an assessment fact and posts it to the working memory.

This rule is depicted in Listing 1.

Listing 1. Example rule based on rainfall over last 12 hours

```
definitions
        set 'rainfallInfo' to an info in the infos of request
                where the event of this info is "HeavyRainfall"
                        and the certainty of this info is "Observed"
                        and the severity of this info is "Severe" ;
if
         there is no fact where the type of this fact
is HEAVY RAINFALL,
         and there is at least one parameter in the parameters of rainfallInfo
                 where the value name of this parameter is "RainfallLevel12H"
                         and the numeric value of this parameter is at least 150,
then
        create an assessment of HEAVY RAINFALL using rainfallInfo;
```

As can be seen, a rule is represented as an if-then statement. A rule, though, is different from an if-then statement in a traditional procedural language, such as Java or C. Let's dissect this rule to gain some insights into rule programming. Table 1 identifies the different parts of a Business Action Language rule and interprets its meaning from a developer perspective.

Table 1. Rule parts and their meaning

Rule part	Meaning

`definitions`
`set 'rainfallInfo' to an info in the infos of request`
`where the event of this info is "HeavyRainfall"`
`and the certainty of this info is "Observed"`
`and the severity of this info is "Severe" ;`
	This code sets up a rule variable called *"rainfallInfo*." This rule variable matches on every *InfoType* in the incoming *Alert* whose event is *HeavyRainfall*, *certainty* is "Observed," and *severity* is "Severe." If an alert comes in with three different *InfoTypes*, all of them being "HeavyRainfall," but only two of them are "Observed" and the third is "Possible," then this rule variable is instantiated with only the two "Observed" *InfoTypes*. Before the rule executes, other conditions specified in the rule must be met, too.

`if`	This code starts the set of other rule conditions.
`there is no fact where the type of this fact is HEAVY RAINFALL`	To avoid duplications of assessments, this code is a check to ensure that the HEAVY RAINFALL assessment has not already been made.
	An *InfoType* object can have several

© Copyright IBM Corporation 2011

`there is at least one parameter in the parameters of rainfallInfo`
`where the value name of this parameter is "RainfallLevel12H"`
`and the numeric value of this parameter is at least 150 ,`
	parameters. This condition checks that there is at least one parameter called "RainfallLevel12H" (which signified rainfall in the last 12 hours) whose value is to at least 150 mm.

`then` `create an assessment of HEAVY RAINFALL using rainfallInfo`	If all the conditions specified in the *definitions* and *if*"\ part of the rule are satisfied, then the rule is executed. In this rule, this code creates an *assessment* called HEAVY RAINFALL and associates *`rainfallInfo`* with it.

Another assessment rule checks the observed rainfall values over the last 1 hour as well as the last 12 hours and makes a *`HEAVY RAINFALL`* assessment if it rains at least 10 mm over the last hour and over 100 mm over the last 12 hours. This rule is shown in Listing 2.

Listing 2. Example rule based on multiple parameters

```
definitions
        set 'rainfallInfo' to an info in the infos of request
                where the event of this info is "HeavyRainfall"
                        and the certainty of this info is "Observed"
                        and the severity of this info is "Severe" ;
if
         there is no fact where the type of this fact
is HEAVY RAINFALL,
         and  there is at least one parameter in the parameters of rainfallInfo
                where the value name of this parameter is "RainfallLevel1H"
                        and the numeric value of this parameter is at least 10,
         and there is at least one parameter in the parameters of rainfallInfo
                where the value name of this parameter is "RainfallLevel12H"
                        and the numeric value of this parameter is at least 100,
then
        create an assessment of HEAVY RAINFALL using rainfallInfo;
```

Directive rules

Directive rules match on the assessments to generate action items, such as notifications and directives using information from the associated *`InfoType`* object. Any number of events might lead to an assessment of "HEAVY RAIN," but only one set of action items is created based on this assessment.

Listing 3 illustrates a directive rule.

