

# *Implementing a Rules-Driven Service Oriented Architecture*

A technical white paper by

**Daniel C. Hayes**

[dan@mydogboris.com](mailto:dan@mydogboris.com)

[www.mydogboris.com](http://www.mydogboris.com)

July 2003

|                                                              |    |
|--------------------------------------------------------------|----|
| Executive Summary .....                                      | 2  |
| The “Next Big Thing” .....                                   | 2  |
| Business Process Orchestration.....                          | 3  |
| In This Paper .....                                          | 3  |
| Multi-Risk Insurance Company .....                           | 4  |
| Problem Scenario .....                                       | 4  |
| The Solution Architecture .....                              | 4  |
| The Business Object Model.....                               | 5  |
| Creating the Rules Project in ILOG JRules.....               | 6  |
| Defining the Rule Flow.....                                  | 10 |
| Publishing a Rule Service to a J2EE Application Server ..... | 11 |
| Testing the Service .....                                    | 12 |
| Moving to Web Services.....                                  | 14 |
| Web Services Overview.....                                   | 14 |
| Moving the Quote Request Service to a Web Service .....      | 14 |
| Conclusion .....                                             | 18 |
| About the Author .....                                       | 19 |
| References.....                                              | 20 |

# Executive Summary

## *The “Next Big Thing”*

Service Oriented Architecture (SOA) has been labeled “the next big thing” in software development. Why? Because, according to experts, implementing a service-oriented architecture will help eliminate the need for lengthy and expensive integration projects that are the norm today. By simply wrapping applications in a “service” façade and providing a client interface, our integration efforts may focus on the much simpler task of message transport and assembling of the data required to satisfy the contract of the interface. We don’t need to concern ourselves with the inner workings of the application providing the service.

Sound familiar? This concept could easily be mistaken as nothing more than good object oriented design. The idea of hiding implementation via encapsulation is a premise of O-O design. Even for distributed systems, numerous design patterns have become de facto standards for dealing with the extra layer of complexity provided by the network layer. If these design patterns commonly prevail, then how can service oriented architectures be the *next* big thing?

The reason is simple. Up until recently, issues of cross-platform interoperability and remote object invocation developed into religious debates between proponents of one technology platform versus another. This effectively stymied widespread adoption of any technology for distributed application communication. However over the past few years, the industry has managed to standardize around a small, manageable set of technologies for interoperability and information exchange, namely XML and Web Services. The reasons these standards prevailed is beyond the scope of this paper but many attribute their success to the single fact that no vendor *owned* rights to them. With the establishment of well defined standards [that were being adopted by every major player in enterprise software] a world of cross platform “distributed” application development became possible. This sparked a renaissance in service based application development. Software components, regardless of platform, could fulfill their promise of reuse by exposing an interface to any application that has an interest in the service it could provide. It was now *paying* to design with services in mind. After all, one day the application may be exposed as a web service in order to facilitate some business opportunity or cost cutting motivation.

Enterprise architects have been able to piggy back on these advances in standards and technology to solve complex enterprise integration (EAI) problems. After all, enterprises usually find themselves in the middle of a complex ecosystem of information silos and tangled proprietary applications. In these situations, service oriented “wrappers” could front these legacy applications in order to add years to their useful lives. Having a clear cut set of rules and technologies for accomplishing this simplifies the solution scope and greatly increases the likelihood of a successful project.

## ***Business Process Orchestration***

In most cases, these services strive to provide coarse grained functionality to clients in order to simplify the interface, reduce the need to continuously call the service, and to standardize processes that require the coordination of multiple components. As a result, these services are usually “wrapping” (or hiding) the interplay between several software components. At the very least, the service layers can contain sophisticated rules for processing requests and potentially routing of these requests to various individual components. These coarse grained services have become known as “Business Processes” and the orchestration required within a service as Business Process Management (BPM).

As the sophistication of these services increases so does the effort required to manage them. Frequent changes in business strategy or policies would likely require the re-compilation and deployment of various components. This is error-prone and suffers from a slow cycle time. The business process management community and BPM tool vendors have organized to study and address many of the common problems faced in service orchestration. As a result, business process management tools have emerged to increase the *agility* and manageability of service based applications.

Rules engines are one such tool. Although hardly a new technology, rules engines are a perfect match for coarse grained services as they can increase agility by externalizing the business rules that “orchestrate” a business process. By not having to “hard code” rules in the application, companies can visualize the business process and react more quickly to changing requirements. Rules engines have the added advantage of being able to host the business rules *within* an individual service component. For new component development, the incorporation of this technology becomes a natural strategy when complex business rules are involved.

### ***In This Paper***

In this paper we will implement a simple rules-driven, service oriented application using Java, ILOG JRules, and Apache AXIS (for web services deployment). The business challenge will be for a fictitious insurance company to coordinate several steps of a common business process – handling the request for an automobile insurance quote. The process involves the validation of the request, performing a credit check, determination of eligibility and selecting a rate plan (i.e. underwriting tier). The application will first be implemented as a Java (J2EE) Stateless Session EJB and then ported to a platform neutral web service with an XML interface.

While the application is necessarily trivial, it demonstrates the technologies and techniques that would be used in a larger production application. It will also demonstrate the flexibility of a rules-driven approach to a service based architecture.

# Multi-Risk Insurance Company

## ***Problem Scenario***

Multi-Risk Insurance Company is a fictitious property and casualty insurance company that sells its products via the web and through agents. As part of its ongoing effort to streamline operations and provide better service to customers and agents it has decided to offer rate quotations in real time via the web and through relationships with 3<sup>rd</sup> party comparative rating services.

One important requirement is that there is only one rating application service that serves all clients. In the past, problems arose due to the fact that multiple quoting applications had to exist due the heterogeneous nature of the platforms that hosted these clients. Problems developed in rating discrepancies, workflow inconsistencies and system maintenance. Every attempt should be made to prevent this from occurring again.

The general activity that occurs when a request for a quote is received is as follows. First the request is validated for missing data, invalid entries, etc. Next, a determination is made as to whether a credit score should be run on the applicant based upon business rules. Next, a series of potentially complex rules determine whether an applicant is acceptable or not. Finally, a multi-variable matrix determines what underwriting tier (program) an applicant is eligible for.

