Best practices for designing and implementing decision services, Part 1: An SOA approach to creating reusable decision services

Jerome Boyer (boyerje@us.ibm.com)
Senior Technical Staff Member
IBM

Decision services are quite the buzz term lately, and are even turning up as product features. As a solution architect or developer you may have seen the need to go outside of the pure product scope and think about designing and implementing decision services so that they can be easily maintained and consumed. Part 1 of this series addresses some best practices for designing and implementing a decision service using IBM® Business Process Manager Advanced and WebSphere® Operational Decision Management. The article defines the key points to consider when designing rule-based decision services that follow best practices and fit into a large IT architecture.

View more content in this series

Overview

A decision service is a service operation making decisions, and in the context of this article, is implemented using business rule technology. As a service it is reusable logic serving a business function. Recently, there is a tendency to use the term decision service everywhere, even as a product feature name. Another tendency is to use product features indiscriminately, short-cutting the design and considerations of what a decision service needs to be in SOA, forgetting the years of SOA practice. A decision service activity in a business process model that is defined using a top-down approach is not a service if the scope is a single process, without reusability. That is just a decision activity. This article will help developers and architects design and implement decision services as reusable components in a sustainable architecture.

The use of a business rules management system to support the implementation of a decision service, enables transparency and encapsulation of the business logic within the service layer. It is possible to change this business logic as often as the business
users need to without redeploying any business process or business application. Using WebSphere Operational Decision Management (formerly known as IBM ILOG® JRules), business analysts can maintain the rules themselves.

The business rule lifecycle is very different from an executable business process definition (BPD) or a service orchestration (SCA component, BPEL or mediation flow in ESB). It is possible to change the business rules on a daily basis, which is not thinkable for a business process definition or a reusable service.

As a business service, a decision service should promote reuse for any consumers, not only unique business process applications.

One basic example of a decision service is data validation, in which business requirements constrain the data and the decision is whether to log an issue or reject the business transaction. In claim processing, "validate claim" or "adjudicate claim" are other examples of decision service. Approving a loan or an insurance policy and assessing a fraud are also decision services. Externalized from the core application logic, a decision service is a service provider, consumable by a business process application and by any other applications. Reuse and ease of adaptability are key characteristics of decision services.

Rule processing is data-oriented, and most of the time a large scope of data elements are processed to make a decision. Business process management products, on the other hand, are flow-oriented and carry minimum information between activities. The two technologies are someway orthogonal as you will see in the next section.

From a design point of view IT architects and developers have to address questions such as:

- How do I identify a decision service?
- What kind of decisions is the business is making?
- How do I design the decision service?
- What are the object model needs for the rule processing?
- Should I design my service from the business process using a top-down or a bottom-up approach?
- Is my ruleset signature my service specification?
- Where is the data are coming from?

At the implementation level, using product features blindly, developers may go for a dramatic implementation with poor performance and poor maintainability or a quick-and-dirty solution. This article helps to identify the key factors to design reusable decision service, sustainable over time, by applying, the long term proved, SOA design patterns.
BRMS without BPMS

Decision services may or may not be linked to business process activities. Looking only at a business process model to find decision services will limit the scope of potential dramatic improvement. The majority of decision services deployed in SOA are not directly linked to any automated business process. For at least ten years, the main drivers for BRMS deployment are at the service implementation level, where agility is needed, and changes to the business logic are frequent and are driven by business. These projects illustrate that starting from the business process model should not always be considered the way to go! Analyzing existing business service operations can lead to quickly identifying decision operations that you may want to modernize using BRMS technology. This follows a more traditional SOA approach where service identification, specification and implementation are done to address reusability, adaptability, and change management needs. Therefore, it's important to make the distinction that a decision service is a service not a product feature. In addition, the tendency to group the BRMS and BPMS products under the same umbrella may confuse the industry. First the lifecycle of the elements they manage is very different: a few hours or a day for business rules, but multiple weeks or months for a process definition. Embedding rules in processes forces you to change the process definition each time you change the business rule, and migrate all the active process instances. Secondly, very often the business process owner is different from business rule owners. A process with multiple decision activities may have multiple department owners of the decision logic: claim processors, fraud assessors, and adjudicators are different subject matter expert groups than the claim processing business process owner.