Listing 3. Rule to create action items

```
definitions
        set heavyRainfall to a fact where the type of this fact is HEAVY RAINFALL ;
        set info to the param of heavyRainfall ;
if
        true
then
        set new_info to create new info;
        set the sender name of new_info to "CityCommandCenter-BRE";
        set the event of new_info to "AlertLevelRainfallAssessment";
        set the language of new_info to "en-US";
```


© Copyright IBM Corporation 2011

```
        add "Safety" to the categories of new_info;
        set the certainty of new_info to the certainty of info;
        set the audience of new_info to "CityWaterDomain CityPublicSafetyDomain";
        add event code <name> "Type" <value> "Notification" to
new_info;
        set the headline of new_info to "Level Rainfall Assessment";
        set the description of new_info to "Heavy rainfall expected;
          Assessing rainfall vs. sewer capacity";
        set the contact of new_info to "ccc@rotterdam.com";
        set new_resource to create new resource;
        set the resource desc of new_resource to "888001";
        add new_resource to the resources of new_info;
        for each area in the areas of info :
        - add this area to the areas of new_info;
        remove <area #> 1 from the areas of new_info;
        set the urgency of new_info to "Immediate";
        set the severity of new_info to "Severe";

        add new_info to the infos of response;
```

In the rule shown in Listing 3, based on the presence of a HEAVY RAINFALL fact in the working memory, a new *Info* object is created with notifications and is added to the response.

Create rule templates

To ease creation of rules that are structurally similar, developers can create rule templates, which are partially written business rules. These rules can be used as a starting point to create multiple rules with similar content and structure.

In our scenario, we notice that the evaluation rules start by matching on an *info* object in the list of observed *info* objects and then drill down to *parameters* while ensuring that the *assessment* fact does not already exist. This structure is captured through the rule template shown in Listing 4. The underlined items are the placeholders that a rule author fills out to define the rule.

Listing 4. Rule template for assessment rules

```
definitions
set infoEvent to # an info in the infos of request
        where the event of this info [±] is # <enter a string> [±]
            and the certainty of this info [±] is Observed [±]
            and the severity of this info [±] is Severe [±]
if
there is no fact [from/in]
        where the type of this fact is <an object>

    and there is at least one parameter in the parameters of infoEvent
        where the value name of this parameter [±] is # <enter a string> [±]
            and the numeric value of this parameter [±] is at least # <enter a
            number> [±]

then
create an assessment of <an assessment type> using infoEvent
```

Using this template, business users can easily create new assessment rules by essentially filling in the blanks provided by the template.

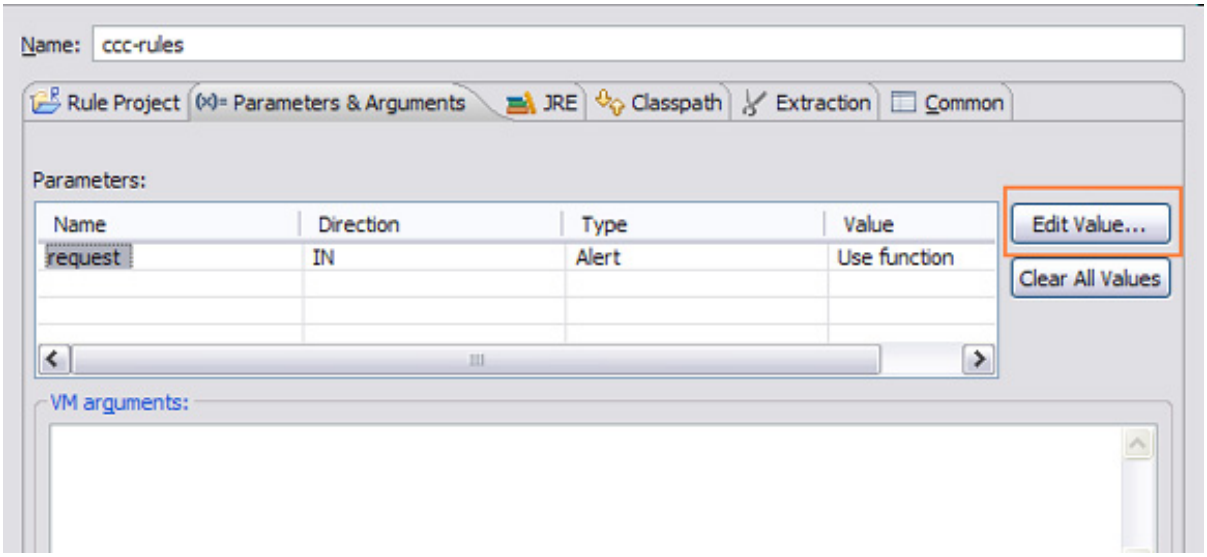
© Copyright IBM Corporation 2011

Unit test rules

Unit testing of business rules is a very important step toward creating a robust rule application. This fact is particularly true when adopting an agile development methodology. A technical developer performs unit testing not only to validate the initial set of rules, but more importantly to ensure that the BOM and XOM are free of errors. Unit testing can be as simple as running tests using the Rule Studio run configuration, or it might be a more elaborate test suite using a JUnit-based test framework.