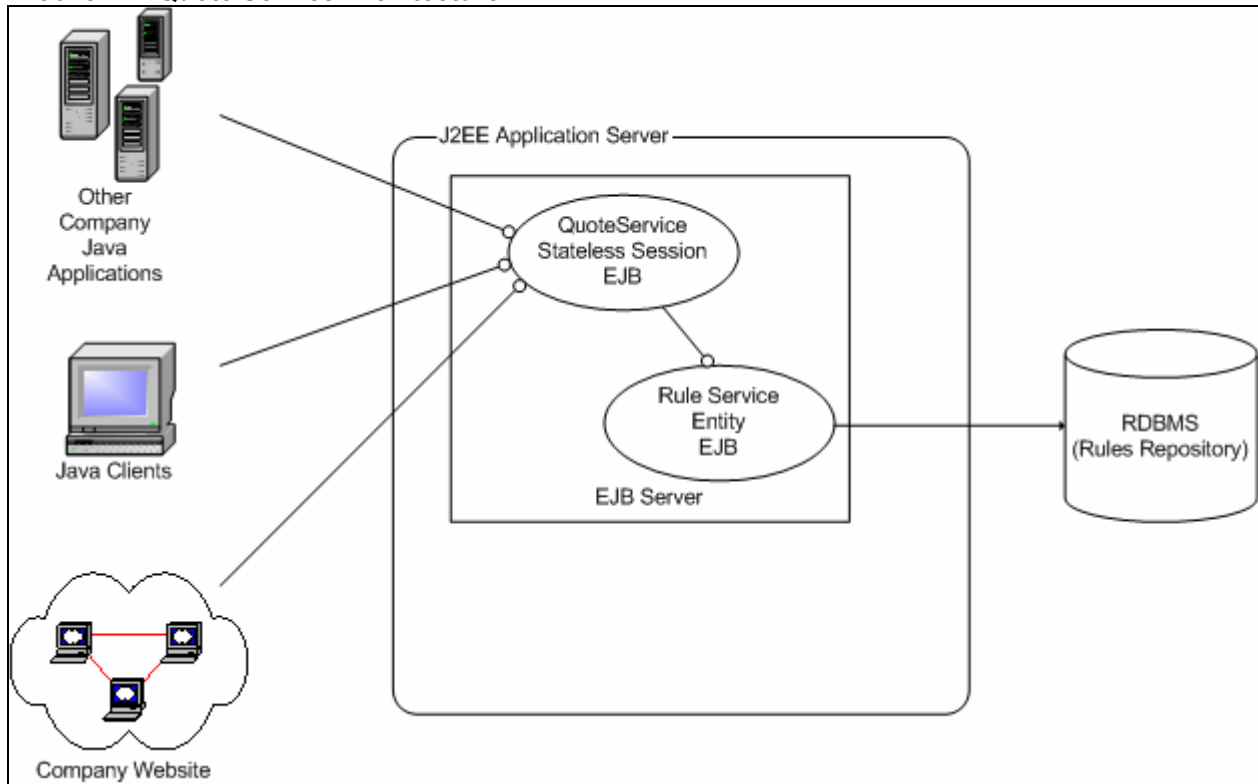
The company's internal architecture for new application development is Java/ J2EE. Accordingly, it would like to build its new applications in Java but doesn't want to exclude the opportunities created by its partnerships with agents or various 3<sup>rd</sup> party rating vendors. Furthermore, the carrier must be able to easily manage the various complex rules both within individual components involved in the quoting process as well as those that manage the orchestration of the various steps in the process.

## ***The Solution Architecture***

The company has decided to use Java to build its application, following the J2EE framework. In addition, ILOG JRules 4.5 will be used to externalize the rules of the various components and orchestrate the process flow. This will provide tremendous flexibility for changing workflow and business rules without the need to recompile and deploy the underlying application.

Initially, the application will be deployed as a service based application within the J2EE platform. As a result, the primary entry point will be a stateless session bean (EJB) deployed on one of the companies J2EE application servers. All Java applications that require access to the quote service will access it through this EJB. This will leverage the transaction and security features built into the J2EE application server. One such client will be the company's website that services consumer with direct rate quotations. This web application will access the service via a Servlet or business delegate. In the future, this service will be exposed as a web service for non-Java clients such as the company's internal CRM application and 3<sup>rd</sup> party rating applications.

## Initial J2EE Quote Service Architecture.



### ***The Business Object Model***

To facilitate our example, we will construct a simple object model that will hold the current state of various attributes related to a request and response. These classes are simple “value objects” and implement the `java.io.Serializable` marker interface. This allows them to be serialized and transported across the wire from client to server. The `QuoteRequest` object will be the input to the quote process and `QuoteResult` will be the output. The `QuoteRequest` holds references to a `Driver` object and a `Vehicle` object in addition to its own credit score and id attributes. The `UnderwritingTier` class represents a rate plan that an applicant will be assigned should the request pass the eligibility criteria. Note that it provides static properties that return instances of the various tiers (prototype design pattern).

In a production application, this object model would be much more sophisticated. However, the purpose of these classes is simply to hold state in order to be serialized and transferred between processes. While the number of attributes may grow tremendously, it is strongly encouraged that these classes remain lightweight value objects. Should more heavyweight objects be required to facilitate the business process (as they likely would), server side representations would be created and instantiated from within the service. In this simple example, we do not create any such objects and will work directly with the value objects.

## The Business Object Model



## Creating the Rules Project in ILOG JRules

ILOG JRules provides an integrated development environment called “Rule Builder” for defining and managing rules projects. In our application, we created a rules project

called QuoteRequest that will contain our business rules and ruleflow. Ruleflow is a new feature of JRules 4.5 and provides a tremendous benefit in rule service orchestration.