Business rules in the scope of a process route control flow, making decisions about which paths of the process to follow. These types of rules are only one sub-classification of behavioral business rules as defined by OMG. A BPMN gateway is not a decision service. The activity before the gateway is more of interest in discovering the decision logic or rules, and to assess the need to externalize business rules.

Figure 1. Different derivations of business rules
Business rules about inference, computation, derivation, constraint, and event processing are the categories that each need a separate BRMS product. Routing rules using a BRMS, especially an embedded rule engine, is not that valuable.

IBM Business Process Manager and WebSphere Operational Decision Management are complementary products, addressing different scope and needs. In the context of this article, let's start by discussing how to identify a decision task within the business process model.

**Identifying decision services**

When doing business process modeling, using IBM Blueworks Live for example, business process analysts can identify activities within the process definition that participate in decisions. Most of the time those decisions are made by humans or by some piece of software supporting the business logic. At this stage of process discovery, the agile business rule development methodology (refer to [Eclipse Process Framework Project (EPF)](https://www.eclipse.org/epf/) or [Agile Business Rule Development](https://www.ibm.com/developerworks/)) recommends naming those activities *decision points*. Decision points are points of variability and decision in a business process that group together all potential business rules that determine decisions. A decision service is the reusable implementation of one or more decision points. Most of the time there will be a one-to-one mapping, but a decision service can include multiple decision points, such as when the data model to process the decisions is the same, and the activities in the process are sequential and could be grouped in one activity. For example, in a claim processing business process *Validate claim* and *Verify coverage* are decision points that may be mapped to one unique decision service because once the claim is valid the process goes directly to verifying the coverage:

**Figure 2. Decision points in the BPD**

To identify a decision point, you need to look at the activity description within a process definition and search for a verb expressing a decision, such as "assess," "validate," "compute," "verify," "choose," and so on. Typical decision points are
things like loan eligibility, underwrite insurance policy, verify retirement plan, review application, compute price, or validate claim. As described earlier, a BPMN gateway within a business process diagram is not a decision service. It is rather a control flow branching mechanism to route the sequence flow within a process. Most of the time, the outcome of a decision service may be used as input to a decision gateway.

When doing process modeling, discussion with the subject matter experts should lead to an understanding of the scope of the decision; for example, are we talking about less than ten rules or more than that? Who owns the logic behind the decision? How often does this logic change over time? What is the lifecycle of such a change? Is the information model complete at this stage of the process?

Normally the business process instance carries minimum information between activities -- key information within the business entities it manages plus a set of data elements it may have gathered in the different user interfaces executed so far. Making decisions may involve a richer data set. For example, to validate a claim, you may need to look at the insurance policy. The historical records attached to the claim, like medical invoices, may be claimant history too.

To become a decision service, the decision activity needs to have reusable logic with clear input and output. Process routing rules that are making local routing decisions do not represent reusable services. They are simply a group of rules completing externalized process variables to help parameterize the process behavior. Most of the time, the number of process routing rules is very limited. Few rules are defined and they work at the process variables level -- no rich data model, no complex decision, no reuse by other applications, no external rule lifecycle needs, very limited need for change, and static by nature. These are better implemented as Javascript™ in an IBM Business Process Manager process application.

Identifying the decision points helps project managers by enabling them to plan for rules discovery, analysis and implementation, most likely using resources other than the business process implementation activities.

To really get the value of a BRMS, it's important to understand where the current business rules are defined: who is triggering the change to the rules, who implements them, and who owns and defines the logic. It's also very important to gather metadata about the rule to do better governance.

