WebSphere. software

IBM

Patterns: Integrating WebSphere ILOG JRules with IBM Software

WebSphere ILOG JRules overview and solutions

Patterns, existing applications, BPM, and connectivity

Scenarios with JRules, CICS, WebSphere components

Andy Ritchie
Sriram Balakrishnan
Duncan Clark
Phil Coxhead
Daniel Donnelly
Kallol Ghosh
Daniel Millwood
Nicolas Peulvast
Ian Vanstone

Redbooks

ibm.com/redbooks

International Technical Support Organization

Patterns: Integrating WebSphere ILOG JRules with IBM Software

Febrary 2011

SG24-7881-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (Febrary 2011)

This edition applies IBM WebSphere ILOG JRules business rules management system (BRMS).

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication describes how the IBM WebSphere® ILOG JRules product can be used in association with other IBM middleware products to deliver better solutions.

This book can help architects position a business rule management system (BRMS) in their existing infrastructures to deliver the value propositions that the business needs.

This book can also help developers design and integrate JRules with those middleware products (focussing on WebSphere Process Server, WebSphere Message Broker and IBM CICS®) and help to illustrate common integration patterns and practices for these products.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Hursley Center.

Andy Ritchie is a Senior Software Engineer with IBM. His area of focus is BPM and Application Integration solutions, integrating with Business Rule Management Systems. He is a Development Architect who often works with IBM services and technical sales teams on solutions and methodologies. With more than 24 years in IBM, he has a broad and varied experience, from architecting IBM SOA middleware solutions in multiple industries involving SAP applications with a specific focus on BPM and SOA service governance areas. Previously, he spent 18 years in IBM Development on ISDN, X.25 communications and Speech/IVR products.

Sriram Balakrishnan is a Senior Managing Consultant with IBM Global Business Services®. He has more than 17 years of experience in architecting and implementing enterprise-wide, mission-critical systems. He holds a Master of Technology in Software Engineering from the National University of Singapore. His current areas of expertise include business process management using service-oriented architecture and J2EE custom development.

Duncan Clark is an IT Architect in the ILOG® Synergies team at IBM Hursley Park development laboratory in the UK. He currently specializes in integrations between ILOG Business Rule Management Systems and Tivoli® products, and their application in industry-focused solutions. These solutions include applying decisions to service availability and performance applications, based on Tivoli Netcool® Impact and making smarter asset life cycle decisions with Tivoli Maximo®. This work concentrates on how to realize smarter decision-making in customer solutions by applying repeatable patterns of product integration. Prior to this work, Duncan led the Governance and Policy architecture for WebSphere Service Registry and Repository.

Phil Coxhead is a Services Consultant in the IBM Software Services for WebSphere organization working as part of the Worldwide Technology Practice. Phil is a recognized specialist in the enterprise service bus (ESB) domain, having spent over 10 years working directly with customers on a wide range of business integration and ESB solutions and within IBM product development teams in the middleware space. Phil's most recent development role was as a Development Team Lead responsible for the new messaging component of WebSphere Application Server Version 6. Prior to this role, he worked in a Pre-Sales role, assisting customers using WebSphere MQ and Broker products in Business Integration proposals and scenarios. Previously, he was a Team Lead in WebSphere MQ Development where he was a member of the original group that designed and developed the early releases of WebSphere Message Broker. Phil is also a frequent speaker at IBM customer and internal conferences.

Daniel Donnelly is an Advisory Software Engineer at the IBM Hursley Park development laboratory in the UK. He is currently part of the WebSphere ILOG team, working on integrating WebSphere ILOG products with the rest of the IBM product portfolio. Prior to this role, he was a member of the CICS development team, where he spent eight years coding and testing for CICSPlex® System Manager.

Kallol Ghosh is a Senior Consultant at IBM for the BRMS product line working with the WebSphere Services team in the UK. He has more than 10 years of experience of delivering bespoke application in various business domains, including more than five years of experience in implementing smart automated decision-making services using ILOG JRules. He also has more than eight years of experience in the manufacturing industry. He has a degree in Engineering from Calcutta University in India.

Daniel Millwood is a Software Developer working within the CICS Transaction Server for z/OS® development team in England. He has worked at IBM for 15 years on messaging and transaction processing products. He holds a degree in Computer Science from Southampton University. His areas of expertise include WebSphere MQ, WebSphere Application Server, and CICS TS. In his role with CICS TS, Daniel has focused on the integration of CICS TS with other products, using technologies such as web services.

Nicolas Peulvast is a Software Engineer on the IBM ILOG Business Rule Management Systems at the Software Lab at IBM in France. He has over a decade of experience developing Java™ Enterprise applications. He worked on the Enterprise Integration team as a Developer in charge of MDB and HTDS. For the past two years, he has worked on the Rules for .Net product as a Software Architect, and participates in multiple aspects of the product, including Decision and Validation Service and Rule Execution Server for .Net product. He has a Master's degree in artificial intelligence systems.

Ian Vanstone is an Advisory Software Engineer, based at IBM Hursley, UK. Ian joined the WebSphere MQ development team in 2000, where he focused on messaging intercommunication and availability architectures, before moving to his current role as an Integration Specialist in the IBM Software Group Federated Integration Test team in 2008.

Thanks to the following people for their contributions to this project:

Chris Backhouse, ILOG Synergies and Integration Team
IBM Hursley

Chris Berg, Product Manager, WebSphere BRMS (ILOG)
IBM USA

Richard Conway
International Technical Support Organization, Poughkeepsie

Louisa Gillies, Manager, ILOG Synergies,
IBM Hursley

Luisa Lopez de Silanes Ruiz, Senior IT Specialist
IBM Hursley

Chris Rayns
International Technical Support Organization, Poughkeepsie

Stephen Smith
International Technical Support Organization, Raleigh Center

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Part 1

Introduction

Part 1 contains the following chapters:

1

What is business rules value proposition

In this chapter, we look at the value proposition that business rules offer, managing business rules, and benefits of business rule management JRules.

This chapter includes the following topics:

- Introduction to business decisions
- Managing business rules

1.1 Introduction to business decisions

We start with a background of where a business rules management system fits in the overall IT and business picture.

Automated business decisions are everywhere. Figure 1-1 shows examples of three common types of business decisions: In the upper left (compliance-related decisions), upper right (marketing or sales-related decisions), and lower right (operational decisions). Although each example seems distinct and separate from the others, each is focused on decisions that take place in business systems and that organizations are looking to automate when possible.

Business decisions are everywhere…
We need to add an eligibility check to meet the requirements of the new regulation.
Let's create a special promotion for our best customers.
Can we automate approvals for this type of order?
And changing frequently

Figure 1-1 Business decisions

Another important similarity between each decision is the variability that exists from one customer or transaction type to the next. The correct decision can vary based on geography, product, contract, or a number of other conditions.

In addition to the need to automate business decisions throughout various functions of the organization, there is unrelenting pressure to evolve, that is to change the details of those decisions over time.

These examples are a few of many. Several more specific decisions that organizations want to automate when possible are as follows:

- How often does an insurance carrier need to update underwriting rules to stay current with its competition?
- How frequently does an online retailer's upsell or cross-sell tactics change?
- Have you ever had exactly the same commission plan three years in a row?

The appropriate or correct decision today will not necessarily be the same in the future. This time is when intelligent management of business rules come into play.

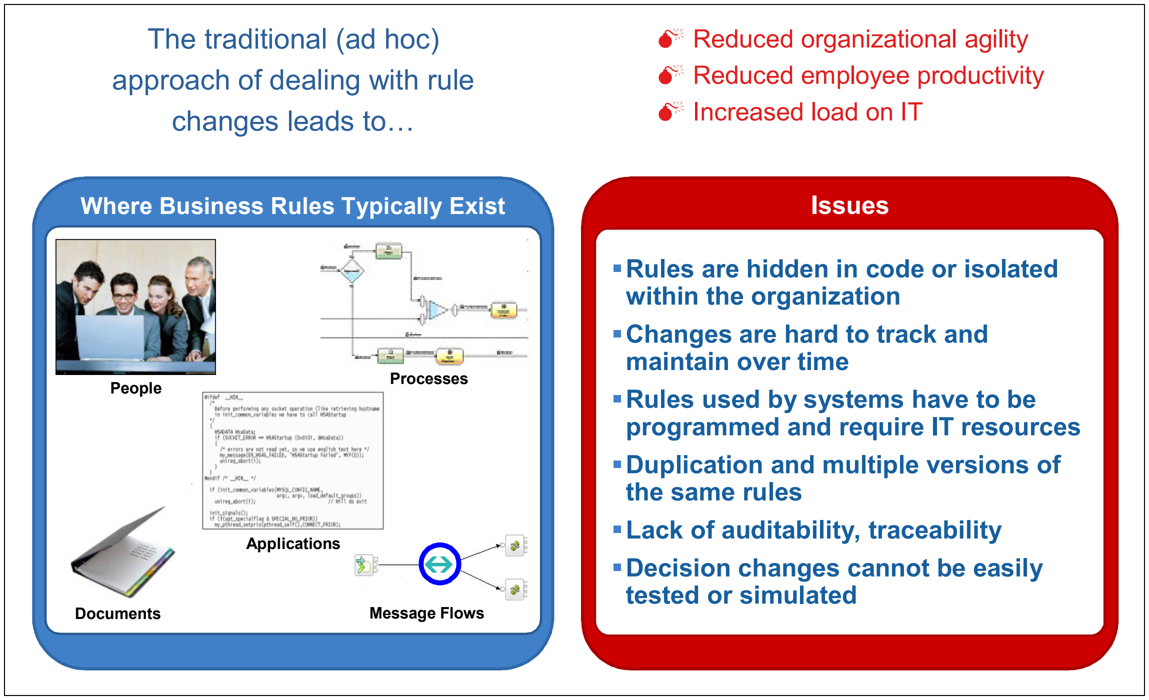
1.2 Managing business rules

Automating highly variable business decisions, and managing changes to them, are at the heart of the business rule management system value proposition.

Decision logic within systems is also referred to as *business rules*. You can think of conditional statements that are used to make decisions such as pricing for insurance or loan underwriting, eligibility approvals for social or health services, or product recommendations for online purchases. Business rules are typically found inside application code, in the form of if-then-else statements, although they may also be stored elsewhere for documentation purposes (such as in procedural manuals and other documents), or in process models. They may even exist only in the minds of subject matter experts (SMEs) who might be involved in dealing with specific operational situations.

To effectively manage change, organizations must be able to easily and dynamically adjust business rule. This ability allows systems to behave with more intelligence, precision, and consistency, and enables increased automation of activities, transactions, processes, and customizes actions based on situational context. Unfortunately, when business rules are embedded within application code, the ability to make changes to it is difficult and costly. The result is a number of issues, as shown in Figure 1-2 on page 6.

By having business rules scattered and embedded in all these various places, organizations find themselves under pressure to deal with change and the constant evolution of decisions. The eventual consequences of these pressures are summarized in the list on the right in Figure 1-2 on page 6.



The traditional (ad hoc) approach of dealing with rule changes leads to…
Reduced organizational agility
Reduced employee productivity
Increased load on IT
Where Business Rules Typically Exist
People
Processes
Applications
Documents
Message Flows
Issues
Rules are hidden in code or isolated within the organization
Changes are hard to track and maintain over time
Rules used by systems have to be programmed and require IT resources
Duplication and multiple versions of the same rules
Lack of auditability, traceability
Decision changes cannot be easily tested or simulated

Figure 1-2 Traditional approach for managing decision change

Business rule management systems (BRMS) are integrated application development and execution platforms, allowing organizations to define, deploy, monitor, and maintain the variety and complexity of decision logic that is used by operational systems. With BRMS, decision logic can be extracted and managed separately from core application code, so that it can be easily understood, maintained, and reused throughout the organization.

By externalizing rules from application code, business experts can define and manage decision logic, reducing the amount of time and effort that is required to update decision logic in production systems and increasing the organization's ability to respond to changes in the business environment.

A BRMS includes three primary components:

- A repository that allows rules to be externalized from core application code

  With the repository, decision logic can be managed as an enterprise asset. This approach can make it easier to understand and update decision logic. By consolidating decision logic from disparate applications and information silos, it can be shared and reused throughout the organization.

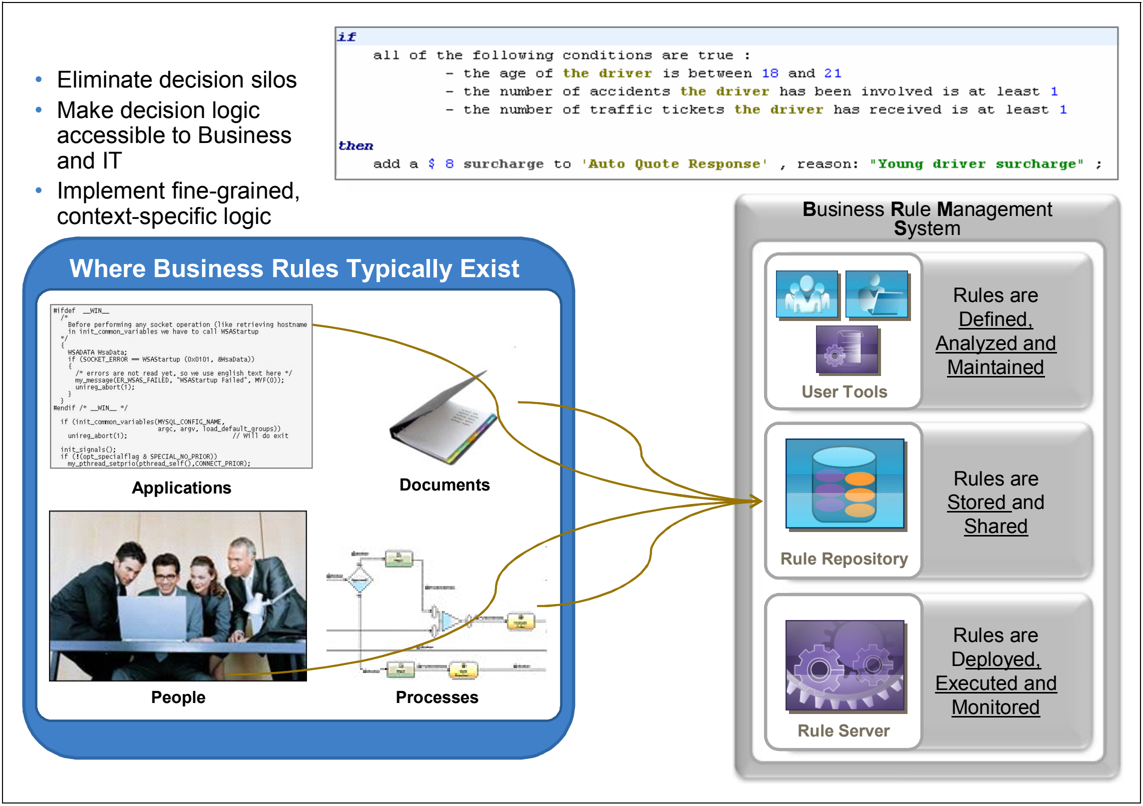
- Tools that business experts use to define and manage decision logic, which was previously buried in code

  With these tools, business functions can define application behavior, and business and IT can work collaboratively on application development and maintenance.

- A runtime engine that allows production systems to access and execute decision logic managed within the BRMS

  This “rule engine” allows complex and interrelated rules to be applied, based on specific business context using a combination of data inputs, sets of applicable rules, and execution algorithms that define how to process the data and rules so that output can be provided.

By adopting the BRMS approach, as shown in Figure 1-3, organizations are able to effectively handle problems that are associated with traditional embedded decision logic. The organization gains visibility and access to business rules, along with the ability to more easily define and automate the rules for use in the organization’s operational systems.



```
if
    all of the following conditions are true :
          - the age of the driver is between 18 and 21
          - the number of accidents the driver has been involved is at least 1
          - the number of traffic tickets the driver has received is at least 1

then
    add a $ 8 surcharge to 'Auto Quote Response' , reason: "Young driver surcharge" ;
```

```
#ifdef  __WIN__
  /*
    Before performing any socket operation (like retrieving hostname
    in init_common_variables we have to call WSAStartup
  */
  {
    WSADATA WsaData;
    if (SOCKET_ERROR == WSAStartup (0x0101, &WsaData))
    {
      /* errors are not read yet, so we use english text here */
      my_message(ER_WSAS_FAILED, "WSAStartup Failed", MYF(0));
      unireg_abort(1);
    }
  }
#endif /* __WIN__ */

  if (init_common_variables(MYSQL_CONFIG_NAME,
                            argc, argv, load_default_groups))
    unireg_abort(1);                              // Will do exit

  init_signals();
  if (!(opt_specialflag & SPECIAL_NO_PRIOR))
    my_pthread_setprio(pthread_self(),CONNECT_PRIOR);
```

• Eliminate decision silos
• Make decision logic accessible to Business and IT
• Implement fine-grained, context-specific logic
Where Business Rules Typically Exist
Applications
Documents
People
Processes
Business Rule Management System
User Tools
Rules are Defined, Analyzed and Maintained
Rule Repository
Rules are Stored and Shared
Rule Server
Rules are Deployed, Executed and Monitored

Figure 1-3 Manage and automate Decision Logic with BRMS

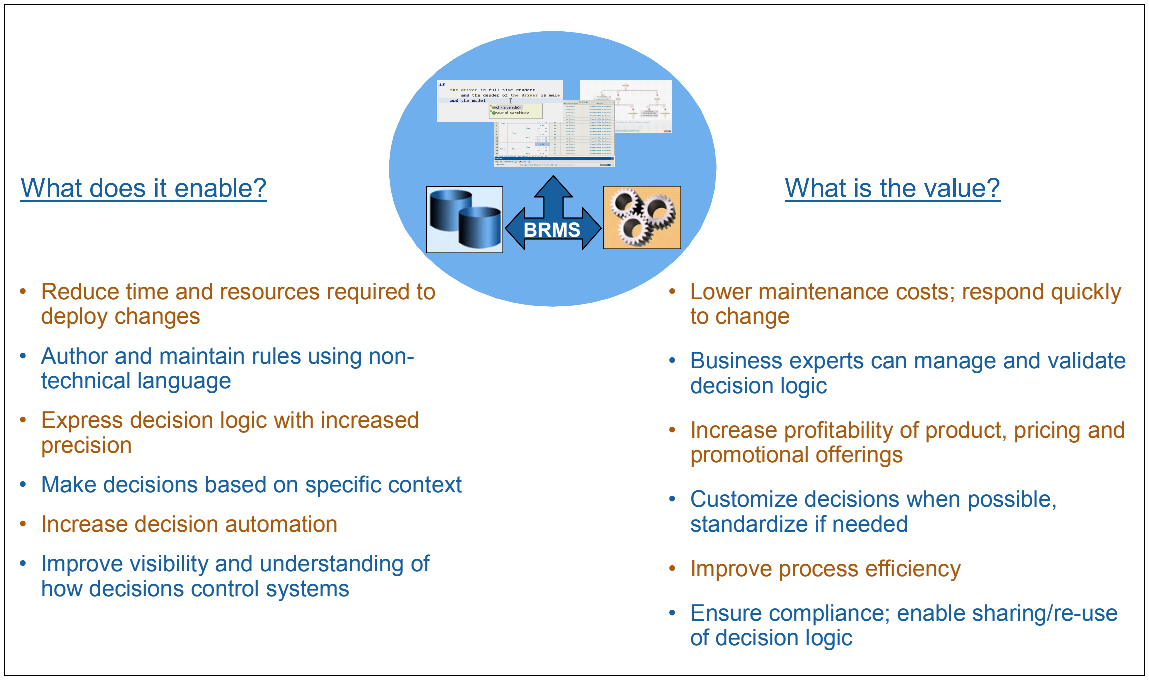
1.2.1 Benefits of business rule management

Several benefits result from the use of a BRMS:

- Reduces the time and resources that are required to change decision logic. BRMS increases the ability to respond to ongoing change, and simultaneously decreases application development and maintenance costs.
- Enables business functions to be involved in the definition and ongoing maintenance of business rules. The result is a significant amount of the workload can be shifted away from IT, so that IT can focus on the maintenance and management of the overall enterprise. This benefit creates a successful situation for the entire organization.
- Provides the ability to express decision logic with increased precision. This ability means that decision variability can be more easily handled, enabling fine-grained decision logic to be implemented. BRMS improves product and pricing decisions, customer service recommendations, and even fraud detection.
- Improves the efficiency of processes through decision automation. BRMS provides the ability to automate many decisions that have traditionally been handled manually (by people), even complex decisions based on a number of factors or conditions.
- Enables decisions to be made based on specific, situational context. With BRMS, decisions can be made based on inputs from multiple sources and use the context of each individual order, customer, transaction, or process in processing the relevant and applicable rules. This benefit in turn allows for highly customized decisions if the situation allows, and also allows enforcing standardization if required (such as meeting a regulatory mandates).
- Improves visibility and understanding of where decision logic is used and how it drives operational systems. Business rules are easily accessible, allowing them to implemented and reused with disparate systems. Organizations can ensure accuracy, consistency, and enforcement of policies.