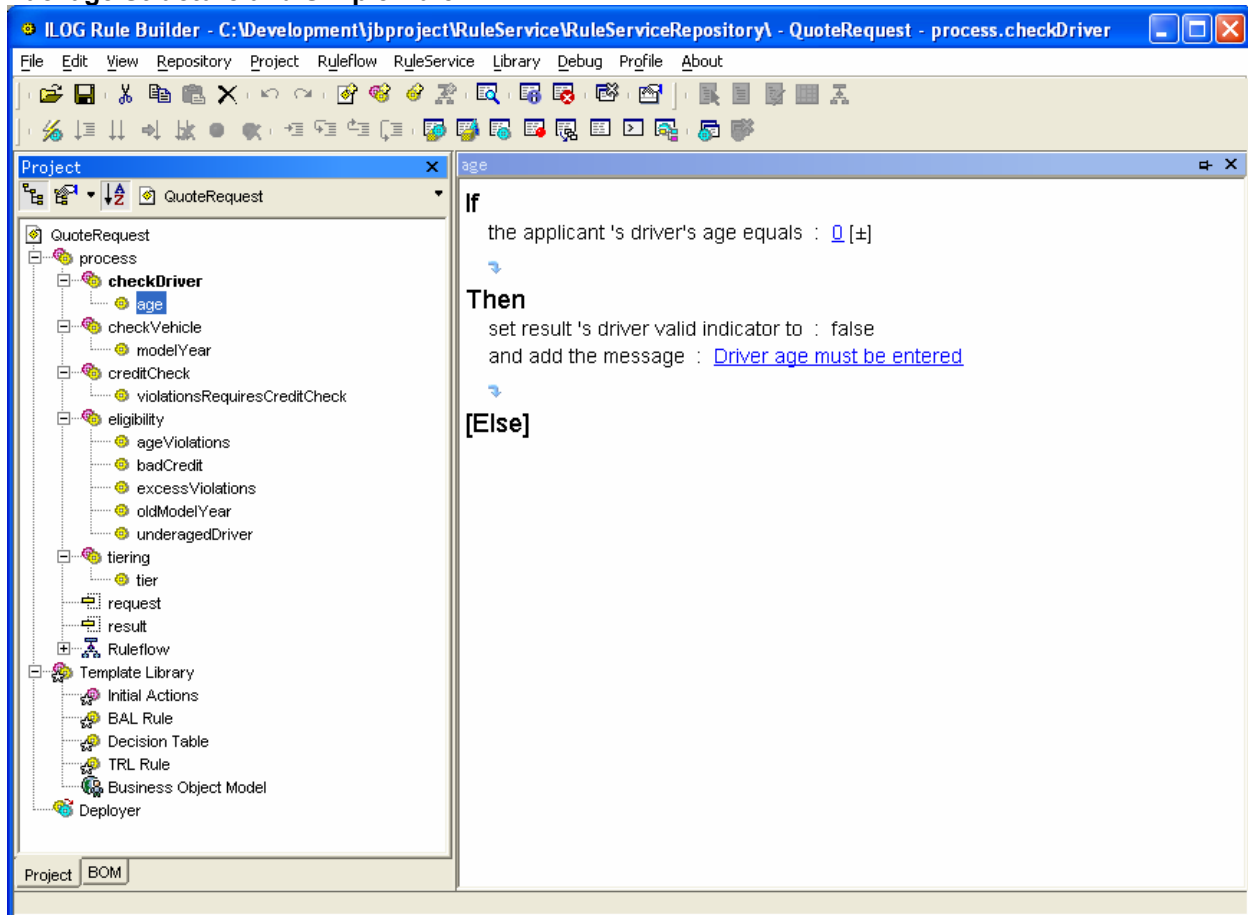
While a detailed description of JRules is beyond the scope of this paper, there are a couple of points worth mentioning. First, a JRules rules project must be aware of Java classes that will be used in the construction of the rules. As such, we must adjust our project deployer settings (Project | Deployer Settings...) to add our model classes to the execution classpath. Then, after importing the classes to the Business Object Model (by right clicking on the Business Object Model node of the Template Library section of the project explorer), we may create rules that reference the properties of object instances. Further, we may assign English-like translations of these properties so that rules may be written in terms that are understandable by less technical personnel. We will see an example of such a rule shortly.

Another important concept in JRules is that rules are organized into packages (much like its Java equivalent). Every rules project must have at least one top level package. In our example, we have a top level package called "process". Within this package, we've organized our rules into various sub-packages called checkDriver, checkVehicle, creditCheck, eligibility, and tiering. These represent the logical grouping of rules that will be fulfilled via our ruleflow. Within a package we create rules that are assigned a logical name. Rules represent the "if-then" conditions that make up a company's business policy.

Rules may be written in a technical rule language or a Business Action Language (BAL) depending on the preferences of the rule creator. In this case our rules were written using BAL. In the checkDriver package, there is a single rule that checks for a valid driver age (i.e. not equal to 0, the initialization value of a Java int type). If the driver age is 0, then the driverValid flag on the QuoteRequest object is set to false and a message is added (via the Java method "addMessage" as per the signature).

Having a single rule in a package that will be executed as part of our ruleflow is hardly a justifiable reason to introduce a technology as powerful as a rules engine. However, as the number of rules grows into the hundreds, the power and manageability of this approach becomes clear. Not only do the rules become more manageable via the IDE, but sophisticated RETE algorithms will automatically optimize the order in which various rules are placed on the "agenda" and "fired" so as to minimize processing requirements and increase speed. If 50 rules exist to validate a driver, many of which are based on complex predicate logic, the performance improvements over a traditional approach (not to mention manageability) become obvious.