**Decision services are business services**

A decision service should be considered as a part of the business services. When using the IBM SOA reference architecture to classify the different type of services, a decision service should be considered part of the business service group. First, it is a service because it is a "logical representation of a repeatable activity that has a specified outcome" (as defined by the Open Group SOA Work Group); secondly, it is a business because it is owned, defined and managed by members of business
units, but still deployed and administrated by IT. Adjudication of a claim is a good example of such a decision service: business defines the business policies on how to adjudicate a claim, how to compute a deductible, what needs to be paid or not, and the service is exposed as a reusable component within the SOA.

**Figure 3. SOA reference architecture**

Most of the decisions apply to the business entities that business users want to control. These could be claims, purchase orders, trouble tickets, service requests, or bank transactions which by design have their own lifecycle. The decision service participates in this life cycle by providing guidelines and decisions the consumers may act on.

The decision service is usually not a standalone component; it needs to be part of a larger scoped component. For example, `AdjudicateClaim` is a synchronous operation within a `ClaimManagerService`, `underwriteLoan` is an operation within `LoanManagerService`, like `rateApplicant`, `saveLoan`, `getLoan` operations. The input includes at least the main business entity, while the output can be the same entity enriched with the decision taken or a dedicated result object. For example, for data validation a list of issues can be reported as a response.

To comply with SOA best practices, the design of a decision service needs to apply the same pattern as loosely coupling, coarse grained operation, and minimum payload between consumer and provider, as described below:

- Coupling or dependency is the degree to which each program module relies on each of the other modules. Service interface specifications should be developed with minimal assumptions about both the provider and the consumer. They even include different characteristics: one from the consumer point of view and one from the producer scope. To be loosely coupled, change in one application or module should not force a change in another application or module. Following are some different coupling factors:
• Time: Availability of one system does not affect the other. This coupling can be efficiently supported by the use of an ESB.
• Data model: Differences in data models do not prevent integration. This could be better supported by the use of an ETL process or a mediation flow.
• Implementation: The service implementation is hidden from the caller, which means, for example, that the consumer does not know the service is using rule technology, and should not specify the rule set version it wants to use, or get the rule names of the rule executed.
• For coarse-grained operation, the service operation should provide significant business usefulness. It usually includes a composition of finer grain services, in particular data services. A business rule service renders significant business decisions that require complex processing on a large number of data elements.
• To minimize payload, the information passed into a service should be reduced to the minimum, including key data elements. The driving factors are performance, dependencies between components, reusability and lighter change management impact.

The responsibility of acquiring the data should be on the service. The service should not ask the consumer to provide all the data it needs. This is especially true for any reference data needed by the service. It's also true when using BPM as a workflow engine where human-centric processes carry minimum information. IBM Business Process Manager Advanced also supports service orchestration and mediation engines that combine fine-grained services to prepare data for decision services.

Service design considerations

As described earlier, a decision service is an operation within a larger component. The design needs to address the typical interface characteristics as extensively documented by Brian Petrini and Kim Clark in their two recent articles (see Resources). For the decision service, the interface characteristics may include at least the attributes in the table below. The example is for an "underwrite loan" decision service:

**Table 1. Interface characteristics for an underwrite loan decision service**

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Functional definition</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Principal data object</td>
<td>Main business entity served or used by the service</td>
<td>Loan application</td>
</tr>
<tr>
<td>Operation</td>
<td>Describe the action offered by the service. Use a verb + noun pair if possible.</td>
<td>assessLoanEligibility</td>
</tr>
<tr>
<td>Read / change</td>
<td>Are data elements changed as a result of the operation</td>
<td>Change. Potentially a list of reasons for rejecting the application or, in case of acceptance, the potential conditions on the loan.</td>
</tr>
</tbody>
</table>
### Request-response

- **What data do you need to send, and what will you get in return?**
  - LoanApplication as request-response.
  - LoanApplication includes property, borrowers, financial information.