1.2.2 The BRMS approach

Figure 1-4 shows the benefits of the BRMS approach to business rule management.



What does it enable?

- Reduce time and resources required to deploy changes
- Author and maintain rules using non-technical language
- Express decision logic with increased precision
- Make decisions based on specific context
- Increase decision automation
- Improve visibility and understanding of how decisions control systems

What is the value?

- Lower maintenance costs; respond quickly to change
- Business experts can manage and validate decision logic
- Increase profitability of product, pricing and promotional offerings
- Customize decisions when possible, standardize if needed
- Improve process efficiency
- Ensure compliance; enable sharing/re-use of decision logic

Figure 1-4 Benefits of BRMS

In terms of the usage of WebSphere ILOG BRMS offerings, our customer base spans a wide range of industries and the public sector. Figure 1-5 shows examples of applications that our customers have implemented and that use BRMS to automate and manage mission-critical decisions.

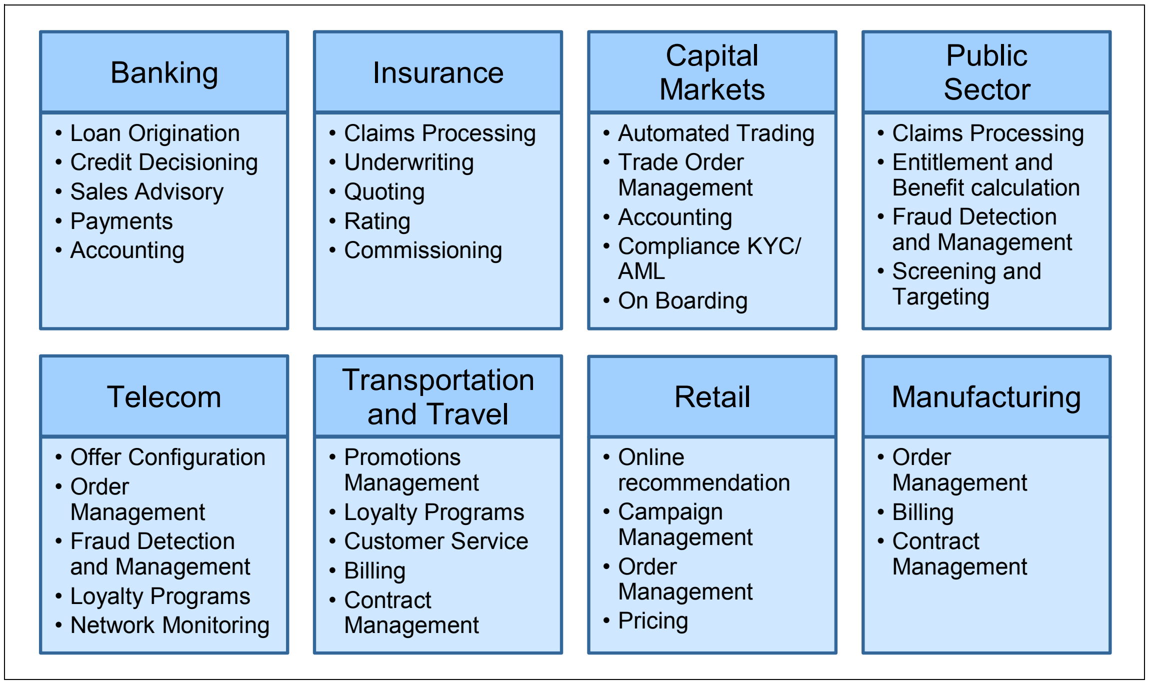


Figure 1-5 Customer BRMS examples

Additional examples of benefits are as follows (Figure 1-6):

- Reduced lead times for changes. Fast, reliable updates of customer loyalty offers, deployed directly by business users to point-of-sale (retail), equal personal care products retailer with 1600 stores across Europe.
- More personalized client interactions. Automated, interactive screening for over 40 government programs (local government) equal US local government, human services modernization program.
- Reduced new policy implementation by 50% through collaborative rule management of thousands of rules (pension administrator).
- Ability to provide ensure compliance with underwriting and account management rules based on regulations that vary by customer location and product line (insurance) = multiple insurance providers in North America, South America, Europe and Asia Pacific.

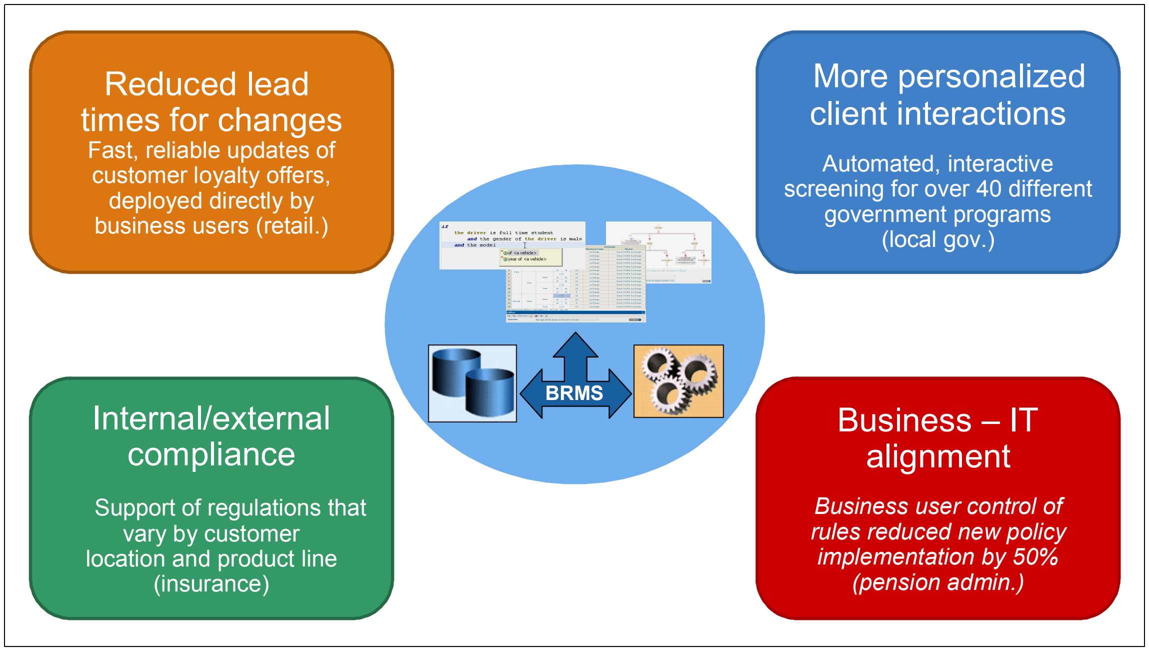

Reduced lead times for changes
Fast, reliable updates of customer loyalty offers, deployed directly by business users (retail.)
More personalized client interactions
Automated, interactive screening for over 40 different government programs (local gov.)
BRMS
Internal/external compliance
Support of regulations that vary by customer location and product line (insurance)
Business – IT alignment
Business user control of rules reduced new policy implementation by 50% (pension admin.)

Figure 1-6 Benefits of BRMS

2

Business rule management system (BRMS)

This chapter introduces the concept of business rule management system (BRMS) and the IBM BRMS offering, WebSphere ILOG JRules.

This chapter includes the following topics:

- Concepts of BRMS and ILOG JRules
- WebSphere ILOG BRMS overview

2.1 Concepts of BRMS and ILOG JRules

The concept of business rule management system (BRMS) is introduced with an overview of the IBM BRMS offering: IBM WebSphere ILOG JRules BRMS.

2.1.1 Introducing BRMS

The main purpose of using a BRMS is to respond faster to business requests for policy changes by separating the life cycle of the rule deployment from the application deployment life cycle.

As companies rely more on information technology (IT) to manage their business, IT departments must develop more complex applications and simultaneously accommodate an increasing rate of change in the applications they support.

Often, the implementation of the business policy in these applications becomes complex, voluminous, and too fast-paced for a traditional software architecture. When the maintenance of an application that uses business logic becomes challenging, a BRMS provides solutions to make this management more efficient, for both developers and business users.

With a BRMS, developers and architects can externalize the business logic from the traditional application and develop and run the business logic independently of the application. In BRMS, testing the new rules is often done by the business users themselves. This approach is in contrast to other software products, where businesses do not (or rarely) have any direct input or participation on the application testing by the business users.

A complete implementation of a BRMS can go even further and enable business users to manage business policies directly with limited dependence on the IT department. The degree of dependence can range from limited review by business users of policies that are implemented by developers, to complete control over the specification, creation, testing, and deployment of the policy by business users.

2.1.2 WebSphere ILOG JRules overview

WebSphere ILOG JRules is the IBM technology for creating, maintaining, and deploying decision services and business rules. It is also the IBM BRMS offering that enables both business users and developers to manage the rules that drive their business.

Business rules are an expression of business policy in a form that is understandable to business users and that can be applied by a rule engine. From a business perspective, a *business rule* is a precise statement that describes, constrains, or controls some aspect of a user's business. From the IT perspective, business rules are a package of executable business policy statements that can be called from an application. A business policy can be expressed as several business rules. Business rules formalize a business policy into a series of "if-then" statements.

The WebSphere ILOG JRules suite provides two BRMS product lines:

- JRules for Java
- JRules for .NET

Each provides a set of tools and environments that are specifically designed for the unique needs of the Java community and .NET community. In addition, the Rules for COBOL module extends the reach of the BRMS solution to the mainframe, System z®. It provides a way to build rule artifacts from existing COBOL structures that are defined in copybooks. The rules are generated as executable COBOL code that can easily be integrated with existing applications, based on System z, running in CICS or IMS™.

WebSphere ILOG JRules consists of a set of modules that operate in separate environments, but also work together to provide a comprehensive BRMS. A full WebSphere ILOG JRules BRMS product family consists of the following items:

- IBM WebSphere ILOG JRules:
  - Rule Studio: Eclipse-based development environment
  - Rule Execution Server: Managed execution environment
- IBM WebSphere ILOG Rule Team Server: Business user rule management environment
- IBM WebSphere ILOG Decision Validation Services: Testing, simulation and audit functions that are integrated with Rule Studio, Rule Execution Server, and Rule Team Server
- IBM WebSphere ILOG Rule Solutions for Office: Guided authoring and editing of rules through Microsoft® Office Word and Excel

Figure 2-1 shows the various modules in the environment in which they are used, and how they work together through synchronization and deployment.

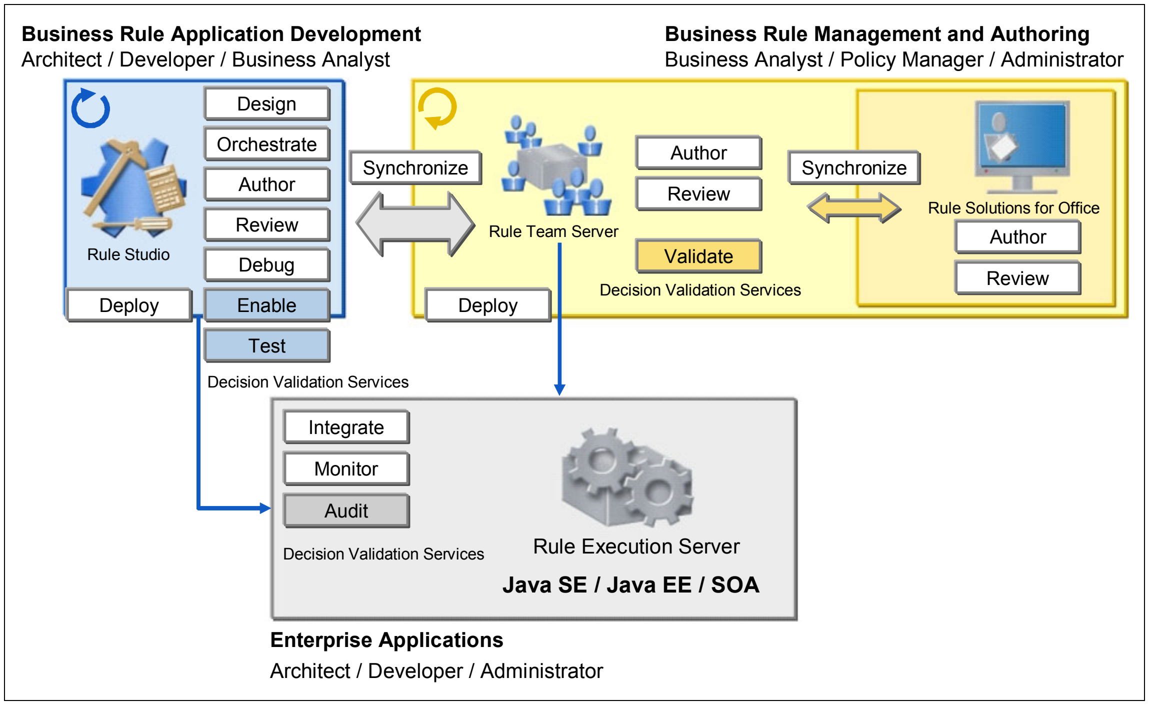

Business Rule Application Development
Architect / Developer / Business Analyst
Business Rule Management and Authoring
Business Analyst / Policy Manager / Administrator
Design
Orchestrate
Author
Review
Debug
Rule Studio
Deploy
Enable
Test
Decision Validation Services
Synchronize
Rule Team Server
Author
Review
Validate
Decision Validation Services
Deploy
Synchronize
Rule Solutions for Office
Author
Review
Integrate
Monitor
Audit
Decision Validation Services
Rule Execution Server
Java SE / Java EE / SOA
Enterprise Applications
Architect / Developer / Administrator

Figure 2-1 Usage of WebSphere ILOG JRules modules

WebSphere ILOG JRules includes three primary components (Figure 2-2 on page 17):

- A repository allowing rules to be externalized from core application code

  With the repository, decision logic can be managed as an enterprise asset, making it easier to understand and update decision logic. The repository uses consolidated decision logic from disparate applications and information silos so that it can be shared and re-used across the organization.

- Tools (Eclipsed-based and web-based) allowing business experts to define and manage decision logic that was previously buried in code

  With these tools, business functions can define application behavior, and business and IT can work collaboratively on application development and maintenance.

- A runtime engine allowing production systems to access and execute decision logic managed within the BRMS

  The “rule engine” allows complex and inter-related rules to be executed based on specific business context, using a combination of data inputs, sets of applicable rules, and execution algorithms that define how to process the data and rules to provide an output.

A WebSphere ILOG JRules life cycle is divided into two main periods:

- Build-time, when the application is constructed
- Change-time, instead of run time, when the application is deployed, available to its users, and continuously adapted to new and evolving business policies

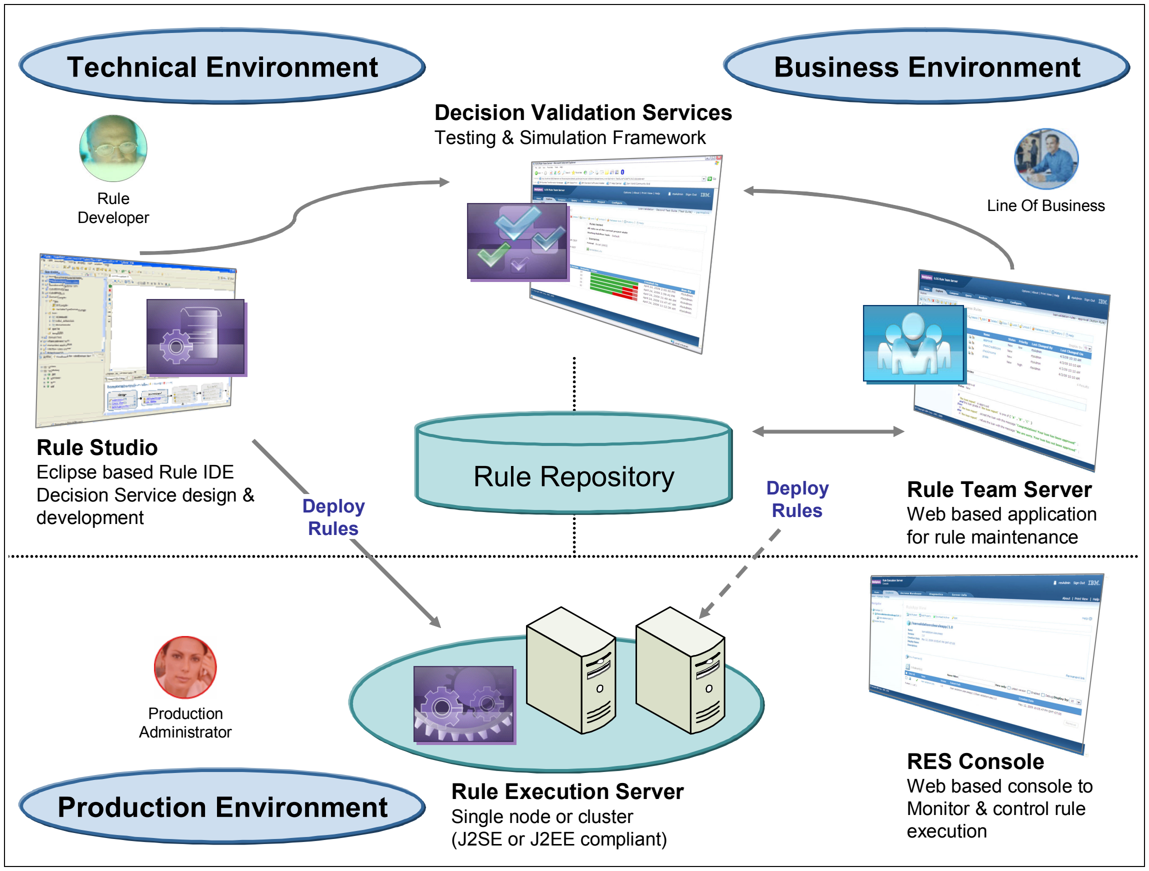

Technical Environment
Business Environment
Decision Validation Services
Testing & Simulation Framework
Rule Developer
Line Of Business
Rule Studio
Eclipse based Rule IDE
Decision Service design & development
Deploy Rules
Rule Repository
Deploy Rules
Rule Team Server
Web based application for rule maintenance
Production Administrator
Rule Execution Server
Single node or cluster (J2SE or J2EE compliant)
RES Console
Web based console to Monitor & control rule execution
Production Environment

Figure 2-2 WebSphere ILOG JRules architecture components

Benefits of WebSphere ILOG JRules BRMS are as follows:

- Easy implementation and reuse of business rules
- A convenient communication channel between IT and business teams
- Improved regulatory compliance
- Consistency in applying business decisions to multiple applications

2.2 WebSphere ILOG BRMS overview

Separating the business rule life cycle from the mainline message flow development life cycle enables message-flow-based solutions to gain the same benefits as other rule-based applications, as shown in Figure 2-3 on page 19.

The initial release of the solution includes both message flow and business rule components. When the message-flow-based solution is in production, business policy changes may require changes to the business rules that are currently executing in the solution. When these policy changes are implemented in changed or new rules, they can be redeployed into production without the need to redeploy the main message flow.

Multiple changes to business policies and their rule implementation may occur between major deployments of the parent message flow. When a change is made to the main message flow that requires a redeployment, the changes to the business rules can be synchronised with the base message flow.