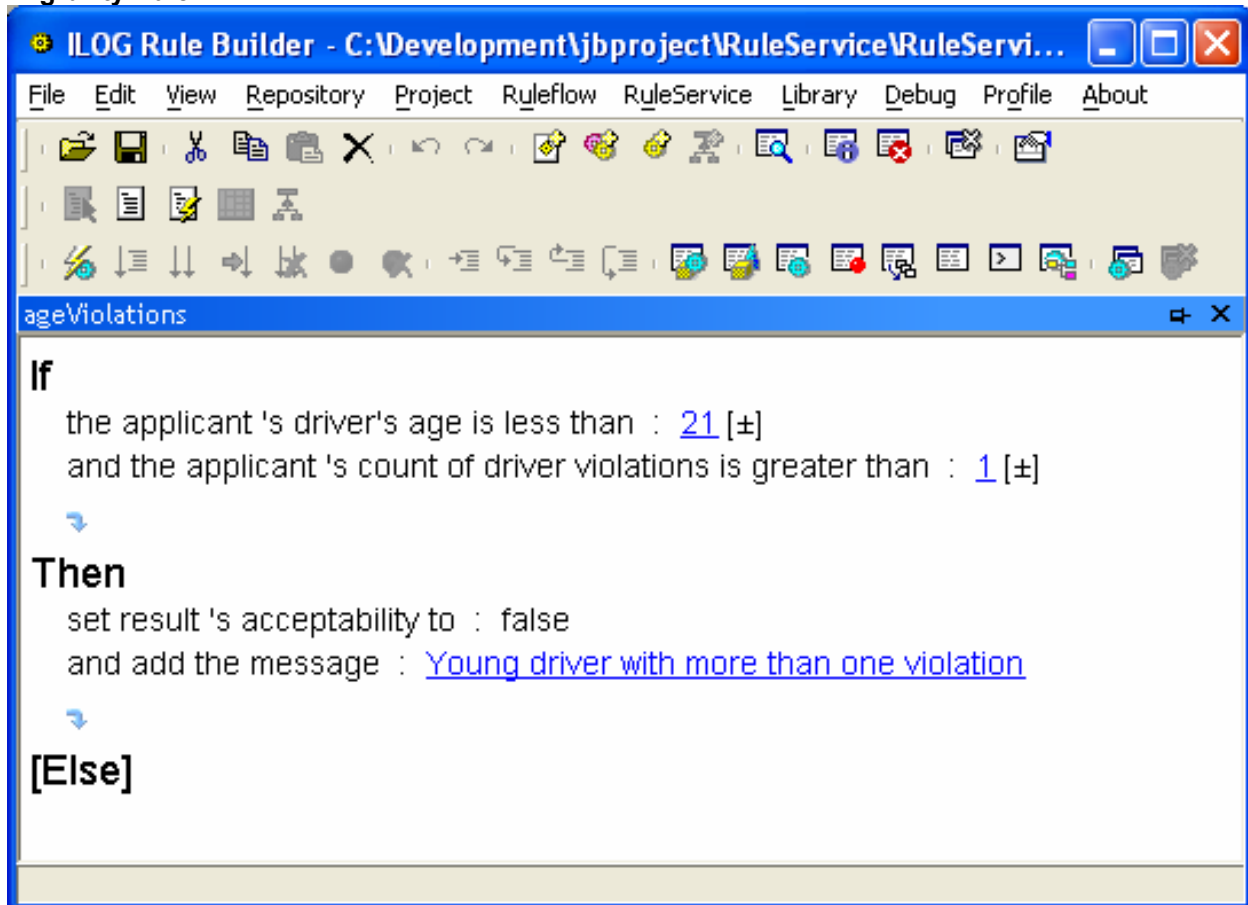
## Package Structure and Simple Rule



In the eligibility package, we have a number of rules to determine if the application represented by the QuoteRequest is eligible. One such rule is “ageViolations”. This rule states that if the driver’s age is less than 21 and there is more than one violation, then the applicant is not eligible. It also adds the relevant message to the QuoteResult. This is an example of a rule that is based upon two predicate conditions.

Rules can get extremely complex and the actions of rules can instantiate new objects, call methods on those objects, etc. For example, we assumed that credit score was sent in as part of the QuoteRequest. In the checkCredit package we have a rule that determines whether credit is relevant to the process. Later, in the eligibility package, a rule called “bad credit” checks whether credit is relevant (via the flag) and then declines those with a score below the threshold. In reality, the checkCredit package rule would likely update the QuoteRequest creditScore attribute with the actual score obtained from calling a method on a heavier weight CreditScoring object in the action portion of the rule. The CreditScoring object, visible only to the server, would perform the responsibilities of actually ordering credit. More interestingly, instead of accessing a simple Java object, the action portion of the rule could actually call another “credit ordering” rules service with its own orchestration and rules. The possibilities are tremendous.

## Eligibility Rule



The tiering rule package has a single rule of the type “Decision Table”. Another new feature of JRules 4.5, decision tables provide a convenient representation of rules that are better visualized via a table. In our example, we will use driver age, violations, and performance type of vehicle to determine which rating plan (underwriting tier) the applicant receives. The underwriting tier holds a factor that will be applied to the base premium to compute the annual premium.

## Decision Table

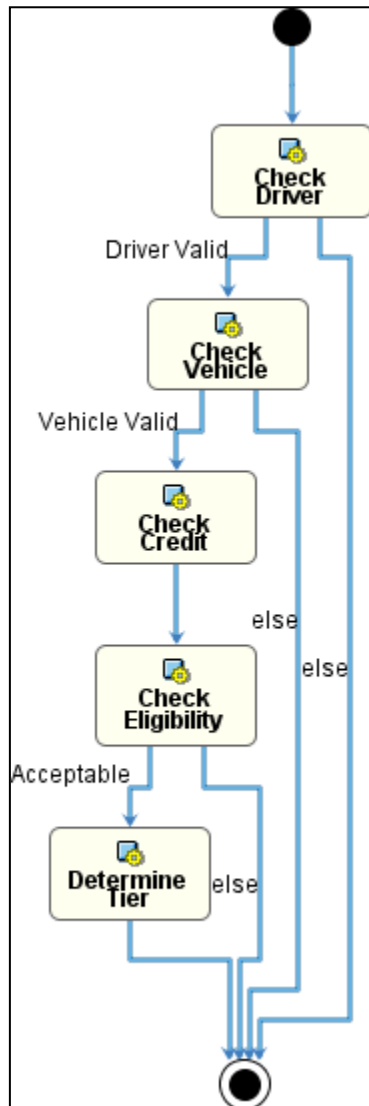
The screenshot shows the ILOG Rule Builder interface with a decision table titled 'tier'. The table has four columns: 'driverAge', 'driverViolations', 'vehiclePerfor...', and 'setUnderwritingTier'. The data is organized into three main age groups, each with sub-categories for violations and vehicle performance levels.

| driverAge   | driverViolations | vehiclePerfor... | setUnderwritingTier |
|-------------|------------------|------------------|---------------------|
| 16 ≤ v ≤ 20 | Clean            | STANDARD         | Non Standard        |
|             |                  | INTERMEDIATE     | Non Standard        |
|             |                  | HIGH             | Non Standard        |
|             |                  | SPORTS           | Non Standard        |
|             | 1                | STANDARD         | Non Standard        |
|             |                  | INTERMEDIATE     | Non Standard        |
|             |                  | HIGH             | Non Standard        |
|             |                  | SPORTS           | Non Standard        |
|             | 2                | STANDARD         | Non Standard        |
|             |                  | INTERMEDIATE     | Non Standard        |
|             |                  | HIGH             | Non Standard        |
|             |                  | SPORTS           | Non Standard        |
| 21 ≤ v ≤ 29 | Clean            | STANDARD         | Standard            |
|             |                  | INTERMEDIATE     | Standard            |
|             |                  | HIGH             | Standard            |
|             |                  | SPORTS           | Non Standard        |
|             | 1                | STANDARD         | Standard            |
|             |                  | INTERMEDIATE     | Non Standard        |
|             |                  | HIGH             | Non Standard        |
|             |                  | SPORTS           | Non Standard        |
|             | 2                | STANDARD         | Non Standard        |
|             |                  | INTERMEDIATE     | Non Standard        |
|             |                  | HIGH             | Non Standard        |
|             |                  | SPORTS           | Non Standard        |
| 30 ≤ v ≤ 64 | Clean            | STANDARD         | Preferred           |
|             |                  | INTERMEDIATE     | Preferred           |
|             |                  | HIGH             | Standard            |
|             |                  | SPORTS           | Standard            |
|             | 1                | STANDARD         | Preferred           |
|             |                  | INTERMEDIATE     | Standard            |
|             |                  | HIGH             | Non Standard        |

## Defining the Rule Flow

Rule flows provide an orchestration of our service and thus make up a Business Process. JRules provides a visual aid, in the form of an activity diagram, to help design our business process. We will design the process around a series of tasks that correlate to our rule packages. Individual rules or entire rule packages can be assigned to a rule task (we assigned rule packages to tasks to improve understandability). First

we check the validity of the driver and if it is not valid, we end processing and the QuoteResult is returned. However, if everything is correct we move on to checking the vehicle. Once again, if everything proves okay we execute the Check Credit and Check Eligibility steps. If the applicant remains eligible, the decision tree rule in the tiering package is executed and a rate plan is selected.