### Technical interface

<table>
<thead>
<tr>
<th>Transport</th>
<th>Medium used by the interface to transfer data from requester to provider and back.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol</td>
<td>Agreement between requester and provider about how and when they will send each other data over the transport.</td>
</tr>
<tr>
<td>Data format</td>
<td>Wire format of the data</td>
</tr>
<tr>
<td></td>
<td>XML document UTF-8</td>
</tr>
</tbody>
</table>

### Interaction type

<table>
<thead>
<tr>
<th>Interaction type</th>
<th>How the consumer will interact with the service. The type could be fire-forget or request-response blocking transport, request-response non-blocking transport, request with acknowledgement, callback, and so on.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch or individual</td>
<td>Individual business action or group of actions processed together?</td>
</tr>
<tr>
<td>Message size</td>
<td>Size of a message accepted by the provider to meet its SLA.</td>
</tr>
<tr>
<td></td>
<td>1 MB</td>
</tr>
</tbody>
</table>

### Performance

| Response times | Duration between a request made and the receipt of a response time via this interface. |
|               | <800 ms average. Range from 300ms to 1.5s. |

| Throughput | Number of requests that can be processed over a relevant period of time. |
|           | 20 |

| Volumes | Maximum capacity over a long time period. |
|         | 2 million applications per month. |

| Concurrency | Number of requests that can be processed at the same time. |
|             | Expect 3 to 6 concurrent requests at the same time. |

### Integrity

| Validation | What data validation rules must a requester adhere to for a request to be accepted? |
|           | A validation ruleset is executed before this operation. Also the constraints on the model ensure a level of control over the required field. |

| Transactionality | Define the level of transactional support the interface provides. |
|                 | None |

| Statefulness | State set-up required before performing a sequence of calls. Should the service keep states between invocations? |
|             | Stateless |

| Event sequence | Order of request |
|               | None |

| Idempotence | If the same request is made twice, will it achieve the same result? |
|            | The same result for the same ruleset version, which means no guarantee over time. For this example, there is no constraint to ensure a loan application may have the same response over time. |

### Security

| Identity/authentication | Ensure agreed users and systems are able to make requests. May be |
|                        | None |
Authorization

Authority to perform the requested action.

Open / no control

Privacy

Who can see the data that is sent or received via the interface?

None specified so far.

Ownership

Who owns the data touched by the operation?

Loan underwriter department for the business logic. Enterprise Architect team for the RBO.

Reliability

Availability

How often and for how long do unexpected outages occur?

Service available 12 hours a day, 7 days a week. Office hours.

Delivery assurance

Confidence that a request is processed.

Web service does not support WS-Transactionality, no delivery assurance.

Error handling

Error management capabilities

If errors occur when invoking the interface, how are they handled, and by whom?

Errors reported to the requester through the fault.

Known exception conditions

Errors understood at design time and therefore can expect to be meaningfully reported back to the caller.

Business errors are reported as issues attached to the loan application.

Unexpected error presentation

How will unexpected errors be presented to the requester?

Unexpected errors are not caught by the ruleset.

Information model considerations

One of the main design considerations when working with a business rule application is to look at the information model. Often called a fact model by knowledge engineers, the information model for rule processing differs from the process variable definition within IBM Business Process Manager in that it is built from the analysis of the different business rules, whereas process variables are designed to gather data from Coaches and to carry data between activities. Rule definitions are driving the data model needed for the rule processing. When declaring business policies, business users use a vocabulary that may be represented by a conceptual data model (CDM). Using products like Rational® Software Architect, you can model the conceptual data model using simple UML class diagrams or sketch diagrams. The goal is to communicate with the business analysts so that the diagram includes high-level entity descriptions and the relationships between them. Figure 3 presents a CDM for a claim processing application built by doing the analysis of business rules such as the following:

A claim must have a data of loss set between the effective date and expiration date of the active insurance policy.