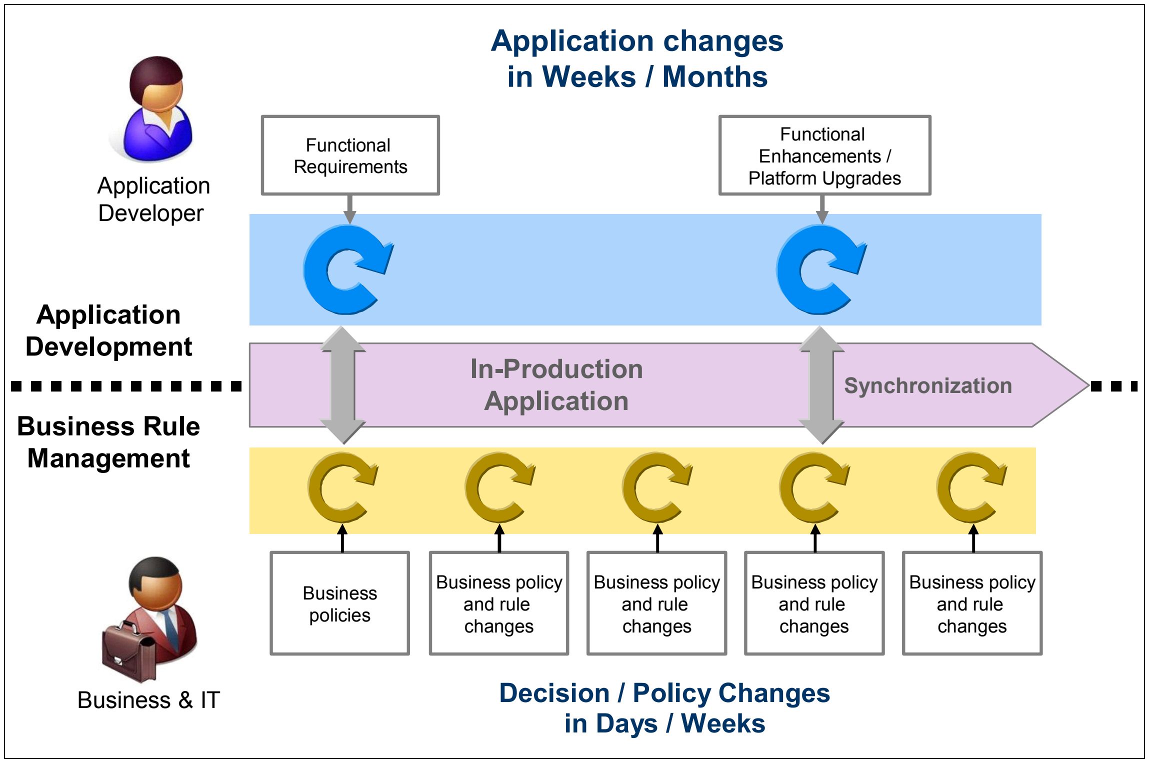

Application changes in Weeks / Months
Application Developer
Functional Requirements
Functional Enhancements / Platform Upgrades
Application Development
In-Production Application
Synchronization
Business Rule Management
Business policies
Business policy and rule changes
Business policy and rule changes
Business policy and rule changes
Business policy and rule changes
Business & IT
Decision / Policy Changes in Days / Weeks

Figure 2-3 Separate application development and rule life cycles

2.2.1 JRules BRMS and Rules for .NET BRMS

The WebSphere ILOG family of BRMS products includes two primary product lines:

- JRules BRMS
- Rules for .NET BRMS

Each provides a set of tools and environments specifically designed for the unique needs of their application operating environments (Java and .NET). They also provide common components and capabilities that can be used with both product lines.

With JRules BRMS, developers create and maintain rule applications in an Eclipse-based IDE; business users manage rules through the web-based, collaborative WebSphere ILOG Rule Team Server environment. In addition, business users can export rules out of the rule repository into a file-based format, allowing rules to be maintained in Microsoft Word and Excel by using the

WebSphere ILOG Rule Solutions for Office product. In addition, the JRules BRMS offers a set of testing, simulation, and execution audit capabilities, called WebSphere ILOG Decision Validation Services, which are integrated directly within the various user environments. Rules or rule sets can be deployed into production in several ways:

- Directly to the J2EE-based Rule Execution Server
- As web services (Transparent Decision Services)
- Generated as COBOL copybook code by using the WebSphere ILOG Rules for COBOL product

With Rules for .NET BRMS, developers create and maintain rule applications in a Visual Studio-based IDE; business users manage rules by using either Rule Team Server or Rule Solutions for Office. Rules or rule sets can be deployed into production either directly to the .NET-based Rule Execution Server or as Transparent Decision Services.

Although each product line has its own specific tools and environments, there are points of commonalities. Figure 2-4 on page 21 illustrates the interactions of the product lines.

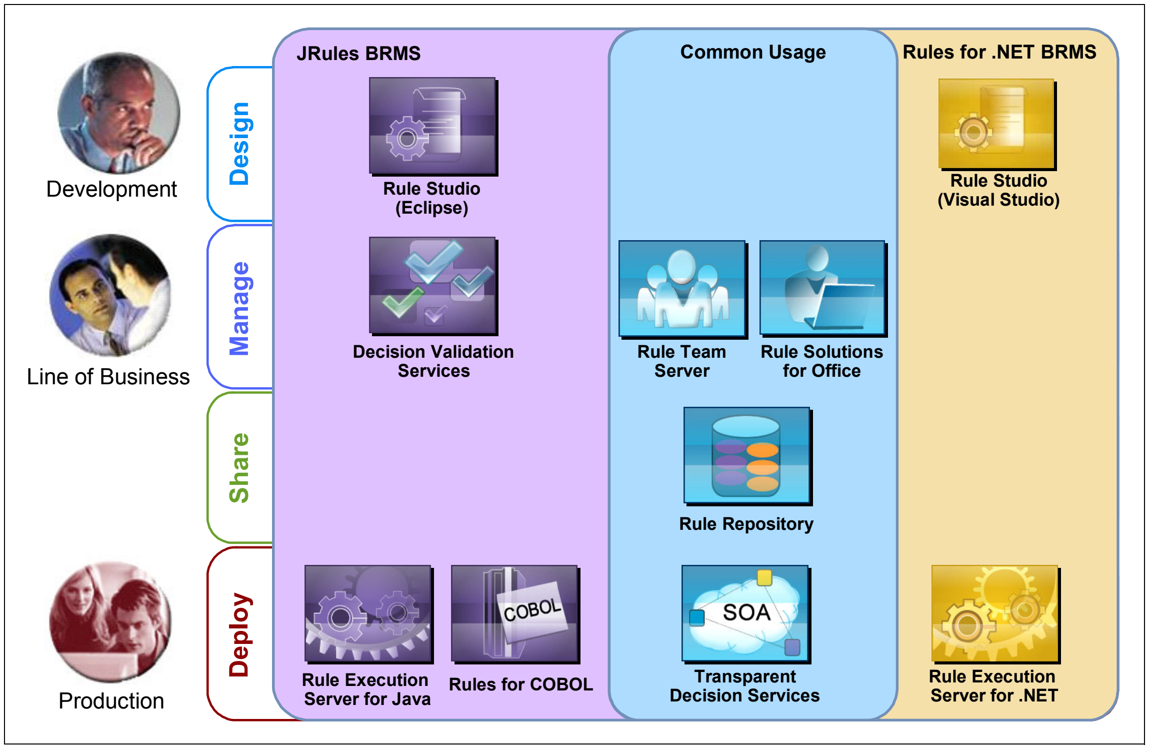


Figure 2-4 ILOG Business Rules product family

2.2.2 JRules BRMS for System z

JRules BRMS provides two options for System z:

- Option 1: JRules plus Rules for COBOL

  This option is for Customers who want to manage their business decisions, and continue to execute decision logic directly within COBOL applications.

- Option 2: JRules running Rule Execution Server on zOS

  This option is for Customers who are moving to a Java SOA Mainframe Architecture.

WebSphere ILOG Rules for COBOL can help with the following tasks:

- Improve the manageability of your COBOL applications by externalizing decision logic and managing it through the BRMS.
- Execute rules natively within your existing application flows through Rules for COBOL copybook code generation, which then can be placed directly with the COBOL application.

- Ensure consistency of business decisions across applications.
- Create rules for your COBOL applications now and use them in Java and SOA later.

With the latest release of Rules for COBOL, transforming rules into code for use in COBOL applications can be easier and more powerful than ever. See Figure 2-5.

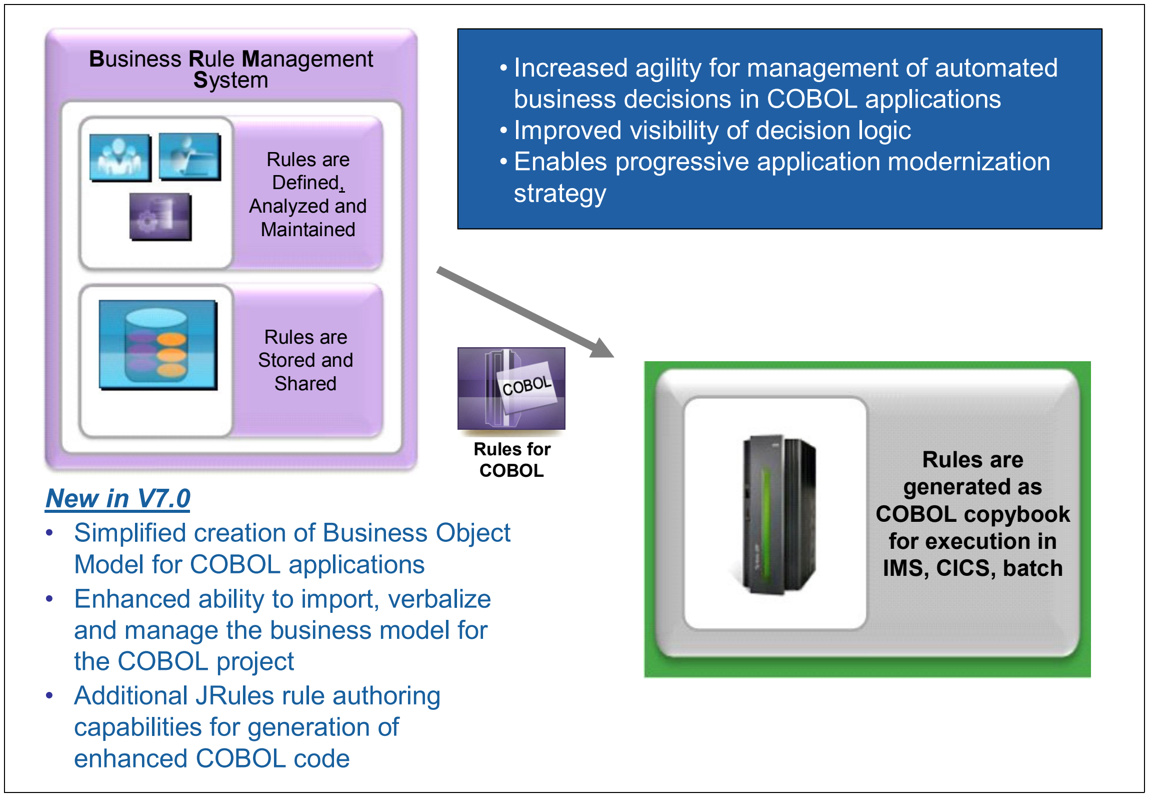

Business Rule Management System
Rules are Defined, Analyzed and Maintained
Rules are Stored and Shared
• Increased agility for management of automated business decisions in COBOL applications
• Improved visibility of decision logic
• Enables progressive application modernization strategy
COBOL
Rules for COBOL
Rules are generated as COBOL copybook for execution in IMS, CICS, batch
New in V7.0
• Simplified creation of Business Object Model for COBOL applications
• Enhanced ability to import, verbalize and manage the business model for the COBOL project
• Additional JRules rule authoring capabilities for generation of enhanced COBOL code

Figure 2-5 BRMS for Legacy applications

To summarize, we review why BRMS can benefit organizations. To effectively manage change, organizations need to be able to easily and dynamically adjust decision logic within their business systems. Doing so enables systems to behave with more intelligence, precision and consistency, enables increased automation of activities, transactions, processes, and customizes actions based on situational context.

Many organizations are looking for ways to improve their management of business systems decision logic to increase operational efficiency by the following tasks:

- Implementing change in a way that is easy, safe, and predictable
- Reducing the time and cost required to develop and maintain operational systems that are affected by change
- Providing a way for IT and business functions to work collaboratively to define and update the decision logic that drives operational systems.
- Increasing the visibility of how systems use and are affected by decision logic

IBM can offer all these benefits to customers through the most comprehensive, industry-leading business rule management offering.

3

Typical BRMS solutions

A common theme in industry is that everything is about pricing. Success against the competition depends on the ability to be innovative about introducing new products and creating new ways to offer pricing and promotions.

For years, industries maintained their prices in a database, which caused pricing changes to be a database problem. This problem meant that a database programmer was needed to make any change. Today’s economic climate and market conditions is forcing industries to change the way they perform decision-making to gain market shares. Businesses must do more with less, and therefore must automate more processes and engage fewer manual steps.

Additionally, new regulations that must be integrated into the decision process are making manual, hard-coded business decisions obsolete. Tools that provide flexibility and adaptability are needed to keep pace with these changes. WebSphere ILOG JRules BRMS provides these requirements.

In this chapter, we examine how WebSphere ILOG BRMS can be used in solutions for the following industries:

- Solutions for insurance
- Solutions for manufacturing operations
- Solutions for promotions and loyalty programs
- Solutions for consumer lending and credit
- Solutions for e-government
- Solutions for service fulfillment

3.1 Solutions for insurance

In this section, we introduce a solution for the insurance industry.

3.1.1 Overview

Policies, procedures, and regulations dominate the insurance industry, but manual and paper-intensive processes, and hard-coded applications are driving up costs. Quotes and settlements can take weeks; business logic or rate changes can take months; and introducing new products and services can seem to take forever. Delivering a consistent level of service and establishing best practices has become a daunting task.

With a changing market, competitive and regulatory conditions, the insurer's need for flexibility, transparency, and speed-to-market is greater than ever. Forward-thinking insurance companies know that to improve their financial outcome and competitive edge they must achieve new levels of responsiveness or they must risk being left behind.

3.1.2 Use case

WebSphere ILOG BRMS can provide a cost-effective way for insurance carriers to enhance their system and operational agility. Unlike ordinary "one-size-fits-all" solutions, WebSphere ILOG BRMS can help to deliver unparalleled speed, flexibility, and functionality. Companies can maximize their existing platforms with open technologies and industry data standards, such as ACORD XML, to enable a wide range of strategic applications throughout multiple channels and systems.

Leading property and casualty, health, and life insurance companies worldwide rely on WebSphere ILOG BRMS to reduce their underwriting and claims leakage costs, enhance operational efficiency by optimizing processes, and improve customer retention and acquisition. Using WebSphere ILOG BRMS, insurance carriers go beyond basic automation to enable intelligent decisions throughout their core processes and throughout lines of business, whether for underwriting, policy administration, claims processing, or billing. In addition, WebSphere ILOG BRMS provides a large panel of technology for building cutting-edge applications for key insurance operations such as configuring and recommending customized health plan packages and life products, fraud detection, e-claims and online delivery of insurance quotes. WebSphere ILOG BRMS has a track record of reducing development time and risks while delivering high return on investment (ROI).

Leading property and casualty, health, and life insurance companies rely on WebSphere ILOG BRMS to optimize their core processes and enable cutting-edge applications that improve competitiveness and profitability. From web-based solutions and call centers to front- and back-office systems, WebSphere ILOG BRMS is used throughout the enterprise for the following functions:

- Underwriting

  WebSphere ILOG BRMS is ideal for adding and automating intelligent decisioning capabilities in underwriting systems. Carriers can implement exception-based processing for both new policies and renewals, enabling underwriters to focus on more complex cases. Decisioning software is used throughout the underwriting process and throughout lines of business from determining eligibility and risk assessment to scoring, rating, and referrals. Insurance carriers can deliver quotes in just minutes, accurately correlate risk to price on a consistent basis, and proactively segment unacceptable business and reduce their loss ratios.

- Claims processing

  Customer loyalty and retention is often affected by an insurance company's ability to process and settle claims in a timely manner. Insurance carriers can improve the effectiveness of their claims processing and deliver a consistent level of service throughout customer touch points, realizing significant gains. WebSphere ILOG BRMS is widely used throughout claims systems and lines of business for such functions as case assignment, task management, and adjudication. In addition, carriers can identify exceptions earlier in the claims life cycle, enabling rapid detection and referral of potential fraud. They can automate workflow-related tasks, trigger notifications, and implement best practices for claims handling, which can reduce training, and claims leakage costs while improving customer retention.

- Billing

  Process delays and high call-center volume pertaining to billing inquiries can quickly increase costs for insurers. Leading insurance carriers are using WebSphere ILOG BRMS as a core solution in their enterprise billing systems to improve performance and service levels. Created to handle large volumes of transactions, WebSphere ILOG BRMS deliver the utmost flexibility and scalability, enabling insurance carriers to easily meet customer needs with a variety of promotions, discounts, and payment plans.

3.2 Solutions for manufacturing operations

In this section, we introduce a solution for the manufacturing industry.

3.2.1 Overview

Demand is rising fast throughout the manufacturing industry for greater control and faster execution of business processes. A new wave of improvements through business process management (BPM) and service-oriented architecture (SOA) solutions is rolling through the industry. Companies are especially keen to adopt SOA, which links resources on demand for greater flexibility and efficiency. Many companies see the implementation of an SOA strategy as their primary IT priority.

3.2.2 Use case

WebSphere ILOG BRMS can capture the logic behind processes in accurate and readily understood business rules that are used to automate decisions. These rules are stored externally from applications, making them easier to maintain and share throughout enterprise systems. This approach helps policies to be managed through WebSphere ILOG BRMS based decision services as part of an SOA.

WebSphere ILOG BRMS allows global companies to generate reports and maintain a comprehensive audit trail of historical, active, and shared rules to improve internal control measures and overall process transparency. WebSphere ILOG BRMS provides effective SOA by the following benefits:

- Enhancing process flexibility and agility.
- Increasing efficiency and scalability.
- Improving cross-channel interactions and impact analysis.
- Capitalizing on knowledge and automating best practices.

WebSphere ILOG BRMS helps manufacturers get the most from their applications and increase their operational efficiency throughout the enterprise:

- Sales and distribution
- Product and service configuration
- Order management and billing (cross-channel pricing and billing)
- Promotions and loyalty programs
- Commissioning management (resellers, dealers, and cross-brands)
- Warranty and claims management
- Production
- Global available to promise, order sourcing, routing and fulfillment

- Technical configuration (process and equipment configuration)
- Quality and process monitoring and control
- Automated quotation, cost calculation and margin analysis
- Design validation (automation of best practices)

WebSphere ILOG BRMS supplies the tools needed to anticipate and react:

- A fully integrated IDE for modeling, maintaining, debugging, deploying and managing business rule applications.
- A central repository that provides highly customizable rule management features and supports collaboration between IT and business users.
- Intuitive, business user-friendly rule editors with point-and-click interaction.
- Automatic consistency checking that ensures the integrity of rules for greater performance and accuracy.
- Web-based rule management for remote access and management.
- Performance through the fastest business rule engines on the market.
- Ability to make decisions based on data in multiple systems.

3.3 Solutions for promotions and loyalty programs

In this section, we introduce a solution for promotions and loyalty programs.

3.3.1 Overview

After urging customers to earn points through a loyalty program, how do you get them to spend the points? How do customers react when they realize they did not use the right coupon to get the best price in a transaction? Are they satisfied when they cannot take advantage of a special promotion because their loyalty accounts are not up-to-date?

A simple software engine that is based on hard-coded rules for managing points or promotions cannot handle these challenges. What is necessary is a solution that can clearly identify and independently manage the rules, and a solution that can perform that following tasks:

- Create an offer (rebates, product selection, sales conditions)
- Check client eligibility (customer profile, history of shopping activities)
- Execute a variety of offers (exclusive, seasonal)

In short, today’s promotional programs require an entirely new form of management.

3.3.2 Use case

To make offers more attractive, marketing departments have to implement highly granular rules that can result in complex statements. With WebSphere ILOG BRMS, marketing personnel define simple to complex offers in their preferred natural language (English, for example) with guidance from rule wizards, and run simulations to test and validate offers. Commercial partners can also be given limited access to certain rules.

Another way to make offers more attractive is to determine customer eligibility in real time at the point-of-sale, whether it is a cash register or a web browser. With WebSphere ILOG BRMS working through the ESB of the enterprise, several hundred rules can be executed just before the sales ticket is rendered. The customer can receive personalized messages, for example, the number of accumulated points or announcements for new promotions on the receipt or another medium.

To further ensure the effectiveness of special offers, a retailer must be able to limit certain offers to specific groups of stores, test the offers at certain stores, or limit them to particular sales channels. With WebSphere ILOG BRMS, the IT department can control the automated deployment of rules in the BPM of the organization. They can be easily limited to specific outlets or made widely accessible to applications throughout the enterprise, including those based on the Internet.

WebSphere ILOG BRMS generates reports that can be used to evaluate the effectiveness of campaigns. Marketing personnel can track applied offers, never-used offers, and eligible but never-used offers. They can gauge the impact of a campaign and gain insights into customers’ preferences and shopping habits.