Rules that define the if-else conditions in the branching are called “guards”. Decision nodes with more sophisticated branching can be added as well. The ruleflow guards may access any object we define as parameters on the rule project. These parameters may also be characterized as either input, output, in/out, or local parameters. Defining these parameters correctly is important for proper publishing of our rule service.

Rules that define the if-else conditions in the branching are called “guards”. Decision nodes with more sophisticated branching can be added as well. The ruleflow guards may access any object we define as parameters on the rule project. These parameters may also be characterized as either input, output, in/out, or local parameters. Defining these parameters correctly is important for proper publishing of our rule service.

### ***Publishing a Rule Service to a J2EE Application Server***

Now that we have our business rules written and our process orchestrated, we begin the task of publishing our rule service to our J2EE application server. Fortunately, ILOG once again provides some helpful tools.

From the Rule Service menu in Builder, we can select “Extract Rule Service”. This will take us to a Rule Service Explorer pane where we can confirm that our top level package is identified along with our input and output parameters. Our rule service will expect an instance of a QuoteRequest object and return an instance of a QuoteResult. The top level package will become the method name of the service (i.e. “process”).

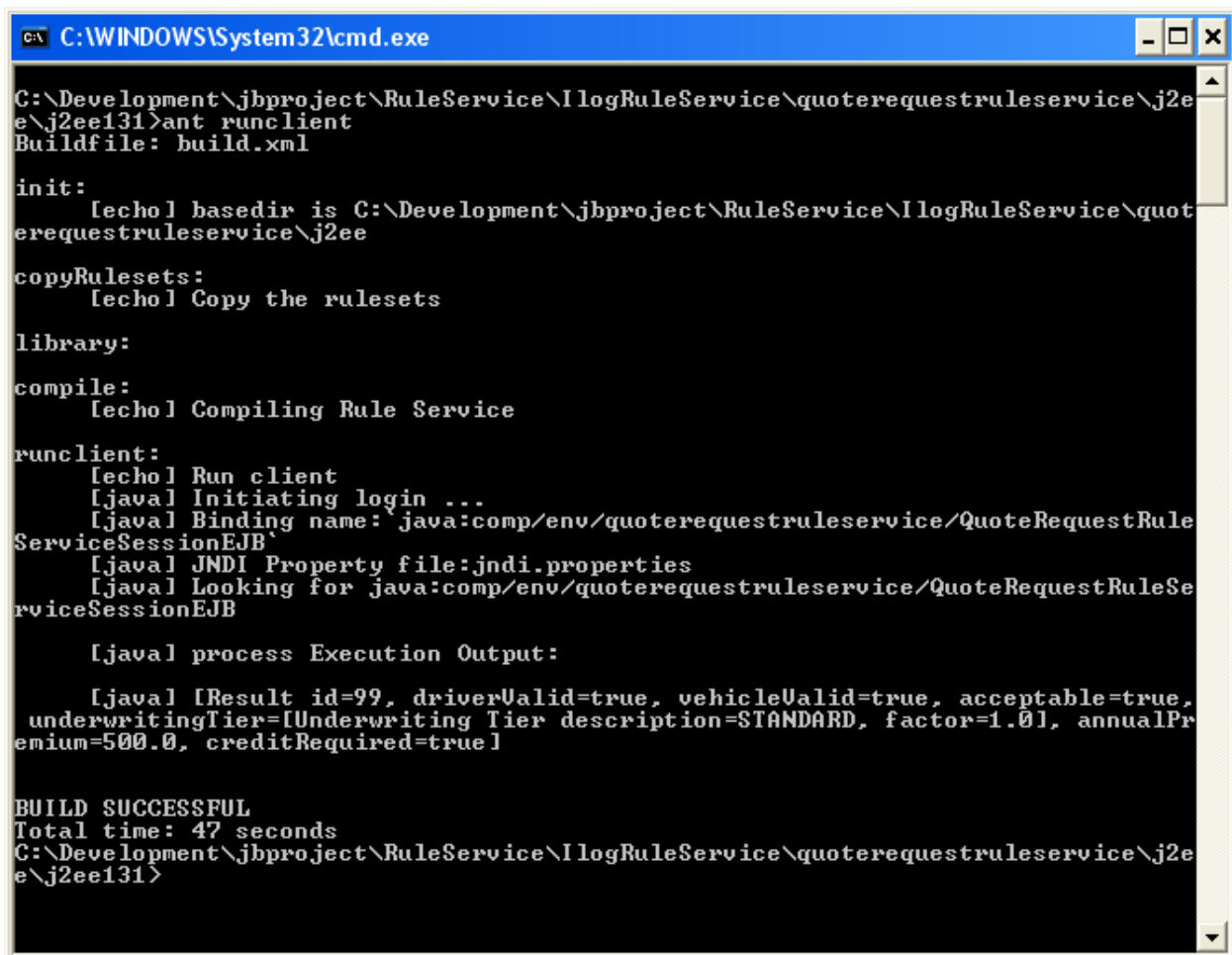
In our first implementation, the company will publish a J2EE service to serve other Java client applications. We add a J2EE target by right clicking on the rule service and selecting it from the menu. In the properties settings for this target we can set the destination location for the generated files, and various server parameters. When we publish rulesets and code, ILOG will generate a series of directories and files to the specified destination. In a J2EE publication, those files represent Java code for Stateless Session Beans, an Entity Bean to manage the persistence of the ruleset in a data store, a test java client, EJB deployment descriptors, and ANT build scripts. Even more exciting, these files are generated with the application server specific deployment artifacts for Oracle 9iAS, IBM Websphere 5.0, BEA Weblogic 6.1 and the Sun reference implementation included with the J2EE SDK distribution.

Deploying the service becomes a simple process of executing a series of ANT build scripts to prepare your database, compile and build the EAR file (top level J2EE deployment artifact) and adding the EAR to the running J2EE server.

## Testing the Service

The ILOG code generation steps create a sample test client that can be executed from the build scripts. After successfully deploying our rule service to a running instance of a J2EE server, we can execute the “runclient” task of the ANT build script. The test client creates an instance of the QuoteRequest object, connects to the JNDI context of the J2EE server on a specified port, finds the home interface of the rule service stateless session bean and creates an instance of the remote interface (proxy). It then invokes the “process” method on the interface and outputs the results from the QuoteResult object returned (via the overridden “toString” method).