In our scenario, we use a simple run configuration to run and debug rules. By right-clicking the rule project and selecting *Run As - Rule Project*, the *Run As Configuration* is created, which is then modified to create an input request using a WebSphere ILOG JRules function. Figure 17 shows this scenario.

Figure 17. Rule project run configuration



The WebSphere ILOG JRules function we use to create an `Alert` as the request is the code shown in Listing 5.

Listing 5. The WebSphere ILOG JRules function

```
oasis.names.tc.emergency.cap._1_1.Alert result
        = new oasis.names.tc.emergency.cap._1_1.Alert();
result.sender = "UNIT_TEST";
result.identifier = "TEST";
result.sent = new ilog.rules.xml.types.IlrDateTime(new java.util.Date());
result.infoList = new java.util.Vector();
oasis.names.tc.emergency.cap._1_1.Alert.Info info
        = new oasis.names.tc.emergency.cap._1_1.Alert.Info();
result.infoList.add(info);
info.certainty = "Observed";
info.severity = "Severe";
```


© Copyright IBM Corporation 2011

```
info._event = "HeavyRainfall";
info.parameterList = new java.util.Vector();
oasis.names.tc.emergency.cap._1_1.Alert.Info.Parameter param
        = new oasis.names.tc.emergency.cap._1_1.Alert.Info.Parameter();
info.parameterList.add(param);
param.valueName="RainfallLevel1H";
param.value="16";
return result;
```

When we run this configuration in debug mode, we can create breakpoints in ruleflows and rules, view rules that are fired, and browse the objects in working memory. By running a few scenarios in this manner, a rule developer can ascertain that the BOM and the initial set of rules are error free. The rule developer works closely with business users to ensure the validity of the rules.

Conclusion

We have seen that WebSphere ILOG JRules is a powerful tool that offers a rich set of wizards and accelerators for rule application development. Using a case study, we walked through a commonly used rule development process that can be used by a technical developer to create and test rules that are discovered during the initiation phase. In the next article in this series, we describe the tasks that enable non-technical business users to write and test rules.

© Copyright IBM Corporation 2011

Downloads

Description	Name	Size	Download method
	cccrules_pif_051811.zip		HTTP

Information about download methods

© Copyright IBM Corporation 2011

Resources

Learn

- IBM WebSphere ILOG JRules product page: Access more about the features and benefits, system requirements, and support from the product home page.
- IBM WebSphere ILOG JRules Information Center: Find more information about this product line and its features.
- OASIS Common Alerting Protocol 1.1: The Common Alerting Protocol (CAP) is a simple but general format for exchanging all-hazard emergency alerts and public warnings over all kinds of networks. See the October 2007 update too.
- Policies and Rules – Improving business agility: Part 1: Support for business agility (Hondo, Boyer, Ritchie, developerWorks, March 2010): One challenge in architecting and implementing agility in business solutions today is that the use of the terms, policy, and rule, differs across products. Learn the concepts and relationships of policies and rules technologies to implement specific business strategies and tactics.
- Creating intermediate facts in WebSphere ILOG JRules using synthetic objects (Raj Rao, developerWorks, November 2010): See this WebSphere Technical Journal article for more information about virtual classes to implement synthetic objects.
- IBM developerWorks Industries: Get all the latest industry-specific technical resources for developers.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks on Twitter: Join today to follow developerWorks tweets.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

- WebSphere ILOG JRules V7.1: Get the trial download.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox, and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- developerWorks blogs: Check out these blogs and get involved.

© Copyright IBM Corporation 2011

About the authors

Raj Rao

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

Sandeep Desai

Sandeep Desai is a Senior Certified Enterprise IT Architect with IBM WebSphere's Business Partner Technical Professional team. He works with strategic business partners from startup to large firms. He mentors them to SOA-enable their solution. He evangelizes, educates, and enables partners on IBM software platform. Sandeep is Open Group Distinguished Certified IT Architect, IBM Senior Certified Enterprise IT Architect, and IBM Certified SOA Solution Designer.

© Copyright IBM Corporation 2011