3.4 Solutions for consumer lending and credit

In this section, we introduce a solution for consumer lending and credit.

3.4.1 Overview

Lenders must be efficient. Manual processes, hard-coded applications and rigid processes increase costs and lengthen the lending cycle. Pressure to reduce costs and credit risks, while increasing the customer acquisition rate, make streamlining processes critical.

Lending is a highly rule-intensive process. It involves a large number of complex, ever-changing policies and regulations that govern everything from qualifying applicants and assessing credit risk to pricing loan products and ensuring compliance. Not surprising then is that most lenders fall short in their ability to respond quickly to change.

3.4.2 Use case

With WebSphere ILOG BRMS solutions, banks and other financial organizations can implement changes to their lending systems in days (or even real time) instead of months. Advanced functionality such as decision tables, central rule management, and modeling and loan configuration capabilities enable them to engage in effective risk-based pricing, and also create customized rules around product, rating, scoring, and eligibility determination to best support their business strategies. Lenders can deliver instant, accurate credit decisions and make alternative product recommendations that meet their customers' needs, while at the same time adhering to underwriting guidelines. Benefits of a WebSphere ILOG BRMS solution are as follows:

- Shorter loan-processing cycle
- Reduced cost per loan
- Improved credit decisioning
- Lower credit and operational risks
- Better up-sell and cross-sell opportunities

Using WebSphere ILOG BRMS, financial institutions go beyond basic automation by adding intelligent decisioning capabilities throughout the lending process and throughout loan products, from mortgage and home equity to auto loans and credit cards.

Financial institutions that are ranked among the Global 500 use WebSphere ILOG BRMS in the following ways:

- Validation
- Scorecards
- Fraud detection and reporting
- Eligibility determination
- Rating
- Loan recovery and collection
- Pricing
- Risk assessment

- Customer segmentation
- Best fit
- Deal repair (counteroffers)
- Campaign management
- Document selection
- Fee calculation
- Loan pooling
- Credit scoring
- Compliance
- Accounting

3.5 Solutions for e-government

In this section, we introduce a solution for government.

3.5.1 Overview

Government agencies work hard to implement complex, ever-changing laws, regulations, and internal policies. Even so, the time between adoption and implementation can be considerable, and delays and errors can occur as agency personnel apply new regulations.

3.5.2 Use case

With WebSphere ILOG BRMS, governments automate critical operations such as eligibility determination, benefit calculation, and claims processing. It delivers agile, customer-centric solutions for both multichannel and web-based applications. All types of government services, such as government to consumer (G2C), government to business (G2B), and government to government (G2G), can benefit. Agency personnel can achieve tremendous levels of productivity.

WebSphere ILOG BRMS gives government agencies greater control over policy implementation. WebSphere ILOG BRMS stores policies as logic statements called *rules*, and changes to these rules can be immediately reflected in all the applications that use them. This approach ensures consistent implementation of policies throughout an agency, fewer errors, and reduced costs. It also provides a clear audit trail for changes and updates.

With WebSphere ILOG BRMS, agencies can achieve the following tasks:

- Accelerate policy implementation
- Automate key processes
- Focus on constituents
- Extend legacy systems
- Reduce development time and risk

Public administrators can increase the quality of their work and do far more work, including:

- Interactive services
- Data compliance validation and interoperability
- More user-centered e-government through shorter delivery cycles, higher service quality, and greater reliability
- Greater internal operational efficiency through flexible automation that reduces personnel costs and the number of human errors
- Immediate implementation of policy and regulatory changes by nontechnical personnel through an auditable, user-friendly development environment
- Success with business process reengineering initiatives that provide cost savings and operational agility for future operations, while using legacy systems

3.6 Solutions for service fulfillment

In this section, we introduce a solution for service fulfillment.

3.6.1 Overview

Reducing manual processing losses and service fulfillment costs is necessary.

Manual processing losses reduction

Communications service providers (CSPs) lose hundreds of millions of dollars annually to bottlenecks in their order provisioning. Partially integrated business and operations support system (BSS/OSS) applications force service fulfillment operations to rely too heavily on manual processing for tedious routine tasks such as order validation and data consistency. The result is slower service fulfillment that is more costly and more prone to human error.

Service fulfillment costs reduction

From service configuration to activation, flow-through provisioning dramatically reduces fulfillment costs, delays, and delivery times. By automating data conversion and routine complex tasks such as compliance checking, and also better integrating fulfillment systems with one another, a flow-through provisioning system based on BRMS solution, combined with several BPM and ESB products can greatly improve service fulfillment.

3.6.2 Use case

WebSphere ILOG BRMS technology automates critical processes, especially repetitive routine tasks that involve a large number of complex constraints, such as business or wholesale order validation and consistency checking. Solutions that are built with WebSphere ILOG BRMS can process thousands of rules with exceptional speed and flexibility, and enable quick and easy maintenance with no coding.

A distributed architecture ensures maximum flexibility and scalability, and successful process automation, resulting in the following improvements:

- Faster, more accurate provisioning of orders.
- Considerable cost reduction from focusing human resources on exception handling rather than routine tasks.
- Significant reduction in human errors and financial penalties.
- Dramatic reduction in implementation time for strategic changes determined by the company.
- No-coding maintenance saves additional costs and builds agility.
- Empowered business users with specific access for maintaining business logic in a secure mode.

From service configuration to activation, flow-through provisioning dramatically reduces fulfillment costs, delays, and delivery times. By automating data conversion and routine complex tasks such as compliance checking, and also better integrating fulfillment systems with one another, a flow-through provisioning system that is based on WebSphere ILOG BRMS can greatly improve service fulfillment.

Applications that use WebSphere ILOG BRMS technology can help CSPs build a decisive competitive advantage by enabling faster changes to service offerings, bringing fulfillment costs down and allowing CSPs to perform the following tasks:

- Ease integration of established systems.
- Preprocess incoming orders.
- Automate compliance checking.
- Create an agile service fulfillment chain.
- Centralize business knowledge.

Years of rapid growth, deregulation, and industry consolidation have created a highly complex business environment for CSPs. Customers want innovative services, lower rates, and faster delivery, while business and regulatory limitations make staying competitive increasingly challenging. BSS/OSS applications must deliver fast, flexible solutions for order management, service provisioning and activation, automated flow through, back-end fulfillment, service modeling and configuration, and service resource management. CSPs have to expand customer service while improving operational efficiency and finding new ways to cut costs. Flow-through provisioning has emerged as the best solution.

WebSphere ILOG BRMS solutions empower business users with a natural-language rule interface that helps those users implement changes and perform maintenance and updates without coding. This approach produces a tight, yet flexible, integration of BSS/OSS applications, and cuts the risk and cost of human error from these repetitive tasks. Using WebSphere ILOG BRMS solutions, users can achieve the following results:

- Cost reductions through automation
- Fewer human errors
- Faster, more accurate processing
- Easy maintenance and updating, with no coding required, which saves additional costs and builds business agility

Solutions that are based on WebSphere ILOG BRMS infuse flexibility and agility deep into the fulfillment chain by centralizing all relevant information while allowing for rapid updating and maintenance (without coding) while the system is running. As a result, users are able to see the following benefits:

- Achieve competitive advantage by accelerating the pace of change and the reactivity of the fulfillment chain.
- Maintain centralized business knowledge.
- Realize cost reductions through updated business- and resource-related information.
- Provide ability to provide more differentiated and customized offers to clients.

Part 2

Patterns and scenarios

Part 2 contains the following chapters:

4

Scenario and the solution architecture

Business rule management systems provide a variety of value propositions and can be used in customer solutions in a variety of ways.

This chapter provides a product-objective view of those value propositions and the integration patterns that are used to realize the solutions.

In this chapter, we provide a fictitious context for the integration of WebSphere ILOG BRMS into a solution that requires integration with various middleware capabilities. Although a specific scenario is used to clearly illustrate the value propositions and integration patterns that are encountered in an insurance solution, they are applicable to any industry or solution.

The scenario describes a fictitious auto insurance organization (named Fictional IBM Redbooks Company Insurance) that has a solution in place for generating insurance quotes for its customers. However, this solution does not meet the company's needs and there are several issues (pain points) that the company must address. This chapter describes the current solution architecture, highlighting the goals of IBM Redbooks Company Insurance.

4.1 IBM Redbooks Company Insurance background

IBM Redbooks Company Insurance has grown by acquisition and is now trying to consolidate its IT systems to deliver more profitable insurance services. This diversity is reflected from various perspectives:

- Products

  The companies that are acquired focus on separate but often overlapping risks including automobile, motorbike, medical, and home.

- Providers

  The companies offer well known insurance brands that must be retained and are usually localized to a particular country with its specialized legal, risk, and language requirements.

- IT infrastructure

  Each of the acquired companies has developed its own infrastructure, which, although now partially integrated throughout IBM Redbooks Company Insurance, has resulted in a wide variety of technologies and applications fulfilling similar roles but in different ways.

4.2 IBM Redbooks Company Insurance objectives and pain points

IBM Redbooks Company Insurance has identified inconsistent and fragmented insurance policy quotes (and thus the resulting policy and claim risk). The company wants to establish a rule-driven infrastructure that allows the company's business to consistently perform in the following ways:

- Concentrate on risks and quotes that are appropriate to the business, off loading those that are not appropriate to business partners and directing requests to the most appropriate product or provider.
- Provide a consistent base pricing and underwriting strategy across the business.
- Handle routine quotes automatically and quickly to improve customer satisfaction, reduce costs, and increase market share.

- Provide a human channel for processing higher risk, high value quotes and policies
- Apply dynamic marketing promotions to balance the product portfolio and increase the number of profitable deals.
- Reject policy applications that are likely to lead to fraudulent or high risk claims.

4.3 IBM Redbooks Company Insurance solution architecture

The infrastructure that is used by IBM Redbooks Company Insurance is varied because of the acquisitions and mergers that have occurred. Certain integration has been undertaken so each of the independent lines of business have a solution architecture, as shown in Figure 4-1.

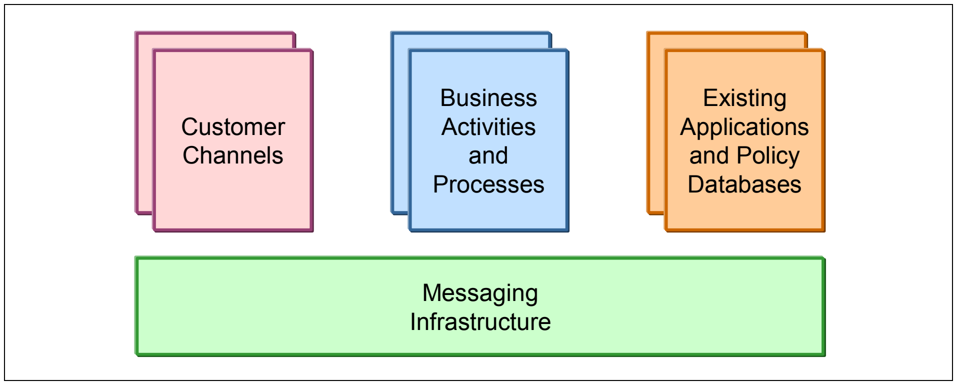

Customer Channels
Business Activities and Processes
Existing Applications and Policy Databases
Messaging Infrastructure

Figure 4-1 IBM Redbooks Company Insurance line-of-business solution architecture

The process of the infrastructure is similar to the following process:

1. Customers request insurance quotes through various channels. These channels can include IBM Redbooks Company Insurance product-specific websites or partner and price comparison sites.
2. Requests are routed to the appropriate line-of-business (LOB) or provider, quoting processes through the messaging infrastructure.

3. In the business processes, several checks are applied to review the risk, eligibility, and profitability before passing to the appropriate insurance product application, to provide a basic quote by using the underwriting policies appropriate to that product.
4. This basic quote is passed back to the business process where promotion price adjustments can be made to the quoted price, and further risk assessment can be made before passing the quote back to the customer for acceptance.

4.4 IBM Redbooks Company Insurance scenario information model

To explain the scenario and how a BRMS can add value, we describe the simple information model that represents the insurance quote requests and responses that are being processed by the solution.

4.4.1 Insurance quote request

The information model for an insurance quote request is shown in Figure 4-2 on page 43 and consists of three main parts:

- Driver characteristics

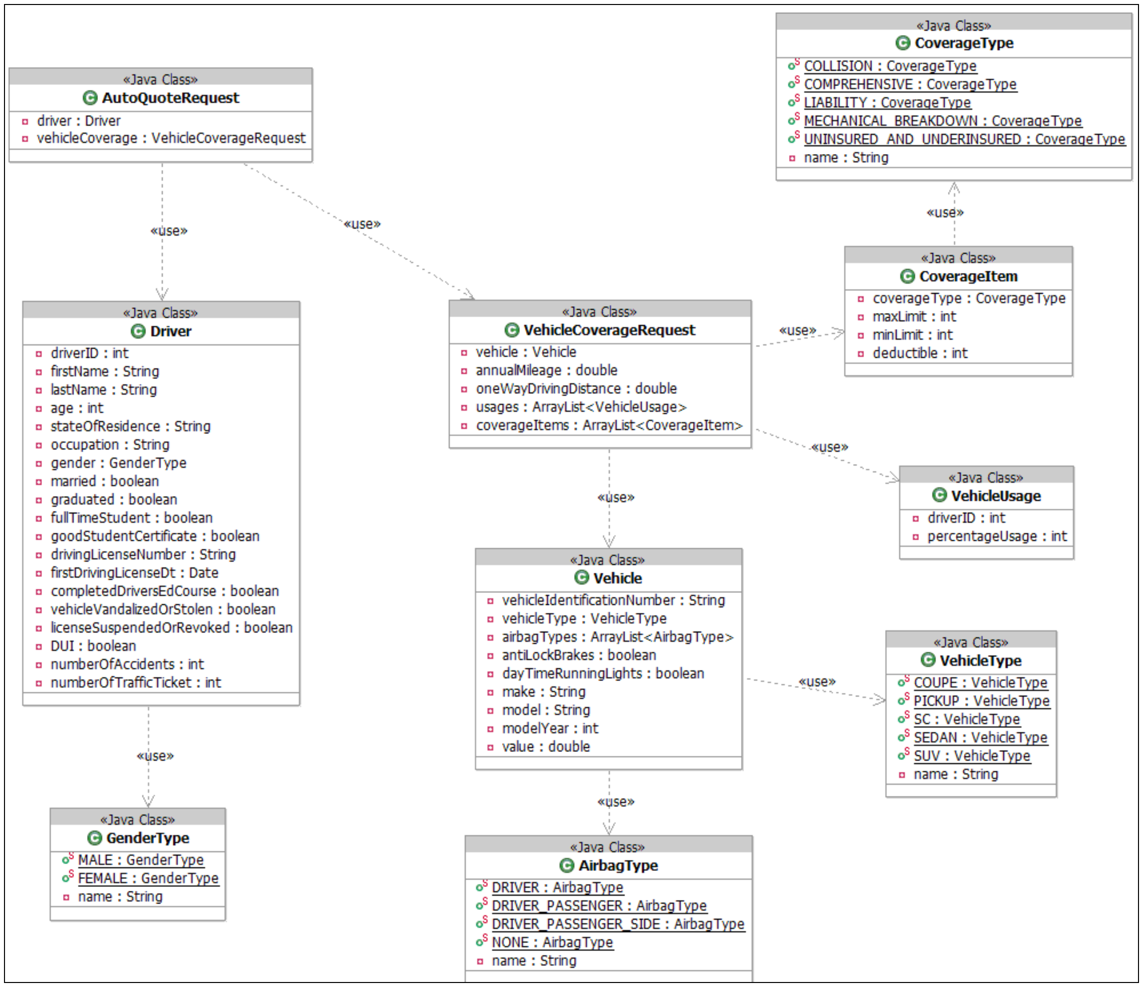
  This part provides personal information about the driver to asses the basic risk, such as age, state of residence, occupation, gender, and marital status. It also includes additional information that influences the risk, such as accidents or previous claims, driving offenses, or additional driving courses taken.

- Coverage

  The insurance coverage requested defines the type of coverage required (liability, comprehensive, and others) together with values of coverage and excess (deductible). Characteristics of vehicle usage that influence the coverage are also requested and can include annual mileage and proportion of time each driver uses each vehicle.

- Vehicle characteristics

  This part describes the basic make, model, year, and vehicle identification, together with categorization of vehicle type and a vehicle value. Additional information that influences the risk is also provided and can include airbag types and antilock brakes.



«Java Class» AutoQuoteRequest
driver : Driver
vehicleCoverage : VehicleCoverageRequest
«use»
«use»
«Java Class» CoverageType
COLLISION : CoverageType
COMPREHENSIVE : CoverageType
LIABILITY : CoverageType
MECHANICAL_BREAKDOWN : CoverageType
UNINSURED_AND_UNDERINSURED : CoverageType
name : String
«use»
«Java Class» CoverageItem
coverageType : CoverageType
maxLimit : int
minLimit : int
deductible : int
«Java Class» Driver
driverID : int
firstName : String
lastName : String
age : int
stateOfResidence : String
occupation : String
gender : GenderType
married : boolean
graduated : boolean
fullTimeStudent : boolean
goodStudentCertificate : boolean
drivingLicenseNumber : String
firstDrivingLicenseDt : Date
completedDriversEdCourse : boolean
vehicleVandalizedOrStolen : boolean
licenseSuspendedOrRevoked : boolean
DUI : boolean
numberOfAccidents : int
numberOfTrafficTicket : int
«Java Class» VehicleCoverageRequest
vehicle : Vehicle
annualMileage : double
oneWayDrivingDistance : double
usages : ArrayList<VehicleUsage>
coverageItems : ArrayList<CoverageItem>
«use»
«use»
«Java Class» VehicleUsage
driverID : int
percentageUsage : int
«use»
«Java Class» Vehicle
vehicleIdentificationNumber : String
vehicleType : VehicleType
airbagTypes : ArrayList<AirbagType>
antiLockBrakes : boolean
dayTimeRunningLights : boolean
make : String
model : String
modelYear : int
value : double
«use»
«Java Class» VehicleType
COUPE : VehicleType
PICKUP : VehicleType
SC : VehicleType
SEDAN : VehicleType
SUV : VehicleType
name : String
«use»
«use»
«Java Class» GenderType
MALE : GenderType
FEMALE : GenderType
name : String
«Java Class» AirbagType
DRIVER : AirbagType
DRIVER_PASSENGER : AirbagType
DRIVER_PASSENGER_SIDE : AirbagType
NONE : AirbagType
name : String

Figure 4-2 Insurance quote request

4.4.2 Insurance quote response

The responses to an insurance request can be as follows:

- A Validation Response when information is missing or inconsistent.
- An Entitlement Response when the driver-vehicle combination cannot be supported.
- An Insurance Quote Response providing quoted prices and breakdown.

These responses are shown in Figure 4-3. They show that a vehicle quote for the requested vehicle and driver, indicating the vehicle that has been quoted for (vehicle identification number, abbreviated as VIN), with a total price and a quote breakdown for each type of coverage that is requested. The quote also provides a list of global adjustments, applied to all types of coverage (for marketing and discount purposes). Messages indicating quote conditions are also provided. Each type of coverage is priced individually and includes a basic price and list of adjustments for that cover.