In this example we modified the test client java code generated by ILOG to instantiate an appropriately configured QuoteRequest object. The following is the output generated as a result of calling the rule service. As you can see, this request was validated, approved, and assigned to the STANDARD underwriting tier. Also, the credit required indicator was set to true since the driver has a violation (and all drivers with a violation require a credit check as per the business rules).



```
C:\WINDOWS\System32\cmd.exe
C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee\j2ee131>ant runclient
Buildfile: build.xml

init:
[echo] basedir is C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee

copyRulesets:
[echo] Copy the rulesets

library:

compile:
[echo] Compiling Rule Service

runclient:
[echo] Run client
[java] Initiating login ...
[java] Binding name: java:comp/env/quoterequestruleservice/QuoteRequestRuleServiceSessionEJB
[java] JNDI Property file:jndi.properties
[java] Looking for java:comp/env/quoterequestruleservice/QuoteRequestRuleServiceSessionEJB

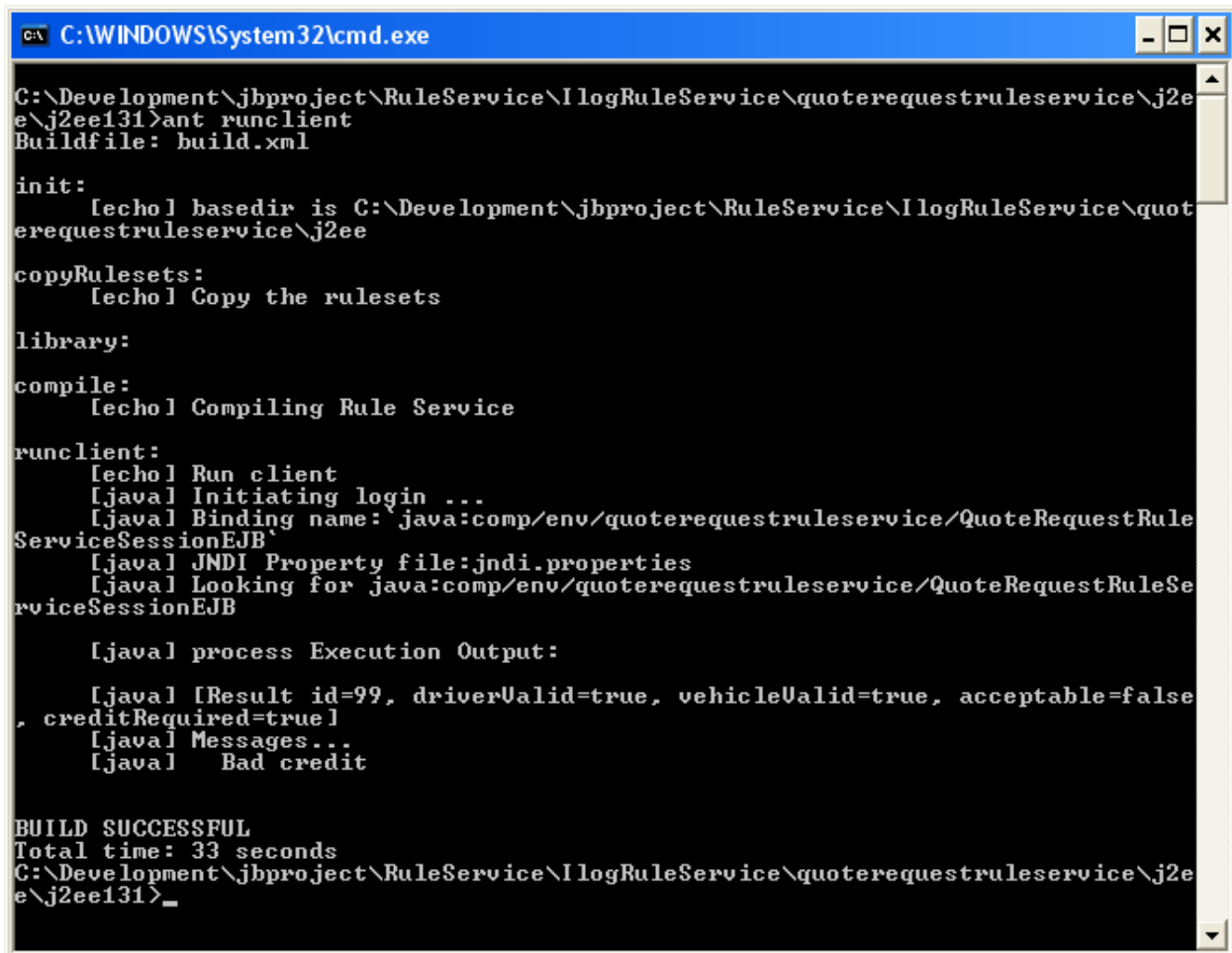
[java] process Execution Output:

[java] [Result id=99, driverValid=true, vehicleValid=true, acceptable=true, underwritingTier=[Underwriting Tier description=STANDARD, factor=1.0], annualPremium=500.0, creditRequired=true]

BUILD SUCCESSFUL
Total time: 47 seconds
C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee\j2ee131>
```

To make a change to the rules we do not need to duplicate each of these steps. We will change the bad credit rule to decline any one with a score lower than 400. In our

example, the applicant has a score of 350 [Note that credit score is NOT evaluated if the driver is clean. However, in this case, our driver has one violation so credit is relevant]. After making the change in Builder, simply right click the rule service in the rule service explorer to “publish rulesets only”. This will re-create the rule file. Then simply run the ANT task “update” to republish the changes to the server. There is no need to recompile classes or redeploy any components. When we run the test client this time we receive the following output:



```
C:\WINDOWS\System32\cmd.exe
C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee\j2ee131>ant runclient
Buildfile: build.xml

init:
    [echo] basedir is C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee

copyRulesets:
    [echo] Copy the rulesets

library:

compile:
    [echo] Compiling Rule Service

runclient:
    [echo] Run client
    [java] Initiating login ...
    [java] Binding name: java:comp/env/quoterequestruleservice/QuoteRequestRuleServiceSessionEJB
    [java] JNDI Property file:jndi.properties
    [java] Looking for java:comp/env/quoterequestruleservice/QuoteRequestRuleServiceSessionEJB

    [java] process Execution Output:

    [java] [Result id=99, driverValid=true, vehicleValid=true, acceptable=false, creditRequired=true]
    [java] Messages...
    [java]   Bad credit

BUILD SUCCESSFUL
Total time: 33 seconds
C:\Development\jbproject\RuleService\IlogRuleService\quoterequestruleservice\j2ee\j2ee131>_
```

The applicant is rejected and the reason is given as “Bad credit”.