When there are injured persons, all the medical invoices attached to the claim should have a data of servicing within a year after the accident (date of loss)
To write efficient rules using WebSphere Operational Decision Manager, developers may apply object-oriented modeling best practices. Therefore, the design activity includes transforming the conceptual data model to a logical data model (LDM) from which XSDs or Java models are generated. To support a better rule writing experience (and incidentally better rule execution performance) in WebSphere Operational Decision Management, I recommend using the Java language as the execution model. This enables me to add behavioral methods to facilitate rule writing and processing, use inheritance to simplify the model, use encapsulation to support specific settings, add factory methods (to create an instance attached to collection, for example), add reusable features like sorting collection methods, computation, local data caching, and so forth.

It's also important to note that the information model used for the rules has to be adapted constantly during the implementation of the ruleset. Having an independent model helps with frequently adapting the model and in a more effective manner without impacting consumers.

From a methodology point of view, to clearly communicate to the project stakeholders I use the term Rule Business Object for this specific information model, or RBO. This helps to differentiate it from the Service Implementation Model, from the process variables in BPM, from a canonical model at the messaging level and from the Service Exposition Model.
A good architecture leverages different information model views to carry the data to the different layers of the business application: the graphical user interface, the database, the service contract, the business process data, the messaging structure, and the business rules. There is nothing new in this. BPM and Java EE applications have been using this pattern for some years.

In Figure 5 the business objects created by integrating external application are called Application Specific Business Objects, or ASBOs. When the back-end system is updated, the ASBO definitions need to change, too. The ASBOs do not represent the service data type or the RBO. With IBM Business Process Manager Advanced, you can leverage the set of existing connectors (WebSphere Business Integration heritage) to retrieve the information model from an external enterprise system and build ASBO as a set of XSDs.

The Service Exposition Model is externalized within the service interface definition, a WSDL file in the IBM Business Process Manager Advanced Library. On the opposite side, a Service Implementation Model (SIM) is defined internally to the module supporting the service implementation; its scope is the service domain.

This encapsulation helps to adapt the SIM from an implementation perspective, in which the model is stable for the consumers. Finally the ASBO is not the SIM, so you should avoid writing logic against the ASBO; instead write it against the more stable SIM.
As an example, suppose you have a "claimdb" database persisting claims, insurance policies, insured persons, and so on. Using IBM Business Process Manager Integration Designer, you can use the outbound JDBC adapter to directly get the ASBO definitions from the technical services. The same applies if we use an outbound web service. The XSD definitions are part of the module using this JDBC adapter.

For the messaging definition, it's a good practice to apply the canonical data model pattern to limit the number of mappings between the different data models to one between the consumer and the canonical model, and one between the canonical model and the service producer. You should use SEM as the source for the canonical model.

From a decision management point of view, RBO is used as a WebSphere Operational Decision Management eXecutable Object Model (or XOM). It's a good practice to separate this XOM into a separate Java project, packaged in a JAR and easily unit tested. When using XSDs, it's interesting to use an XSD to Java compiler (such as provided by JAXB) to generate JAVA beans. As mentioned above, I prefer using Java as the underlying programming language, rather than XSDs, to enable adding efficient behavior to the business objects, and not to use BOs to simply carry data elements.

The development of the RBO could follow multiple design and implementation paths depending of the team practices, such as UML to Java, UML to XSD then XSD to Java. When using IBM Business Process Manager Advanced, the information model can be defined in Integration Designer using the business objects editor.

---

Figure 6. SEM – SIM – ASBO - RBO
The XSDs are used as input for the XSD to Java compiler available with JAXB. The generated Java beans, as well as other Java classes, are used in the RBO Java project in Decision Designer.