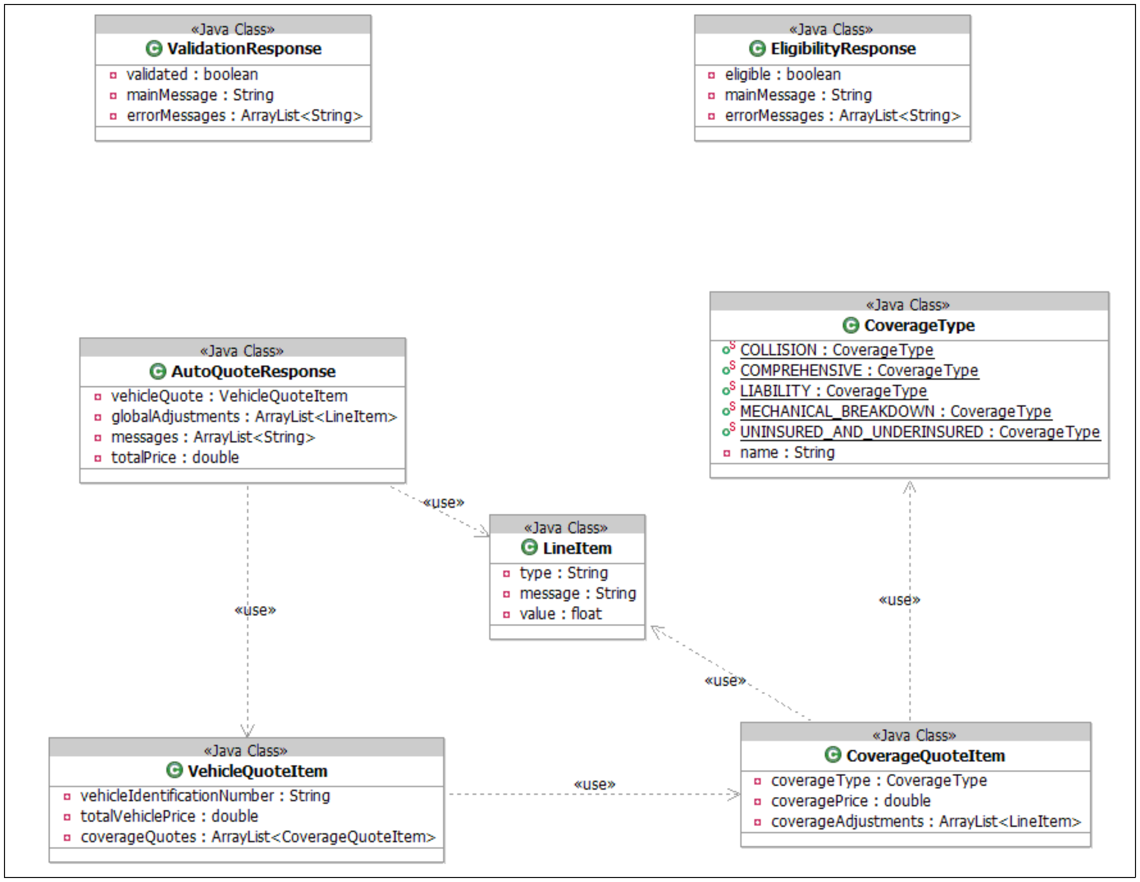

«Java Class»
ValidationResponse
validated : boolean
mainMessage : String
errorMessages : ArrayList<String>
«Java Class»
EligibilityResponse
eligible : boolean
mainMessage : String
errorMessages : ArrayList<String>
«Java Class»
CoverageType
COLLISION : CoverageType
COMPREHENSIVE : CoverageType
LIABILITY : CoverageType
MECHANICAL_BREAKDOWN : CoverageType
UNINSURED_AND_UNDERINSURED : CoverageType
name : String
«Java Class»
AutoQuoteResponse
vehicleQuote : VehicleQuoteItem
globalAdjustments : ArrayList<LineItem>
messages : ArrayList<String>
totalPrice : double
«use»
«Java Class»
LineItem
type : String
message : String
value : float
«use»
«use»
«use»
«Java Class»
VehicleQuoteItem
vehicleIdentificationNumber : String
totalVehiclePrice : double
coverageQuotes : ArrayList<CoverageQuoteItem>
«use»
«Java Class»
CoverageQuoteItem
coverageType : CoverageType
coveragePrice : double
coverageAdjustments : ArrayList<LineItem>

Figure 4-3 Insurance quote response

4.5 Values a BRMS can bring to our scenario

This section describes the values that a BRMS can add to our scenario and how the objectives, described in Section 4.2, "IBM Redbooks Company Insurance objectives and pain points" on page 40, can be achieved and the pain points alleviated.

4.5.1 To-be solution architecture

For the purposes of this scenario, we concentrate on modifications to the three main components (messaging, BPM, applications) of IBM Redbooks Company Insurance's existing infrastructure as shown in Figure 4-4.

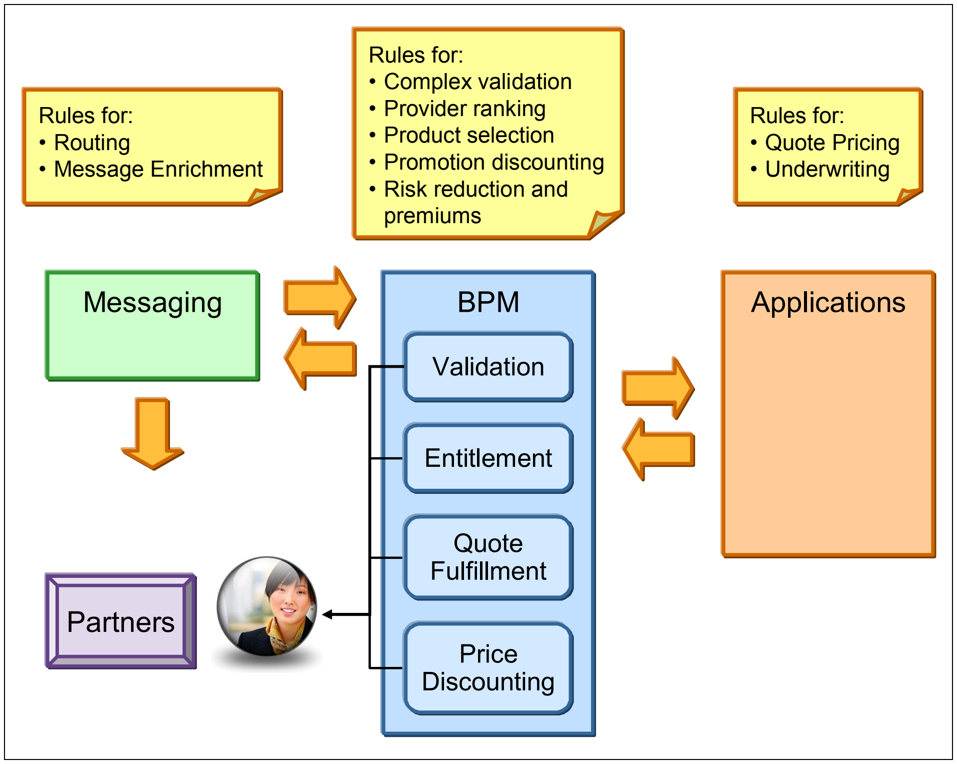


Figure 4-4 To-be solution architecture

The components are as follows:

- Messaging

  The messaging provides a means to accept requests for quotes from any of the channels and to route to the most appropriate insurance product process, human or manual processing, or business partner.

- Business or product process

  The process provides four main activities that involve decisions:

  - Validation of the request for a quote

    This activity must check the consistency and sufficiency of the information that is provided against the norms for the particular insurance product that is being requested.

  - Entitlement

    This activity checks that the requestor represents an acceptable risk before providing a quote for a particular product.

  - Quote fulfillment

    This activity routes the quote request to the appropriate application to apply the underwriting rules and create a basic price quote and corresponding conditions.

  - Price discounting

    This activity then adjusts the price that is passed back to the customer to account for marketing promotions or other risks that cannot be taken into account by the product application.

- Product applications

  Most of the individual organizations in IBM Redbooks Company Insurance have applications that provide quoting and policy creation and management for particular product sets. This basis for the existing business model and revenue will continue to be used in the new solution. However, the underwriting rules and pricing that are defined in these applications must be more configurable to allow a more flexible set of insurance products to be provided.

4.5.2 Messaging value propositions

In the messaging and network layers, business rules can be used to influence where messages are sent or modifications that are made to individual messages before being routed to a particular endpoint.

Routing

The solution architecture for IBM Redbooks Company Insurance includes a messaging and networking layer that can be used to route information to the most appropriate destination. We have seen that IBM Redbooks Company Insurance wants to quickly route messages to partners or appropriate lines of business, dependant on the message or quote request content. However, with the mergers, acquisitions, and rapidly changing business partnerships, the conditions for routing to a particular provider can vary. By using a BRMS, the rules for determining the routing criteria can be managed separately from the messaging structure. This way means that the rules can be changed quickly allowing new partners to be brought on or reallocated as required.

Message enrichment and transformation

Message routing is usually accompanied by message transformation to allow each destination to accept information in an appropriate form. Traditional approaches require a bespoke mapping between consumer and provider of messages. Many connectivity products provide mechanisms to alleviate the transformation, although in many business messages, the mapping is complex and cannot be performed. To alleviate the transformation, rules can be used in the following ways:

- Message enrichment can allow information to be added from rules that are based on existing fields. Effectively, a lookup for missing information can be provided (and managed) from within the rules system.
- Complex message field transformation allows many-to-many field mappings where the value of a required field can be determined only from a complex combination of the existing fields. Expressing this relationship as a business rule allows the complex relationship to be clearly expressed and managed if the relationship is found to be non-ideal or evolving.

4.5.3 Business activity value propositions

In the business activities, business rules can be used to characterize the way that a provider responds to any particular quote request.

Validation

The initial benefit is to quickly assess whether enough information has been provided in the quote request. Rejecting these requests quickly and giving customers a chance to provide the correct information (before they go to another supplier) can improve customer relationships and the chance of obtaining the business. In certain cases, having identified missing information, the possibility exists to supply that information from other sources, thereby easing the interaction with the customer.

Entitlement

Entitlement rules provide a first stage filter to check that the quote is actually based on drivers and vehicles that are acceptable risks. Based on a quick assessment of the quote request information provided, high-risk patterns can be detected and rejected, therefore avoiding further processing of this non-viable request.

This form of “go-or-no-go” decision is common in process decision nodes. This form can be applied automatically or be combined with a human interaction to advise the condition that has been detected and the rationale for it. The human operator can then either follow that advice or override it according to the way the decision is realized in the business activity.

Quote fulfillment integration

Although the main quoting and underwriting is provided by applications in our scenario, there might be some fine-grain routing to the appropriate provider product application or database. This case is especially true where a single process is used to support multiple products and routing to the appropriate product (together with any mapping of information) might have to be undertaken either within the process logic or in the messaging layer from the process to the actual back-end application.

Discounting and risk assessment

The discounting and risk adjustments are independent of the underwriting rules and are applied across the quotes from multiple applications. The decisions and adjustments are also made by separate areas of the business, based on marketing strategies or fraud and risk detection analytics. In these cases, the adjustments are based on rules that compare the quote characteristics with target market characteristics.

4.5.4 Application value propositions

The use of business rules in conjunction with applications is often for reasons that differ from the messaging and activity use of rules. Often, existing applications have these behavior “rules” hard-coded into the applications; the key value proposition of a BRMS is to extract those rules out of the application and allow them be managed and changed to better align with the changing business need. Two insurance examples are highlighted in this section, but the value propositions and details of the rules vary greatly with the industry and application.

Quote pricing

Insurance product quote pricing is based predominantly on a balancing of risk and return. This balance is based on an understanding of those risks by driver and vehicle characteristics. These demographics change with time and extracting the pricing rules out of the applications allows a greater insight into how pricing is calculated across the products and a better focus on pricing to the specific risks in those quote requests.

Underwriting

Individual insurance products do not normally retain the capital to cover instance risks but outsource it to underwriters. An individual product might therefore use several underwriters and must ensure that the policy quote is applicable to the underwriter that is selected. The contracts that are established between the insurance provider and underwriter are based on conditions for acceptable policies and the fee that the underwriter requires from that type of risk. The underwriters that are used by an insurance product can change, therefore the rules that are used in pricing and underwriter selection can also change. Using rules in this way allows the insurance provider to change underwriters to get the best deal and profitability for their policies.

5

Identifying the patterns in general solutions

In Chapter 4, “Scenario and the solution architecture” on page 39, we identified the value propositions that customers can achieve by using a BRMS in their solutions. This chapter provides an overview of common patterns that can be used to realize those solutions.

This chapter contains the following topics:

- Decision patterns: This chapter looks at the decision patterns that use rules to achieve the value propositions being sought.
- Integration patterns: This chapter identifies integration patterns that can be used to integrate rules into the solution.

These topics are deliberately kept separate because describing all possible combinations of decision patterns and integration patterns is not possible. In later chapters, the suitability of integration patterns are brought out for each of the decision patterns. This approach can help you to choose an appropriate decision pattern and then an integration pattern that can meet your requirements.

5.1 Decision patterns

The key goal of a BRMS is to extract the decisions from the applications so that they can be managed centrally and changed at a rate that is required to suit the business. This section describes ways of making decisions by using rules that are commonly encountered in solutions. Understanding the decisions helps us identify those aspects of the solution that can usefully be extracted into the BRMS to deliver the needed business agility.

Each pattern is considered in a similar way, as shown in Figure 5-1.

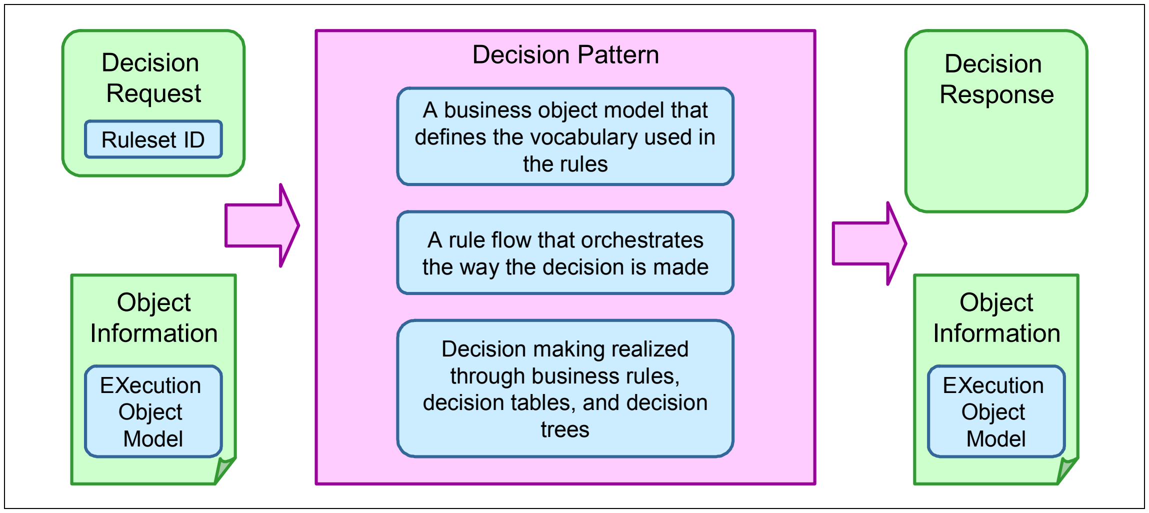

Decision Request
Ruleset ID
Object Information
EXecution Object Model
Decision Pattern
A business object model that defines the vocabulary used in the rules
A rule flow that orchestrates the way the decision is made
Decision making realized through business rules, decision tables, and decision trees
Decision Response
Object Information
EXecution Object Model

Figure 5-1 Decision pattern overview

Your application can "outsource" a particular decision to the BRMS through some decision request and identify the "decision" by passing it in some form of rule set identifier. It will also pass in the context and object information about which the decision is to be made.

In the BRMS, the object information (Execution Object Model, XOM) is translated into a vocabulary (Business Object Model, BOM) that is used in the rules to express the decision-making. On receipt of a decision request, a rule flow (defined for the rule set) routes the information through the various defined rules. The BRMS provides business rules, decision tables and decision trees that can all be used to make a decision, appropriate to the needs of the application or solution.

This decision response is then passed back to the application together with any updated or new information provided by the decision and translated into a form the application understands.

This remainder of this section describes typical patterns that solutions and applications use for making decisions.

5.1.1 Validation pattern

The most fundamental pattern is a “go-or-no-go” decision based on the information available. With this pattern, policies can be set up that define the conditions under which application processes or transactions can proceed. Examples include the eligibility and validation rules from our scenario. Governance and life cycle management also uses validation patterns to ensure that all activities and information needed have been completed before an artefact is allowed to move onto the next stage of its life cycle.

This pattern is illustrated in Figure 5-2.

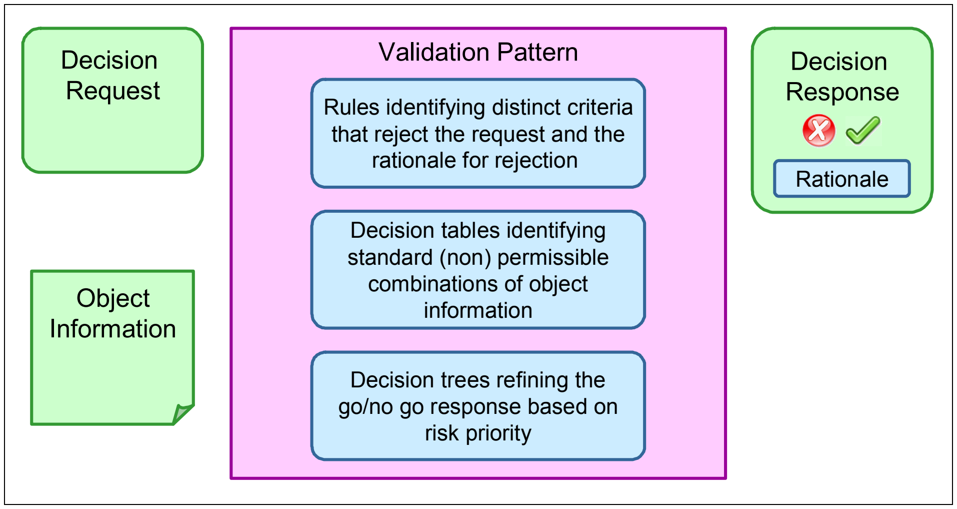

Decision Request
Object Information
Validation Pattern
Rules identifying distinct criteria that reject the request and the rationale for rejection
Decision tables identifying standard (non) permissible combinations of object information
Decision trees refining the go/no go response based on risk priority
Decision Response
Rationale

Figure 5-2 Validation pattern

Business rules can be used to identify the specific situations that must be rejected based on key criteria in the object information. The rule can also provide a rationale to explain the decision that can be passed back to the application. This rationale is important if a human is involved because it explains the decision and encourages the desired behavior for the next time. In certain situations, an advantageous approach is to also identify the corrective action to take in the case of a rejection.

Decision tables allow more complex conditions to be tabulated which effectively define combination ranges of different criteria that are acceptable or not. One of the advantages of rules is that the decisions can be made more precise as an understanding of the risks and patterns evolves. In some of these refinement cases, decision trees can be used to make the obvious decisions first and then use successively more information to obtain a more precise decision.

5.1.2 Enrichment pattern

One problem of heterogeneous solutions is that the separate applications work with different representations of information. In certain situations, information might be needed by one application and is not provided directly or in the correct form by the client application. In other cases, the generation of a piece of information is performed differently by separate applications. With the enrichment pattern, rules can be used to provide a systematic and managed way of providing that information. An example of this pattern is car insurance group information, grouped according to make and model. The key value proposition of this pattern is having a consistent managed way of deriving this information from a variety of information sources.

The enrichment pattern is illustrated in Figure 5-3.

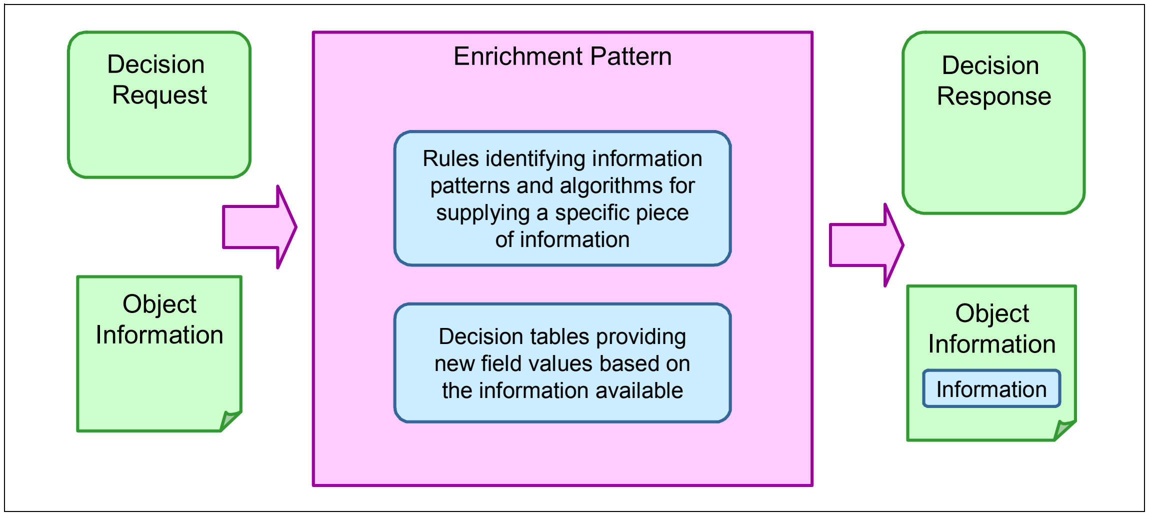

Decision Request
Object Information
Enrichment Pattern
Rules identifying information patterns and algorithms for supplying a specific piece of information
Decision tables providing new field values based on the information available
Decision Response
Object Information
Information

Figure 5-3 Enrichment pattern

In its simplest form, rules can identify the conditions and the algorithms for calculating the missing information. Other forms of decision such as decision tables may be applicable when considering multiple factors in generating a response, for example, engine size, type of vehicle, and vehicle safety features.