The flexibility of this approach to a service-oriented architecture is obvious. Without re-compiling a single Java file, we were able to change our business rules and publish them to our rule service application. We could have just as easily made significant revisions to our orchestrated workflow. Now all our Java clients have an extremely easy and “agile” service to call for processing a quote request.

# Moving to Web Services

## **Web Services Overview**

According to the World Wide Web Consortium:

*“A Web Service is a software application identified by a URI, whose interface and bindings are capable of being identified, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols”*

This statement simply describes a web service as an application that exposes a function that is accessible using standard Web technology and that adheres to Web services standards. Those standards, maintained by the W3C, help applications from one platform communicate with an application from another platform. By defining a lingua franca (in XML), these application no longer need to concern themselves with complex cross platform type mapping (such as CORBA-IDL) or object conversion techniques. The web service standards have added an extra layer of abstraction to absolve us of this responsibility.

Web service technologies [and standards] are concerned with four primary areas:

1. Service Transport – the transport layer used to transfer data from one machine to another. While not specifically limited to HTTP, it is the most common mechanism due to firewall friendliness.
2. Service Messaging – the format of the data used for web services. XML is the text-based protocol whose data can be easily understood by machines in a number of character sets. SOAP defines the non-application specific format of the XML to be used in the web service.
3. Service Description – Web service description language (WSDL) defines, in a standardized manner (with XML, of course) what operations are available from the service, the type of messages it will accept, and the protocol to which the consumer must bind to access the service.
4. Service Registry – Web services also provide a uniform means to dynamically discover other services that may be of interest. Universal Description, Discovery and Integration (UDDI) is a web service itself that supports a set of standards that allows a consumer to dynamically discover and locate the description for a web service.

The bottom line is that we now have a set of standards for cross-platform application communication and service discovery.

## **Moving the Quote Request Service to a Web Service**

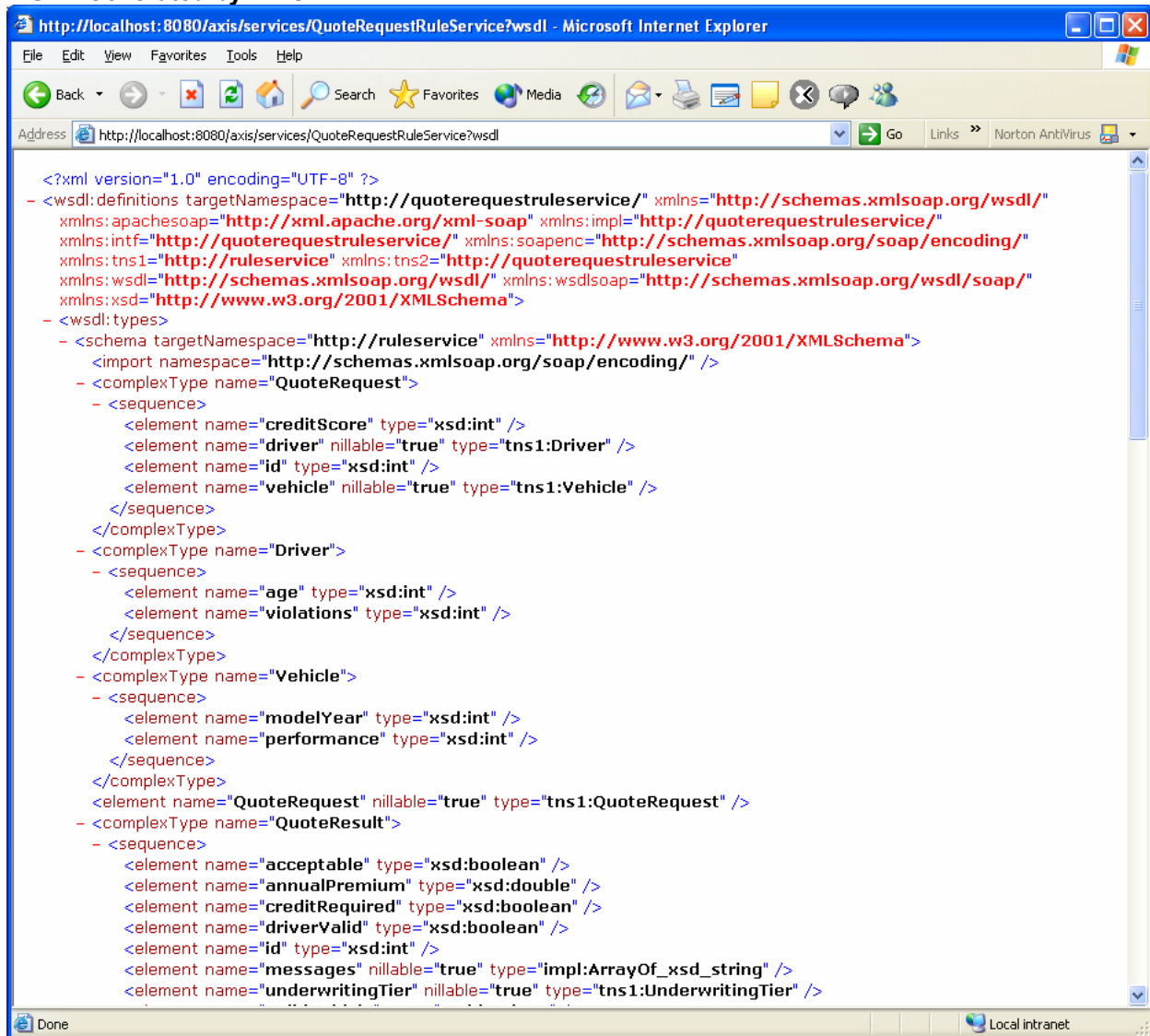
Our requirements indicated that it will be necessary to communicate with non-Java clients as well as Java clients. The company's non-Java CRM system would like to be

able to call the Quote Request service for providing rate quotations over the phone by customer service personnel. In addition, the company has relationships with a number of 3<sup>rd</sup> party comparative rating vendors that would like to communicate directly with Multi-Risk rather than relying on estimated rates developed from published rate manuals. To satisfy these requirements, Multi-Risk will publish a platform neutral, XML based, web services interface to its Quote Request service.

ILOG allows us to add a web services target to our rule service in addition to a J2EE target. Following the same steps as above, we can easily publish our service to Apache AXIS. AXIS is the Apache implementation of the SOAP 1.1 standards for web services. Apache AXIS runs on the Tomcat Web Server and Servlet engine. Other web service implementations are available including those from most major J2EE application server vendors. However, ILOG JRules ships with a distribution of the AXIS and Tomcat so we will use this for publishing of our web service. To create the files necessary for deployment to AXIS, right click on the web services target in the JRules Web Services Explorer. Again, a series of files are generated, including an ANT build script for controlling the server and deploying the web service.