Before concluding our discussion of the information model, let's touch on a very important element of the rule processing -- access to reference data, like enumeration values or list of values. When the enumerated values are static they can be defined as enumeration in XSDs, be part of the BOM, and this be easily accessible by the rule vocabulary. When reference data is dynamic, values can be added frequently over time to support new business requirements. The BOM layer can be used for that, but there are cases where such dynamic data is in a service. A rule condition searching whether a code is known, as shown in the following snippet, may use such service. Local caching is becoming very important to avoid doing calls to external services from rule conditions or actions.

```java
if 
    ... 
    the medical procedure code is known
```

The implementation at the executable model of the previous condition may be supported by a Java method such as:

```java
Public Boolean searchCode(String code)
```

This method uses a cached map of value. The cache is managed inside the application server and follows a dynamic update pattern when new enumerated values are added to the data management (MDM). Reference data influences the choice of using XSD or Java as underlying execution model for rules.

With those best practices in mind, let's move to the decision service design and then implementation.
Designing decision services

A decision service interface should be developed with minimal assumptions about the consumers to enforce the loose-coupling best practice. From a decision service, the major coupling is coming from the information model and the producer should be able to change its data model for the rule processing without impacting the consumers. This can be achieved by designing a request-response schema carrying the most important information the consumer can send and digest. In Figure 8, the claim number is sent instead of the complete data graph with the root element being the claim. The responsibility of acquiring the data should be on the service implementation side, where there are capabilities to use caching and eager loading fetching strategies.

The decision service has to be integrated as an operation within a larger component, and this operation should provide significant business usefulness. This component supports other related business operations. For example, "adjudicate claim," "validate claim," "verify coverage" are decision service operations grouped within a claim processing module, as illustrated in Figure 8. This service interface could be part of an IBM Business Process Manager Integration Designer Library, shared between modules and as a toolkit with process applications developed in IBM Business Process Manager Process Designer.

Figure 8. Part of the interface definition for the claim manager component

It's important to note that as of today a WSDL does not provide enough information of what the interface is doing. As illustrated in Table 1, the other metadata about the service is logged in an individual document. Most of the time, the first interface
Design is driven by the first project, but it's still appropriate and recommended to avoid designing a peer-to-peer integration between the business process and the decision service, and to think in terms of creating a long-term reusable service. The goal is to avoid a strongly coupled interface in which only one consumer can send the data to the service producer. Even in the context of a first project, it's important to design for a long-term, sustainable solution and architecture. IBM SOMA methodology proposes a service litmus test to rate the service for its business alignment, composability (an attribute that enables the service to participate in a service composition), and ability to be externalized. The decision service, as any other business service, should pass this test without exception.

It's important to consider other constraints, as well, such as licensing and cost of operations and maintenance. I've often encountered the need to deploy a JRules eXecution Unit, in a central server, sharing resources between applications. This deployment constraint enforces the use of remote calls, and marshaling and unmarshaling data between servers. When possible, it's better to deploy the XU within the service layer. I'll cover how to do that in Part 2 of this article series.

Looking at data ownership should help drive who provides the data to whom. In most new business process implementation with IBM Business Process Manager, new data elements will be gathered from the different entry forms (Coaches). This data has to be saved to a system of record. Most of the time, an existing database is already defined to persist most of the information, but not always. Often there is a need to adapt the underlying physical data model. This is always a sensitive subject because the current database may be used by different applications. Often, because of time constraints and the need for quick return on investment, the project team may choose to add a new database. This must be avoided to avoid adding complexity to the IT landscape and to the management of the data integrity from one database to the other. This architecture decision needs to be aligned with any data governance in place.

As a first data recorder, the business process should not keep the data as part of the process instance or write directly to the new database using JavaScript or JDBC code. Instead, a data service layer needs to be implemented with its own interface, and used by the executable business process. One of the service operations may map the structure of the process variable, as defined in the process, to the existing physical data. Using IBM Integration Designer, the mapping can be done using mediation flow or maps, which is a better approach than using JavaScript in IBM Process Designer.

In the future, if the databases can be merged, the process definition is not changed because the interface stays the same; only the integration implementation changes. A data access layer service is immediately reused by the decision service implementation, so using BPMS and BRMS enforces creating a data access layer.
strategy which then evolves into reusable services. Again, governance is key here to identify and address data ownership and change management policies.