5.1.3 Classification pattern

One frequent pattern that is used in solutions is that of classifications, domains, and enumerations: identifying a known situation or class of information. The classification system or taxonomy represents the current understanding, by the business, of the possible situations that can occur. The rules then match the object information against those possible situations and select the best fit. In our scenario, the routing might be based on a quote classification system that indicates the organization and product suitability. Based on the classification response, messages can be routed to an appropriate supplier for that classification.

This approach has also been used for identifying unknown situations or information that cannot be classified. Consider fault identification where certain combinations of information are recognized, a class of fault can be automatically diagnosed and processes set in place to resolve. If the fault cannot be classified, the processing must go through a manual routing to diagnose the fault. After this task is performed and a new solution is provided, this new classification can be added to the rules and further instances of the fault can be automatically resolved.

The classification pattern is shown in Figure 5-4.

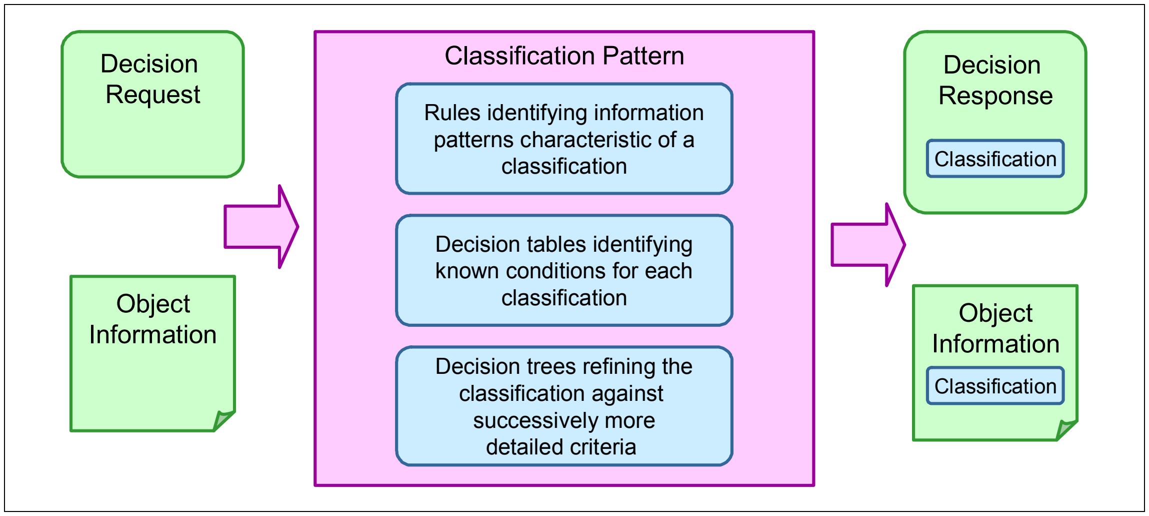

Decision Request
Object Information
Classification Pattern
Rules identifying information patterns characteristic of a classification
Decision tables identifying known conditions for each classification
Decision trees refining the classification against successively more detailed criteria
Decision Response
Classification
Object Information
Classification

Figure 5-4 Classification pattern

As with previous patterns, rules and decision tables can be used to identify the classes, based on particular information patterns. Because classifications are normally hierachic, decision trees can be used to perform coarse level classification first and then finer, sublevel classification as the tree is traversed.

5.1.4 Calculation pattern

Many business decisions that must be made (and changed dynamically to suit the market) relate to prices, margins, schedules, and other criteria. This information has a complex relationship to the object being considered and the context in which it is considered. Consider the pricing rules for insurance: They depend on the car being insured and the driver of the car. Discounts must also be applied according to market promotions in place. The result is that the decision actually produces new information objects that are related to the objects in the original decision request. In our case, the rules provide a list of quotes for the driver and car combination, as shown in Figure 5-5.

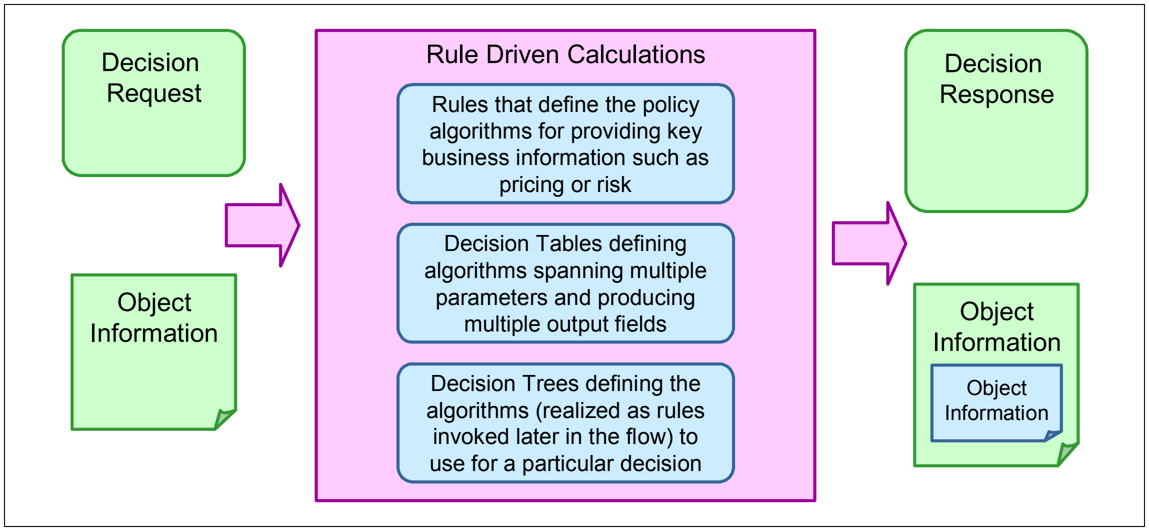

Decision Request
Object Information
Rule Driven Calculations
Rules that define the policy algorithms for providing key business information such as pricing or risk
Decision Tables defining algorithms spanning multiple parameters and producing multiple output fields
Decision Trees defining the algorithms (realized as rules invoked later in the flow) to use for a particular decision
Decision Response
Object Information
Object Information

Figure 5-5 Calculation pattern

This generic pattern can be used in many ways to use all aspects of business rules, decision tables, and decision trees. In complex decision-making, a necessary step might be to design the flows so that a pricing policy is first determined (possibly using a decision tree) and then the selected policy rules are applied. This method allows the strategy about pricing to be separated from the details of the individual policy pricing rules.

5.1.5 Scoring and selection pattern

The patterns described in the previous sections considered a single context and made a decision about that object. Another type of pattern exists in which an application has several available options and has to decide which option to select. Many examples of this pattern exist, in terms of routing selection, vendor selection, provider selection, and so on. In our scenario, we can consider the routing based on service performance from various providers.

This pattern can also be used for resource optimization if the options for the resources are passed into the rule engine as part of the transaction. Effectively, each option is ranked against some cost function and the least costly option with the greatest value is selected based on the relative scoring.

The principles of the scoring and selection pattern are described in Figure 5-6.

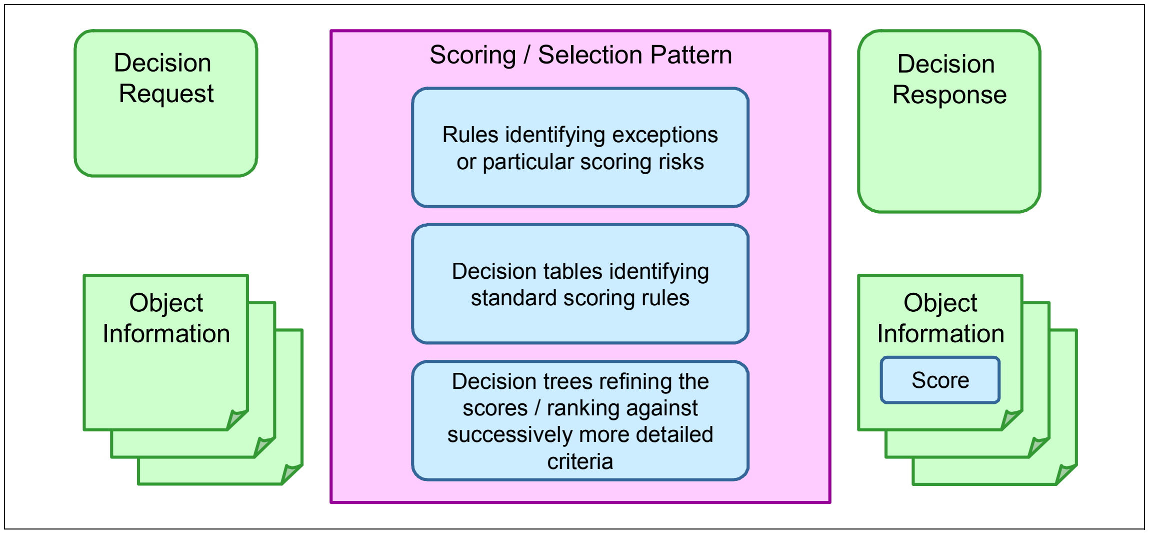

Decision Request
Object Information
Scoring / Selection Pattern
Rules identifying exceptions or particular scoring risks
Decision tables identifying standard scoring rules
Decision trees refining the scores / ranking against successively more detailed criteria
Decision Response
Object Information
Score

Figure 5-6 Scoring and selection pattern

The application or solution that “wants” to make the selection, packages together a list of options to be assessed (the provider status) with the context (the quote request). Each option is then passed through the rules that establish a score, based on opportunity and cost, and risk metrics that are appropriate to the business. The set of options with their scores is passed back to the application, which can then make a selection based on the score.

This approach allows the basic selection decision logic to remain in the application but offers the ability to change the decision outcome though managed consistent rules that can evolve over time. As new options become available (new service providers are available), the selection policies can immediately be applied consistently to them.

5.1.6 Message transformation pattern

Enterprise Application Interaction (EAI) middleware provides connectors that transform information from one representation to another. In most cases, this transformation is based on information architectures or schemas that are defined for the message structure. As modern solutions involve more heterogeneous applications that use various standards (and versions of the standards), the transformation cannot be undertaken based simply on the structure. Rules can use the information content and also the structure to define the transformation and thus cope with these situations. When combined with the previously described patterns, the information can be validated and made more consistent throughout the enterprise, reducing misinterpretation and improving interoperability.

The message transformation pattern is illustrated in Figure 5-7.

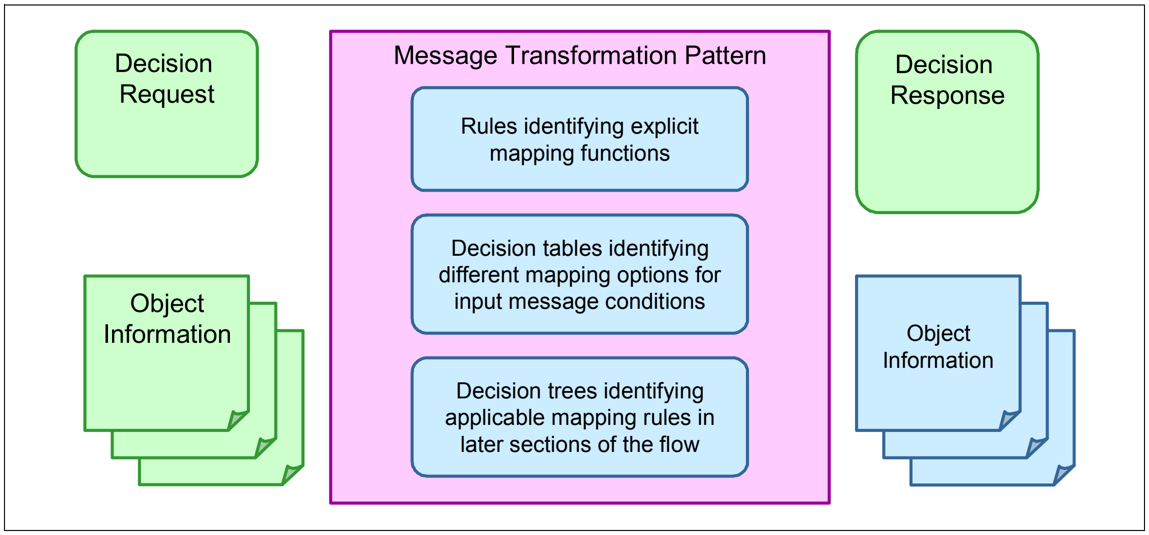

Decision Request
Object Information
Message Transformation Pattern
Rules identifying explicit mapping functions
Decision tables identifying different mapping options for input message conditions
Decision trees identifying applicable mapping rules in later sections of the flow
Decision Response
Object Information

Figure 5-7 Message transformation pattern

As part of an interaction, a message comes in with a number of components, for example, a healthcare ICD-10[1] message. The existing solutions require separate representations for this message, such as ICD-9 formats. However, the mapping of codes from ICD-10 to ICD-9 are not unique. The rules can be used to identify the recommended transformations and decision tables that are used to identify the actual code mappings. The result is an object in the ICD-9 format that can be used with the existing solution.

In practice, the structural transformations do not need to change that often and are better handled by existing transformation connectors and extenders. However, using rules to allow the transformation of the mapping codes means that changes to the recommended code transformations can be accommodated without redeploying the connectors.

5.2 Integration patterns

The technical patterns described in this section represent the various ways of invoking custom web services. The choice of integration pattern is influenced by the IT infrastructure that is being used in the solution and the non-functional trade-offs that are applicable for each pattern.

5.2.1 Web services: Hosted transparent decision services

Hosted transparent decision services (HTDS) provide a simple and quick way of accessing the rule applications that have been deployed to a custom web service from Rule Studio of Rule Team Server. The Web Services Description Language (WSDL) that describes the service can be obtained from the custom web services and then used to generate client code in the calling application or process, as shown in Figure 5-8 on page 60.

[1] The International Statistical Classification of Diseases and Related Health Problems 10th Revision (ICD-10)

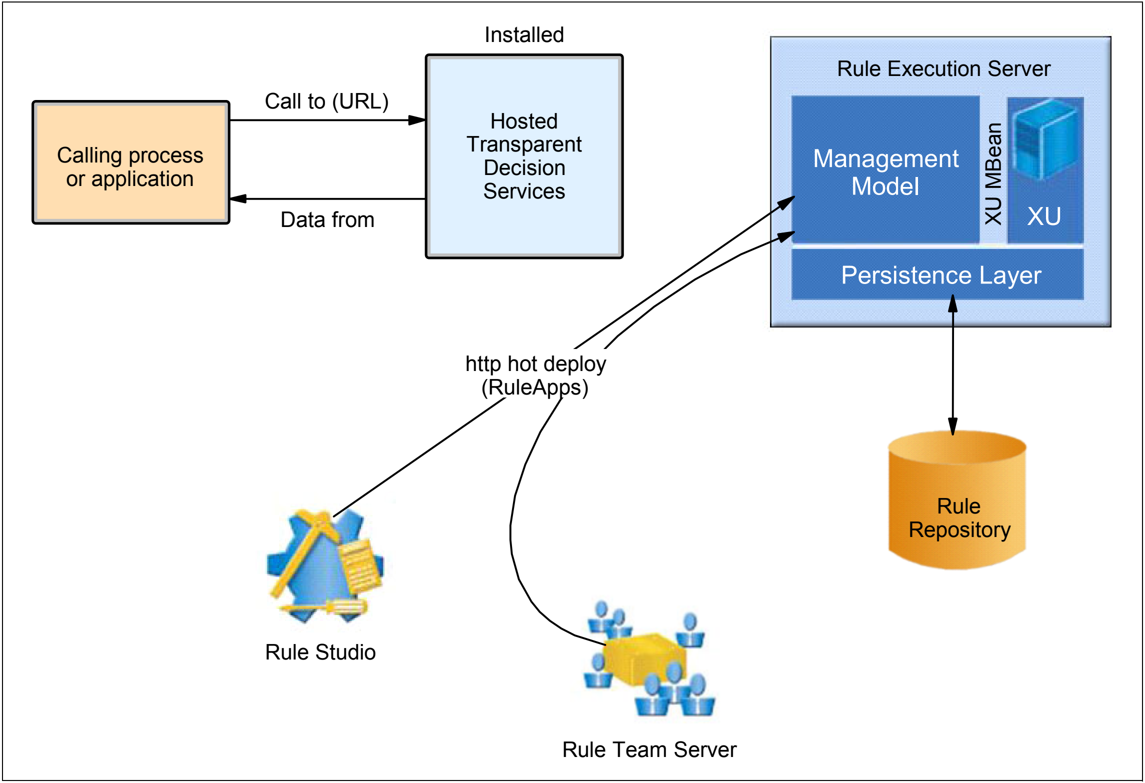


Figure 5-8 Hosted transparent decision services

Although this pattern of usage is the fastest and simplest, the following limitations currently exist:

- There is no control over the name and namespace of the service. This limitation can cause problems for governance and architectural best practices.
- The BOM that is used must be generated from a schema or simple Java XOM; Java XOMs that describe complex types are not supported.
- All services that are generated have the same namespace. This limitation can cause problems if a client application needs to invoke multiple decision services. This problem is the result of conflicts in the java classes that are generated by the client tooling.
- The services do not use an underlying web service stack so imposing security or quality of service policies on them is difficult.

The ease of use and integration of HTDS does however mean that it is often the first choice when evaluating the value of rules in solutions.

5.2.2 Web services: custom

Overcoming the limitations of HTDS is possible by developing and hosting separate web services that then use other integration patterns that map from the web service implementation to the Rule Execution Server Execution Unit (XU). Tooling is available to help develop these decision services, the most advanced of which supports the monitored transparent decision service, as shown in Figure 5-9.

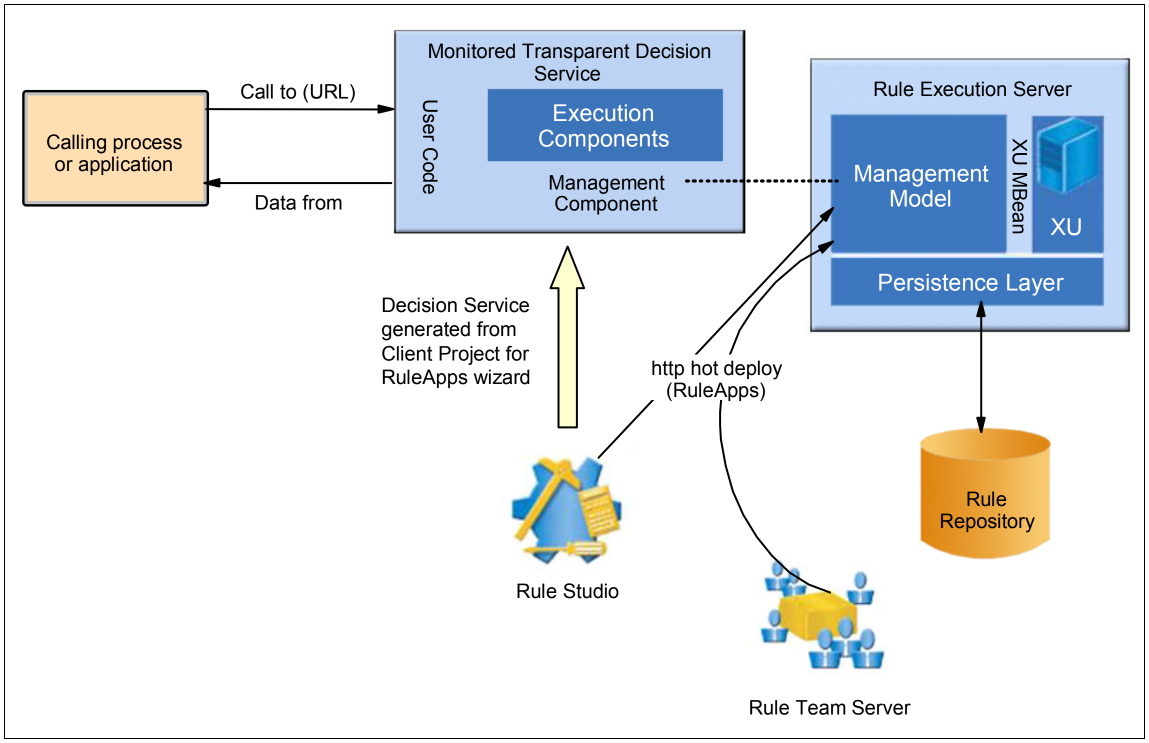


Figure 5-9 Monitored transparent decision service architecture

Wizards within Rule Studio can generate web service implementations (and clients) conforming to JAX-WS 2.1.1. Because these services can optionally include monitoring, statistics can be accessed through the Rule Execution Server (RES) console. To support this version of JAX-WS, the application server must run by using JDK 1.6 or later. At the time WebSphere ILOG JRules 7.0 was released, only Tomcat and JBoss supported JDK 1.6. If you want to use web services (custom) in conjunction with another application server (such as WebSphere Application Server), you can generate the web services (custom) code for a JBoss or Tomcat server and adapt it to be compliant with your application server.