After following the simple instructions for starting the AXIS server and deploying our rules service, we can browse to the WSDL (above) created by AXIS when we deployed our service at <http://localhost:8080/axis/services/QuoteRequestRuleService?wsdl>. As we study the WSDL, we can see that our classes have been transformed into XML complex types. These data structures, along with the header information, describe the contract that our web service is expecting to fulfill and that clients must adhere to. Other applications and platforms can examine this WSDL and create a client stub (i.e. proxy) for connecting to it in a platform neutral manner.

## WSDL Generated by AXIS



```
<?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://quoterequestruleservice/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachsoap="http://xml.apache.org/xml-soap" xmlns:impl="http://quoterequestruleservice/"
  xmlns:intf="http://quoterequestruleservice/" xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns1="http://ruleservice" xmlns:tns2="http://quoterequestruleservice"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <wsdl:types>
- <schema targetNamespace="http://ruleservice" xmlns="http://www.w3.org/2001/XMLSchema">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
  - <complexType name="QuoteRequest">
    - <sequence>
      <element name="creditScore" type="xsd:int" />
      <element name="driver" nillable="true" type="tns1:Driver" />
      <element name="id" type="xsd:int" />
      <element name="vehicle" nillable="true" type="tns1:Vehicle" />
    </sequence>
  </complexType>
  - <complexType name="Driver">
    - <sequence>
      <element name="age" type="xsd:int" />
      <element name="violations" type="xsd:int" />
    </sequence>
  </complexType>
  - <complexType name="Vehicle">
    - <sequence>
      <element name="modelYear" type="xsd:int" />
      <element name="performance" type="xsd:int" />
    </sequence>
  </complexType>
  <element name="QuoteRequest" nillable="true" type="tns1:QuoteRequest" />
  - <complexType name="QuoteResult">
    - <sequence>
      <element name="acceptable" type="xsd:boolean" />
      <element name="annualPremium" type="xsd:double" />
      <element name="creditRequired" type="xsd:boolean" />
      <element name="driverValid" type="xsd:boolean" />
      <element name="id" type="xsd:int" />
      <element name="messages" nillable="true" type="impl:ArrayOf_xsd_string" />
      <element name="underwritingTier" nillable="true" type="tns1:UnderwritingTier" />
    </sequence>
  </complexType>
</schema>
</wsdl:types>
</wsdl:definitions>
```

The Microsoft .NET platform provides some convenient mechanisms for generating stubs from WSDL. Using the command line wsdl compiler, we generated a “dll” that contains the interface proxies generated from the WSDL. We import those into a new Visual Studio C# Console Project wrote a simple C# program to call the web service. The program code and console output were as follows:

```
12  /// </summary>
13  [STAThread]
14  static void Main(string[] args)
15  {
16      QuoteRequestRuleServiceService service = new QuoteRequestRuleServiceService();
17      QuoteRequest request = new QuoteRequest();
18      request.creditScore = 450;
19      request.id = 100;
20      Driver driver = new Driver();
21      driver.age = 32;
22      driver.violations = 1;
23      Vehicle vehicle = new Vehicle();
24      vehicle.modelYear = 1999;
25      vehicle.performance = 0;
26      request.driver = driver;
27      request.vehicle = vehicle;
28      service.Url = "http://localhost:8080/axis/services/QuoteRequestRuleService";
29      ProcessOut processOut = service.process(request);
30      QuoteResult result = processOut.result;
31      Console.WriteLine("The id is " + result.id);
32      Console.WriteLine("The driver is OK..." + result.driverValid);
33      Console.WriteLine("The vehicle is OK..." + result.validVehicle);
34      Console.WriteLine("Credit is required..." + result.creditRequired);
35      Console.WriteLine("The risk is acceptable..." + result.acceptable);
36      if (result.acceptable)
37      {
38          Console.WriteLine("The underwriting tier is..." + result.underwritingTier.description);
39          Console.WriteLine("The annual premium is..." + result.annualPremium);
40      }
41      if (result.messages != null)
42      {
43          for (int i = 0; i < result.messages.Length; i++)
44          {
45              Console.WriteLine(result.messages[i]);
46          }
47      }
48      Console.ReadLine();
49  }
50  }
```

```
C:\Development\jproject\RuleService\dotnet\TestWebService\bin\Debug\TestWebService...
The id is 100
The driver is OK...True
The vehicle is OK...True
Credit is required...True
The risk is acceptable...True
The underwriting tier is...PREFERRED
The annual premium is...400
```

## Conclusion

We have just demonstrated how to build a simple service oriented application that handles quote requests for a fictitious insurance company. More importantly, the service was architected using an expert system rules engine to handle the complex business rules and orchestration of the business process. By externalizing the rules into a tool such as ILOG JRules, we gain tremendous flexibility for changing business policy and workflow without incurring the usual delays associated with traditional approaches. Additionally, we gain a tremendous performance boost when a large number of rules are required due to the inherent optimization performed by the RETE based rules engine.

Finally, after publishing a rule service for Java clients, we demonstrated how to expose our rule service as a platform independent web service. Multi-Risk Insurance Company can now provide a valuable service to its internal and external non-Java client applications, including the 3<sup>rd</sup> party comparative rating vendors.

## About the Author

Daniel C. Hayes is an independent IT consultant and software architect based in Valley Forge, Pennsylvania. Dan specializes in designing enterprise architectures that incorporate expert systems rules engines for complicated business processes. Dan has 15 years of experience as a business and technology leader and is an authorized ILOG JRules integration partner. He holds graduate and undergraduate degrees from Columbia Business School and Florida State University and is also completing his Master's of Software Engineering work at Pennsylvania State University – Great Valley. He can be reached by email at [dan@mydogboris.com](mailto:dan@mydogboris.com).

## References

*“Service Oriented Architecture: A Primer”*, Pallos, EAI Journal, December 2001

*“Predicts 2003: SOA Is Changing Software”*, Schulte, Gartner Research, December 9, 2002

*“The Next Big Thing”*, Keppler, JavaPro Magazine, August 2003.

*“Getting Ready for a Service-Oriented Enterprise Architecture”*, Barry, Cutter Consortium Executive Report, Vol. 5, No. 8

*“Java Web Services Architecture”*, McGovern, Tyagi, Stevens, Mathew, Morgan Kaufman Press, 2003

*“Core J2EE Patterns”*, Alur, Crupi, Malks, Prentice Hall, 2001.

SOAP 1.1 Recommendation, World Wide Web Consortium, 2001.