Implementing decision services

As outlined in the previous section, in order to support agile business processes, you want the business process to carry the minimum information between activities. A call to an external decision service should use the business key of any business entities instead of the business entity itself. Most decision services may need historical data to infer decisions; that data is not defined in the Coaches but is saved in a system of records. The decision service implementation is responsible for loading the data needed and building a rich data graph for the rule engine to process. The best practice for rule programming is to gather all the data before calling the rule engine execution. The implementation of this logic involves orchestrating the different data service layer mentioned above, with other services, such as the ones managing reference data. In IBM Business Process Manager Advanced, you can use a BPEL process for that purpose.

Figure 9. Components involved in decision service implementation

Within a WebSphere Operational Decision Management ruleset, calling a data access layer service to access data inside a ruleflow is not recommended, mostly because it’s difficult to efficiently manage exception, timeout and recovery if the call fails. This processing has to be done before the rule execution.

In Figure 9, the components involved include remote data access services for managing the insurance policy (PolicyManagerService), managing the claim with its historical information (ClaimService), and getting personal information (PersonManagerService).

For performance reasons, when using IBM Business Process Manager the decision service implementation has to be done in the service layer as a pure Java SCA component (ClaimDecisionServices). A Java implementation is preferred to support a rapid implementation of the decision service. This approach provides the flexibility to develop and unit test the Rule Business Object model and the rules themselves separately from any integration. The code is a standard JRules RES integration, using RES factory and RES session API.
The code snipped below is implemented in a RuleProcessingImpl class that supports the RBO as information model (claim) and uses the RES API. This code can be wrapped in an EJB or POJO façade and deployed as a web service on the same server as other reusable services or deployed remotely on a server dedicated for rule processing. The approach is flexible.

```java
public ClaimProcessingResult validateClaim(Claim claim) {
    ClaimProcessingResult outResult = null;
    try {
        IlrSessionRequest sessionRequest = prepareSessionRequest(validateClaimRsName);
        // Set the input parameters for the execution of the rules
        Map<String, Object> inputParameters = new HashMap<String, Object>();
        inputParameters.put("validateClaim", claim);
        sessionRequest.setInputParameters(inputParameters);

        // Execute rules
        outResult = executeRule(sessionRequest, "validateResult");
    }...}
```

When using a remote server for all the rule processing, the packaging of the previous code can be done in WAR or EAR and deployed on the WebSphere Application Server running the XU. In Figure 10 MP_DecisionServices is a WAR that supports the different decision services implemented in WebSphere Operational Decision Management. Those decision services were derived during business process discovery and specified in a unique interface.

```java
public interface ClaimProcessingDecisionService {

    public abstract ClaimProcessingResult adjudicateClaim(Claim claim);

    public abstract ClaimProcessingResult validateClaim(Claim claim);

    public abstract ClaimProcessingResult verifyCoverage(Claim claim);

    public abstract ClaimProcessingResult validateMedicalInvoice(MedicalBill medicalInvoice);
}
```
WebSphere Operational Decision Management supports the concept of a hosted transparent decision service, or HTDS, which exposes a ruleset as a web service. A dedicated web application (jrules-res-htds-WAS7.ear), is deployed to the rule execution server, and consumes XML documents. Until JRules 7.1.1, this feature supported only XML documents; therefore, as soon as there is a need to add Java utility classes to support better rule processing, it's better to use your own service, as described earlier. WebSphere Operational Decision Management supports using Java XOM in HTDS. I'll go into more detail on that in Part 2 of this series. Nevertheless, a ruleset exposed as a web service should not be considered a business service but instead represents a method or operation that can be invoked remotely using SOAP over HTTP. The major concern with using HTDS is having the consumer sending all the data elements to HTDS. When using a BPM workflow like a pure IBM Business Process Manager BPD solution, it is recommended that you avoid doing service orchestration and low-level integration in the process definition itself. This logic needs to be part of the service layer. The decision service implements this logic to prepare the data for the rule engine, call the rule execution, and process the results.