Web service generation tools can be used to generate web service implementations (and clients) that expose the services required. Descriptions of these tools can be found in *IBM Webservices Handbook for WebSphere Application Server 6.1*, SG24-7257.

The following information must be accounted for so that the RES can be invoked in the implementation:

- The rule application (and version): used to specify the rule path and service
- The rule set (and version): used to specify the rule path and operation
- Input parameter names
- Input parameter types: Java, or string for XML XOMs
- Output parameter names
- Output parameter types: Java, or string for XML XOMs
- The XOMs: schemas and Java source that are used to define the types

Generally, three approaches are possible for generating web service implementations:

- Generate a web service from a Java class that specifies the rule sets and signatures as operations on the class.
- Generate a web service from a WSDL file that defines operations that match the rule sets and takes in the schema to define operation signatures.
- Generate a web service implementation from a WSDL file, which defines operations that match the rule sets and uses xsd:any for parameters. The parameters are then passed as XML strings to the RES, avoiding the wide range of Java classes generated by the previous bullet

5.2.3 J2EE synchronous custom web services patterns

Many consuming applications are hosted on a J2EE server; the simplest integration pattern is to use the J2EE facilities that are provided by a Rule Execution Server. JRules provides two types of synchronous sessions:

- Stateless session

  This session is the most commonly used and preserves no information from one invocation to the next. Each decision is made as part of the call; all the needed information must be provided through the input parameters or through calls made from the BOM.

- Stateful session

  This session allows information to be maintained in memory, so that a set of information can be built up and accessed by the rules. Stateful sessions are not discussed further in this book.

This overall J2EE Rule Execution Server application architecture is shown in Figure 5-10.

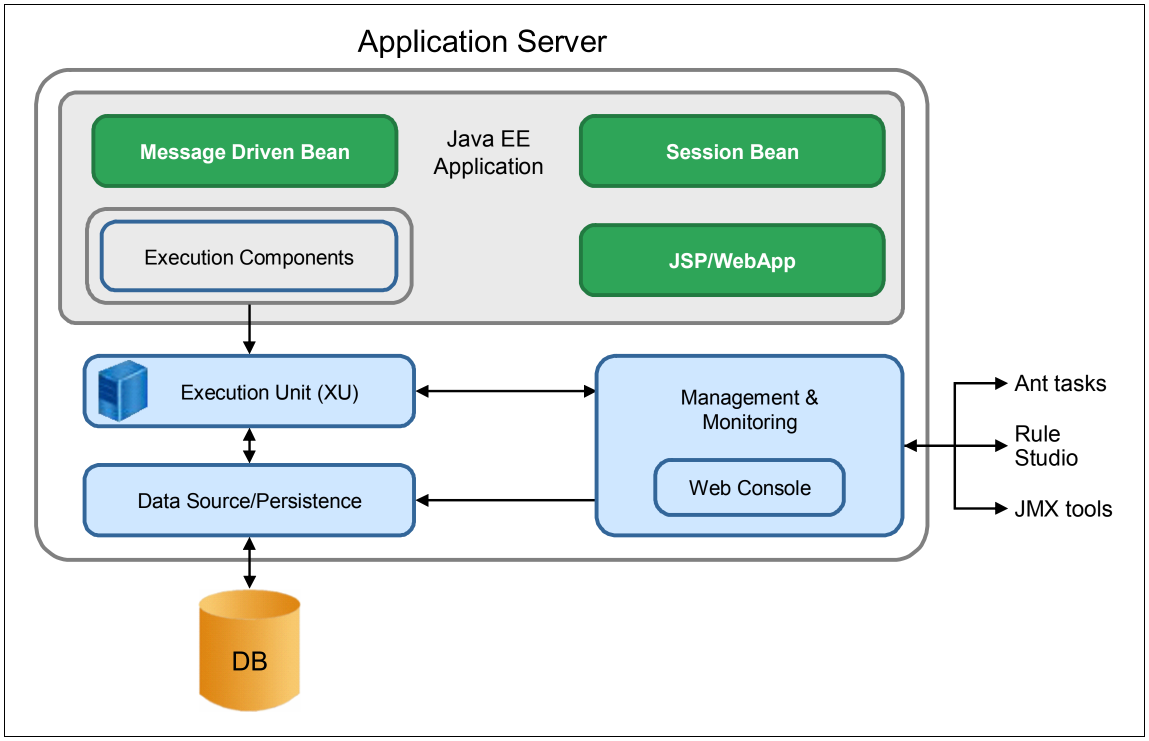


Figure 5-10 J2EE Rule Execution Server architecture

There are two synchronous patterns available, POJO and EJB, and one asynchronous pattern, as described in 5.2.4, “J2EE asynchronous custom web services patterns” on page 65.

POJO

A plain old Java object (POJO) represents a simple interface that can be used by any Java application programming model or any of the advanced models such as Service Component Architecture/Service Data Object (SCA/SDO). In this invocation pattern, the rule application, XU, is invoked directly from the application using the session caching that is provided for that application. See Figure 5-11.

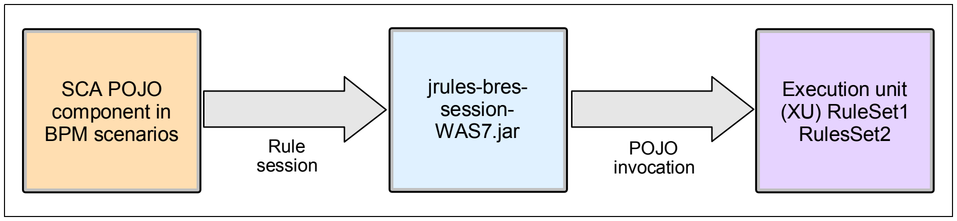

SCA POJO component in BPM scenarios
Rule session
jrules-bres-session-WAS7.jar
POJO invocation
Execution unit (XU) RuleSet1 RulesSet2

Figure 5-11 Using JRules POJO sessions in an application

EJB

JRules provides Enterprise JavaBeans (EJB) modules that can be deployed as part of a J2EE application allowing decisions to be made available across the J2EE cluster. This technique adds scalability to solutions and provides transactionality and the ability to remotely access the server EJB. A simple overview of using an EJB to make decisions in a distributed J2EE application is shown in Figure 5-12.

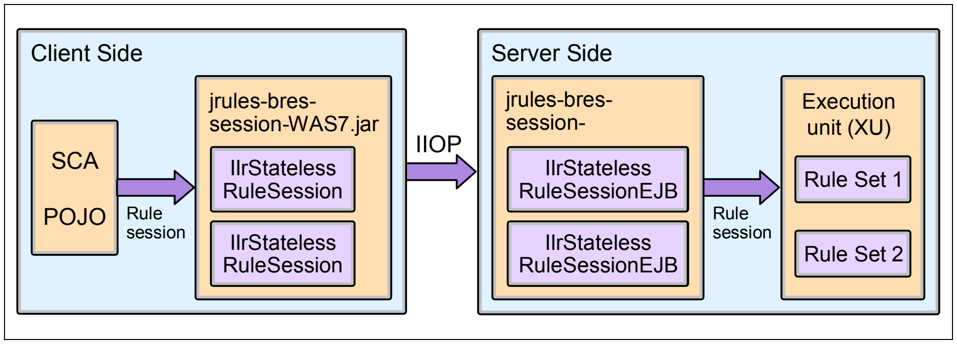


Figure 5-12 Using JRules EJB sessions in an application

5.2.4 J2EE asynchronous custom web services patterns

In addition to the synchronous patterns described previously, JRules also support asynchronous invocation through two methods, POJO and MDB. This pattern is also suitable where queuing is a requirement, for example, where rule requests must be queued if the Rule Execution Server is offline for maintenance.

POJO

A non-blocking call to the RES can be made by declaring and registering an observer that will be invoked when the rule execution fails or completes.

MDB

JRules supports JMS 1.1 by using message-driven beans (MDBs). This integration requires a messaging run time in the J2EE application server, for example, WebSphere MQ queue managers, and associated artefacts such as queues and Java Naming and Directory Interface (JNDI) objects.

This integration requires an MDB that is local to the RES that receives request messages from the client (through a JMS queue), passes them to the RES (using the WebSphere ILOG JRules API), and then returns a reply message from the RES to the client (through a JMS queue). WebSphere ILOG JRules provides an MDB, but you may write your own if you require bespoke functionality. For instance, the WebSphere ILOG JRules MBD supports only JMS object messages; if you want to pass JMS text messages, you can write your own MDB. The MDBs are deployed as part of the J2EE applications as shown in Figure 5-13 on page 66.

Messages come in and are directed to the queues or topics that the MDBs are listening on. This way can be configured for the applications that host the MDBs using the facilities of the MDB container (application server) tooling. The messages must contain properties that are used to filter and route the message body to the correct rule application and also contain the parameter name value pairs needed for the particular rule set. On completion of the decision, the response is captured in a response JMS message and directed back to the appropriate queue.

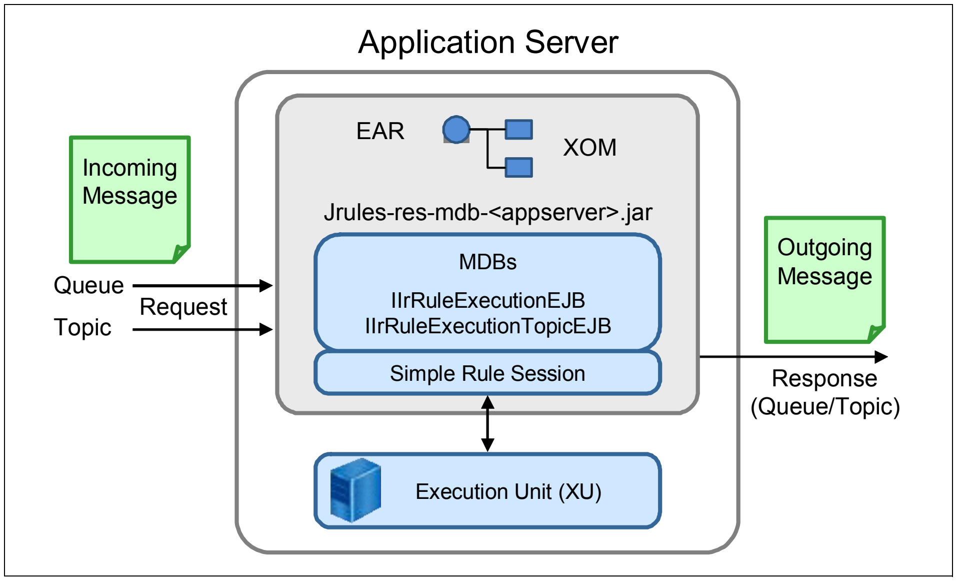


Figure 5-13 Message-driven bean architecture

5.2.5 JRules Java SE Rule Execution Server

In several situations, a full J2EE application server is not appropriate to a solution but the same rule management and invocation patterns must be provided. This style is supported by using the Java Platform, Standard Edition (SE) Rule Execution Server deployment shown in Figure 5-14 on page 67.

Note: The runtime implementation must support at least Java SE 1.5.

Using this style of integration is likely to be more appropriate if rules are not shared across separate domains (such as Connectivity and BPM) or the requirement for rules is only being considered within a single component.

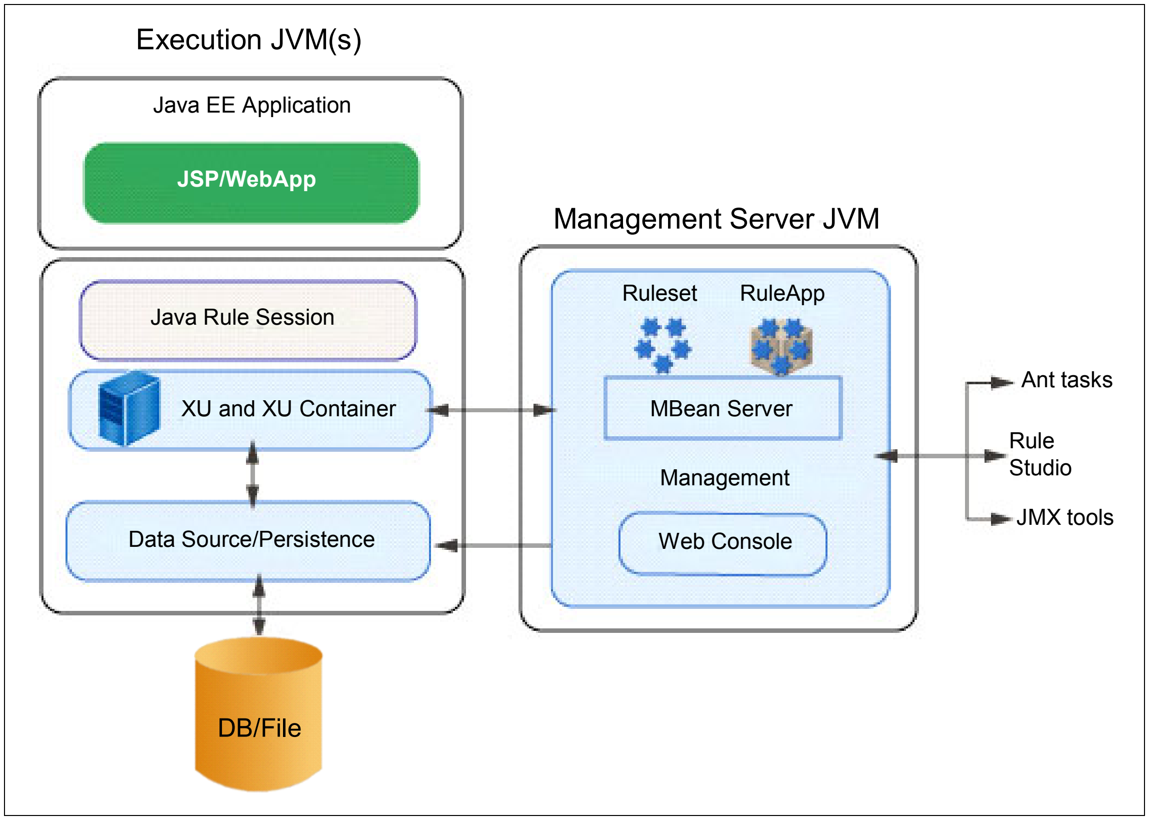


Figure 5-14 Java SE Rules Execution Server architecture

In this configuration, the application requires only a JVM to host the Rule Execution Server. The runtime implementation must support at least Java Platform SE 1.5. The interaction patterns are similar to those described for the POJO J2EE pattern but several constraints and disadvantages exist with using this pattern:

- All execution is within the one JVM, so be sure that memory use does not become excessive.
- The session pooling model differs, which means that hot deployment of rules is not guaranteed by default.
- Centralized management is still possible if a web container (such as Tomcat) is available, allowing the use of Rule Team Server and Business Rule Management.
- There are no statistics and central usage benefits.

5.2.6 JRules Java SE engine

Another option, which has several disadvantages, may be considered: using the Java SE rule engine and its API as shown in Figure 5-15.

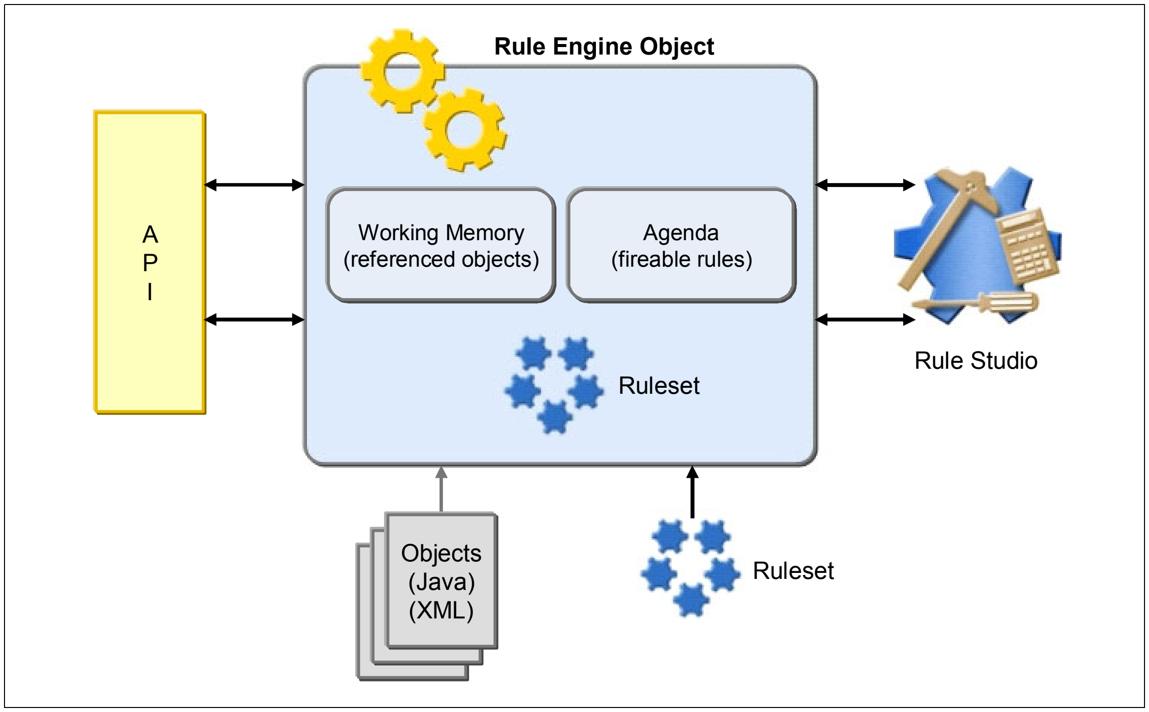


Figure 5-15 J2SE rule engine

This solution has several disadvantages:

- No rule engine or connection pooling requiring all rules to be held in memory or reloaded when needed
- No rule management capability requiring rules to be deployed statically to the engine
- No management facility and thus no rule sharing or business user involvement

5.2.7 Rules for COBOL

Many mainframe applications are written in COBOL which does not efficiently support calling out to a Java rules engine or RES. In this case, another approach is taken that preserves the ability to support and manage the changing rules but relies on technologies that are native to the COBOL and mainframe environments. See Figure 5-16.