In assessing the best implementation choice, it may be interesting to study the different possible use cases or patterns. Table 2 illustrates the most appropriate product features of Business Process Manager and WebSphere Operational Decision Management to support the different use cases:

**Table 2. Product features supporting different use cases**

<table>
<thead>
<tr>
<th>Use case</th>
<th>Business Process Manager capabilities</th>
<th>WebSphere Operational Decision Management capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPD is able to send all the data elements and XML is sufficient to support the information model No reference data outside of XSD enumeration</td>
<td>JRules decision service SOAP client Web service integration Java integration</td>
<td>HTDS RES embedded with XML binding</td>
</tr>
</tbody>
</table>
Rules are reusable cross applications, XML is sufficient, BPD sends partial information
Reference data outside of XSD enumeration in MDM

Advanced integration services
Web service integration
Rule eXecutable Unit deployed in the service layer / local SCA Java component, mediation flow or BPEL component
Decision service in Java
HTDS

Reusable ruleset
Java-based RBO
BPD sends partial information
Reference data outside of XSD enumeration in MDM

Advanced integration services
Web service integration
Decision service in Java
HTDS with Java XOM combined with integration service in ID

Need to validate data at the Coach or activity level
Need to collocate the service within the same JVM as the process engine
Java integration
Rule engine API
Embdded XU

Notes:

1. The first decision factor is linked to the vocabulary and expressiveness of the rules: most business rule applications need efficient vocabulary to expose a simple language for the business analyst to write rules. This implies adding behavioral methods on top of the Java beans representing the information model. It's important to note that WebSphere Operational Decision Management supports adding scripting logic in the BOM to XOM mapping, but when you can avoid using scripts, it is better to use Java.

2. The second factor is to assess whether the BPD is able to get all the data for the rule engine. Most of the process applications are carrying only the data element in the scope of the Coaches or the process activities. The decision service may need a bigger data graph including historical information. The process is not able to or designed to load this data. The service layer should do this work.

3. The third factor is related to reference data management. Most of the business application defines and manages those important elements in a separate component, like an MDM product, and cached within the JVM running the rule engines.

I will describe each of these integration patterns in Part 2 of this series. Of the listed factors, the third use case is most commonly used when designing scalable and sustainable solutions and enforces using the Advance Integration Service feature (AIS).

Conclusion

In summary, it's very important to study the information model in the scope of rule processing, assess who is creating which business entities, and how the data graph is built and sent to the rule engine. You must also study reference data management. Decision services need to integrated in SOA and described using the service interface characteristics for providers. The choice of using XSD or Java may be linked to the technology used; for example, using IBM Business Process Manager
Standard, Java is a better choice so that the service implementation can better manage the data graph needed for rule processing.

In Part 2 of this series, I'll dig into the details of using the IBM Business Process Manager and WebSphere Operational Decision Management features to support the implementation of a decision service integrated with IBM Business Process Manager.
Resources

- **Policies and Rules series**, Hondo, Boyer, Ritchie, developerWorks 2010
- **Capturing and analyzing interface characteristics series** Kim Clark and Brian Petrini, developerWorks 2011, 2012.
- **Design service interfaces with industry models**, Carlo Marcoli (PDF)
- **Making Better Decisions using WebSphere Operational Decision Management**, IBM Redbooks, Duncan Clark and Pierre Berlandier
- **developerWorks BPM zone**: Get the latest technical resources on IBM BPM solutions, including downloads, demos, articles, tutorials, events, webcasts, and more.
- **IBM BPM Journal**: Get the latest articles and columns on BPM solutions in this quarterly journal, also available in both Kindle and PDF versions.
About the author

Jerome Boyer


© Copyright IBM Corporation 2012
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)