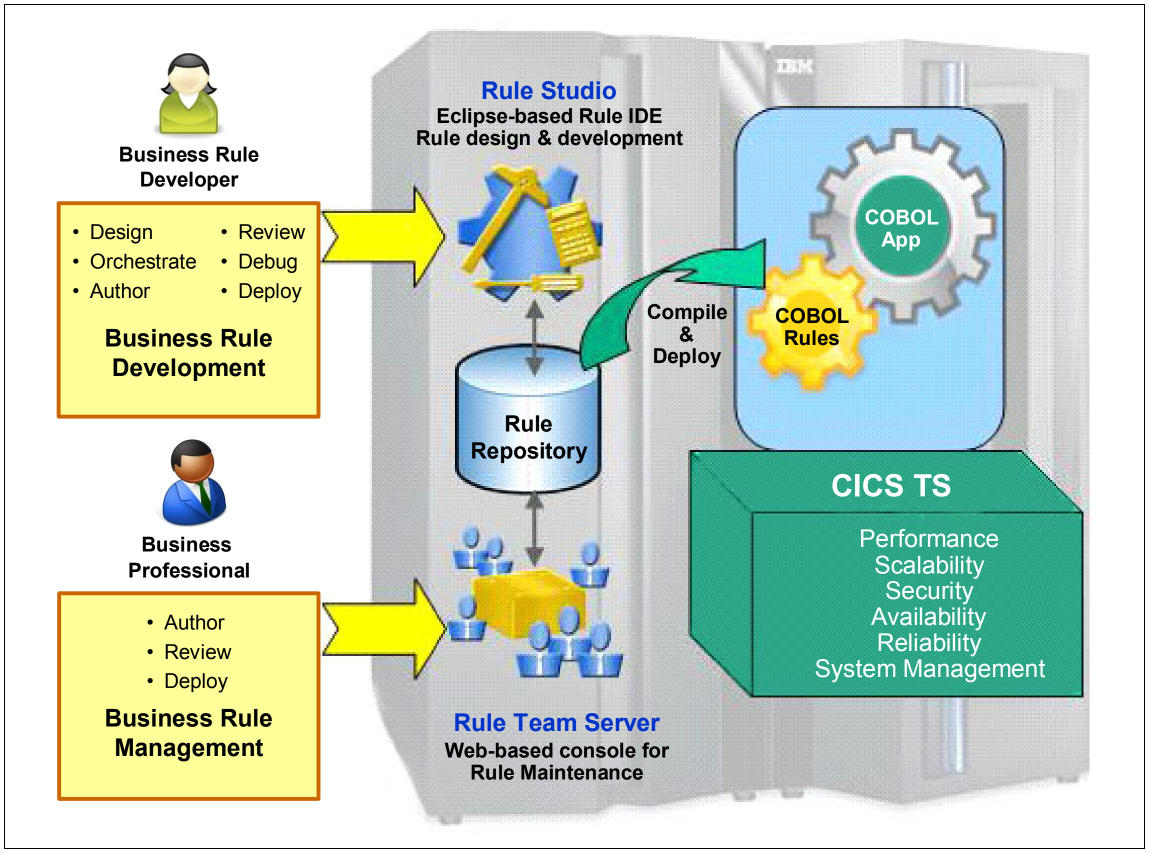


Figure 5-16 Rules for COBOL deployment pattern

In this pattern, the rules are compiled and deployed into the existing COBOL applications as a module. The module interface is well defined, allowing it to be replaced at a faster rate than the existing applications. This pattern is attractive for mainframe solutions because a compiled solution has better performance and fewer architecture changes.

6

Existing applications

This chapter contains the following topics:

- Rule processing within existing applications
- Choosing the best technology pattern for the application environment

6.1 Rule processing within existing applications

In this section, we look at rule processing within existing applications.

6.1.1 Introduction

The scenario in Chapter 4, “Scenario and the solution architecture” on page 39, has existing applications that contain the pricing rules for generating car insurance quotes. Our fictional company, IBM Redbooks Company Insurance, must be able to make regular changes to these rules, to respond quickly to new market opportunities, or to tweak the rules to maximize profit, or minimize risk to the business. When a change needs to be made, an important aspect is that the changes be made live as quickly as possible. With the rules built into the applications, the speed at which these changes can be made depends on the following requirements:

- The availability of resources in the IT department to make and test the necessary code changes
- The need for the business analyst to clearly define and communicate the changes to the IT team
- The length of time taken from making a code change, through the various stages of testing, quality assurance, and deployment into production

Depending on the company, the speed with which a change is made can range from days to months. This delay can affect the ability of the business to respond dynamically to market requirements.

In this chapter, we examine the types of rules that are common in existing applications, and we discuss the benefits of extracting those rules from the existing applications into a BRMS. We then look at how the technology patterns introduced in Chapter 5, “Identifying the patterns in general solutions” on page 51 are applicable when integrating rules with the following application environments: J2EE, J2SE, CICS, IMS, and batch applications on z/OS.

6.1.2 Integration scenarios

Although our scenario is about a fictional insurance company, the equivalent needs of being able to change existing business applications in a dynamic fashion and being unable to meet these needs because of the requirement to change the existing IT applications are appropriate to many industries. Examples include the following rules:

- Rules governing sales promotions
- Rules governing eligibility of customers for certain products
- Rules for offering mortgages, bank accounts, and other financial services
- Rules governing pricing policies
- Rules for fraud prevention and detection
- Rules for maintaining customer loyalty and retention

6.1.3 Benefits of integration

By extracting the rules from the existing applications, and placing them in a BRMS, one or both of the following dependencies on the IT department can be removed:

- The business analyst can make the changes to the rules and perform some testing of the changes, thus removing the requirement on the IT department to make the code changes. This approach also removes the need for the business analyst to communicate the required changes, where there is a risk that the changes might be misinterpreted or made incorrectly.
- Depending on the architecture of the BRMS environment, the business analyst may be able to “hot deploy” the rules, that is, make the rules in a live situation in production without requiring any interactions with the IT department. This approach greatly decreases the time spent making the rules active. We discuss the technology patterns for the application environment later in this chapter.

6.1.4 Solution life cycles

Where business rules are used within existing applications, solution life cycles enable the applications and the rules to have their own life cycles, and enable the rate of change of these components to differ so they align with the required business needs.

In many business applications, the rate of change of business rules changes far more frequently than other components of the application code. The other major factor to consider is that the role in the organization that focuses on owning and implementing the applications normally differs from the role that defines and governs the business rules and policies that the business uses.

In certain cases, the same person might develop the applications for small projects and also define simple rules for the application.

Where the business focusses on capturing its enterprise business policies and rules more generally, and assesses which ones are applicable to specific major decisions (such as pricing, quotes and eligibility), the organizational group doing this typically differs from the application owners.

After the business rules are extracted from the existing applications, they can be changed independently with a final regression test to ensure that the complete solution meets its business requirements, which optimizes the business agility and time to market.

6.2 Choosing the best technology pattern for the application environment

Chapter 5, “Identifying the patterns in general solutions” on page 51 introduced the technology patterns that are available for integrating with ILOG. In this section, we look at the advantages and disadvantages of these approaches with respect to the following application environments: J2EE, J2SE, CICS, IMS and Batch.

In general the technology patterns fall into two categories:

- The rule processing is performed in a centralized Rule Execution Server. Existing applications make a remote call out to the centralized server passing the input data needed to run the rules. The results of running the rules are returned to the existing application. For example, an existing application makes a web service call to the Rule Execution Server to run the rules.
- The rule processing is run as close to the existing application as possible. For example, a Java program calls a Rule Execution Server that is embedded in the same Java virtual machine (JVM).

6.2.1 Advantages and disadvantages

The centralized Rule Execution Server has the following advantages and disadvantages:

- Running the Rule Execution Server in a J2EE environment provides the complete set of capabilities, such as running of rules and hot deployment of new rule sets.
- Rules can be modified, tested, and deployed into production by business analysts without requiring any code changes to the CICS applications, or involvement by the IT department.
- Only a single environment for running rules needs to be managed and maintained.
- The same rules and rule sets can be accessed from separate applications, providing consistency across the enterprise.
- When a rule set is refreshed, all applications that use the Rule Execution Server will be able to use the new version of the rule set from the same point in time.
- A performance cost is associated with making the call from the existing application to the Rule Execution Server.

Running the rule processing close to the existing application has the following advantages and disadvantages:

- A potential exists for better performance because no remote call needs to be made.
- Keeping the processing within the existing application environment can simplify management and security of the existing application.
- If multiple applications use the same rule sets, they might not always start using new versions of rule sets from the same point in time.
- Certain features of the Rule Execution Server might not be available. For example, when running the Rule Execution Server in a J2SE environment, hot deployment of new rules might not be possible.

Important: The original creation of the rules is done by the IT department with a product called ILOG Rule Studio. The IT department can decide which components of each rule is suitable for the business user to be able to change. This level of control is available to help ensure that changes made by the business user cannot break the IT systems.

6.2.2 Integration with existing applications running in Java EE

In this section, we look at integration with existing applications in Java Platform, Enterprise Edition (EE).

Web services

In this pattern (Figure 6-1), a centralized Rule Execution Server exposes a web service interface which the J2EE application can call to run the rules. The server can expose the web service using the web services (hosted transparent decision services) capabilities of the Rule Execution Server, or the web services could be specifically created to expose the rules.

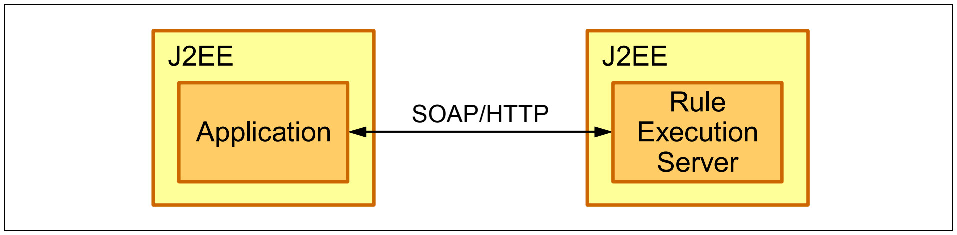


Figure 6-1 J2EE web services integration

If the J2EE application is running inside the same J2EE server as the Rules Execution Server, then integration through web services is not likely to be the best performing option. Integration through EJB calls or POJO rule sessions must be considered as an alternative. If the J2EE application is running in a separate server to the Rule Execution Server, then web services integration might be appropriate.

The web service calls use the HTTP protocol. Most J2EE servers support web services calls over HTTP. In this configuration, the full capabilities of the ILOG JRules product can be utilized by running the Rule Execution Server inside a J2EE server environment.

In addition to the advantages and disadvantages for using a centralized Rule Execution Server (6.2.1, “Advantages and disadvantages” on page 75), consider the following points also:

- Web services uses standards-based integration.
- Overhead exists for an outbound web service invocation each time the rules are run.

JRules message-driven bean

In this pattern (Figure 6-2 on page 77), a call is made to the Rules Execution Server by putting a message on a queue.

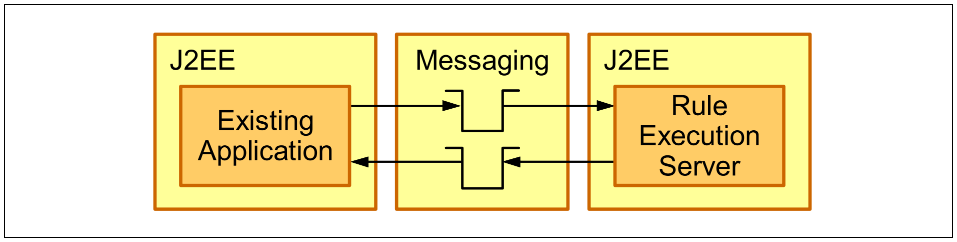


Figure 6-2 J2EE message-driven bean integration

If the J2EE application is running inside the same J2EE server as the Rules Execution Server, integration through message-driven bean may not be the best performing option. Integration through EJB calls or POJO RuleSessions should be considered as an alternative, as the cost of putting and getting a message is removed. If the J2EE application is running in a separate server to the Rule Execution Server, then integration through message-driven bean may be appropriate.

In this pattern, to call JRules, the J2EE application puts a JMS message in a queue. That message flows across a messaging infrastructure to the server hosting the Rule Execution Server, where the message is read, the rules are run and a response message is generated. The response message is then flowed back across the messaging infrastructure to the J2EE application.

This pattern might be appropriate if messaging is the standard way that applications within the enterprise are interconnected. Messaging infrastructures are typically designed to be asynchronous, so the sender and the receiver applications do not need to be available at the same time. However, if we are considering extracting rules from an existing application and moving them into a BRMS, the likelihood is that the existing application will require synchronous communications and will wait for the results of the rules before continuing. Therefore, asynchronous benefits of the messaging infrastructure do not apply.

In addition to the advantages and disadvantages for using a centralized Rule Execution Server (see 6.2.1, “Advantages and disadvantages” on page 75), also consider the following points:

- JRules MDB uses standards-based integration.
- Connections to the Rule Execution Server can probably be pooled (depending on the J2EE servers capabilities).
- Overhead exists for putting and getting a message, and flowing the messages between the J2EE application and the Rules Execution Server.
- Running the rules remotely adds cost.

JRules Enterprise JavaBeans

In this pattern (Figure 6-3), a call is made to the Rule Execution Server by making an EJB call from the existing application.

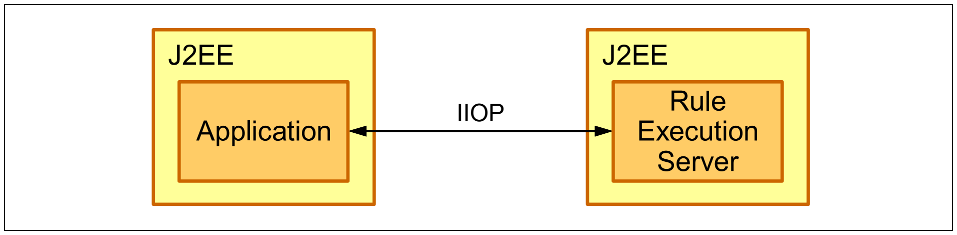


Figure 6-3 J2EE Enterprise JavaBeans integration

If the J2EE application is running inside the same J2EE server as the Rules Execution Server, then integration through a call to a local EJB is likely to be a efficient pattern for invoking the Rule Execution Server. It will have the advantages of both the centralized rule server approach and running the custom web services close to the application, without the associated disadvantages of these approaches.

If the J2EE application is running in a separate server to the Rule Execution Server, then integration through a remote EJB interface is another option that can be used to connect the two products.

In this pattern, the existing application looks up an EJB for the Rule Execution Server using a JNDI. It then fills in several Java objects that are used by the interface to the EJB, and then makes a call to the EJB. On return from the call, the existing application must map the results from the EJB call back into the Java objects that are used inside its application.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75), consider the following points:

- JRules EJB uses standards based integration.
- JRules EJB uses the same programming model to invoke the Rule Execution Server, as is used in the rest of the application.
- Connections to the Rule Execution Server can probably be pooled (depending on the J2EE servers capabilities).

POJO RuleSessions

In this pattern (Figure 6-4), a call is made to the Rule Execution Server by calling a method on a Java class from the existing application.

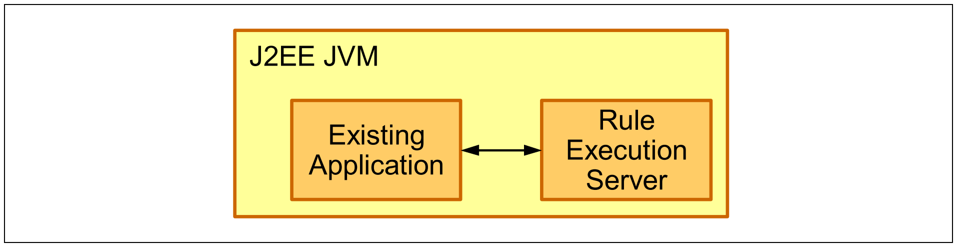


Figure 6-4   J2EE POJO RuleSessions integration

In this pattern, the J2EE application must run inside the same JVM as the Rules Execution Server, so integration is likely to be efficient. As the JVM is running inside a J2EE server, the pattern will have the advantages of both the centralized rule server approach and running the custom web services close to the application, without the associated disadvantages of these approaches.

Plain old Java objects (POJO) represent a simple interface that can be used by any Java application programming model or any of the advanced models such as SCA/SDO.

In this pattern, the existing application fills in some Java objects that are used on the interface to the Rule Execution Server. It then calls a method on a Java class, passing the Java objects, and the rules are run. On return from the call, the existing application must map the returned Java objects into the Java objects it uses internally within the application.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75), also consider the following points:

- POJO RuleSessions uses the same programming model to invoke the Rule Execution Server, as is used in the rest of the application.
- Good performance is likely, because the J2EE application and the Rule Execution Server are running in the same J2EE server JVM.

6.2.3 Integration with existing applications running in Java SE

In this section, we look at integration with existing applications that run in Java SE.

Web services

In this pattern, (Figure 6-5) a centralized Rule Execution Server exposes a web service interface that the Java SE application can call to run the rules. The server can expose the web service by using the web services (hosted transparent decision services) capabilities of the Rule Execution Server, or the web services could be specifically created to expose the rules.

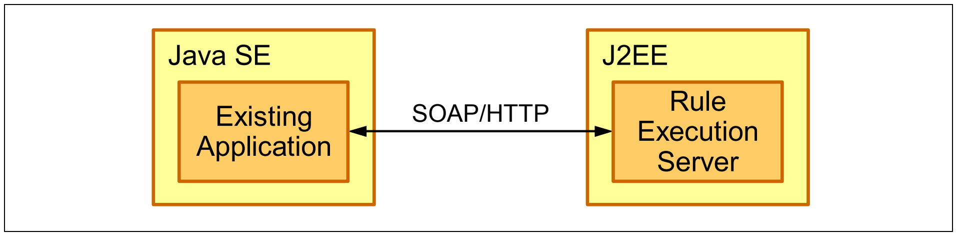


Figure 6-5 Java SE web services integration

The web service calls use the HTTP protocol. Java SE applications can make use of various java web services offerings to enable them to invoke a web service over HTTP. In this configuration, the full capabilities of the ILOG JRules product can be used by running the Rule Execution Server inside a J2EE server environment. In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, consider the following points:

- Web services uses standards based integration.
- Overhead exists for an outbound web service invocation each time the rules are run.

JRules message-driven bean

In this pattern (Figure 6-6), a call is made to the Rules Execution Server by putting a message on a queue.

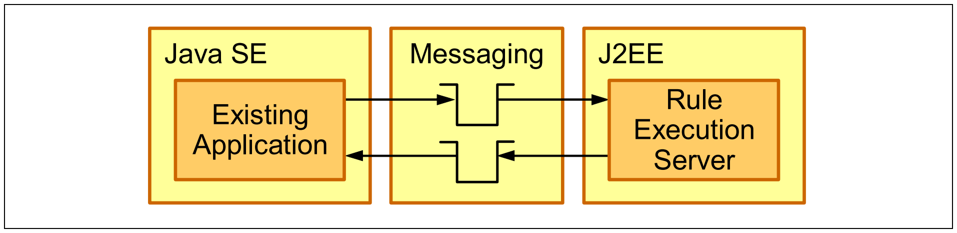


Figure 6-6 Java SE message-driven bean integration

In this pattern, to call JRules, the Java SE application puts a JMS message to a queue. That message flows across a messaging infrastructure to the server that hosts the Rule Execution Server, where the message is read, the rules are run and a response message is generated. The response message is then flowed back across the messaging infrastructure to the Java SE application.

This pattern might be appropriate if messaging is the standard way that applications within the enterprise are interconnected. Because messaging infrastructures are typically designed to be asynchronous, the sender and the receiver applications do not need to be available at the same time. However, if you are considering extracting rules from an existing application and moving them into a BRMS, the existing application will likely require synchronous communications, (that is, it will wait for the results of the rules before continuing). Therefore, the asynchronous benefits of the messaging infrastructure will not apply.

In addition to the advantages and disadvantages (see 6.2.1, "Advantages and disadvantages" on page 75) for using a centralized Rule Execution Server, also consider the following points:

- JRules MDB uses standards based integration.
- Overhead exists for putting and getting a message, and flowing the messages between the Java SE application and the Rules Execution Server.
- A JMS messaging provider is necessary and must be supported in both the Java SE environment and the J2EE environment where the Rule Execution Server is running.

JRules Enterprise JavaBeans

In this pattern (Figure 6-7), a call is made to the Rule Execution Server by making an EJB call from the existing Java SE application.

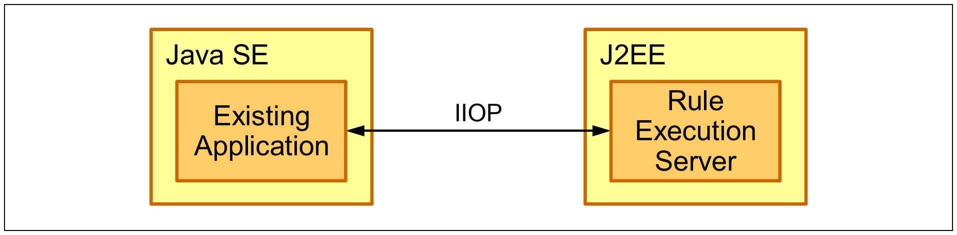


Figure 6-7 Java SE Enterprise JavaBeans integration

In this pattern, a call is made to the Rule Execution Server by making an EJB call from the existing Java SE application.

In this pattern, the existing application looks up an EJB for the Rule Execution Server by using a JNDI. It fills in some Java objects that are used by the interface to the EJB, and then makes a call to the EJB. On return from the call, the existing application must map the results from the EJB call back into the Java objects that are used inside its application.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, consider the following points:

- JRules EJB uses standards based integration.
- JRules EJB uses a Java based programming model to invoke the Rule Execution Server.
- Overhead exists for calling the remote EJB.

JRules Java SE Rule Execution Server

In this pattern (Figure 6-8), the Rule Execution Server is run inside the same JVM as the existing Java SE application, instead of inside a J2EE server.

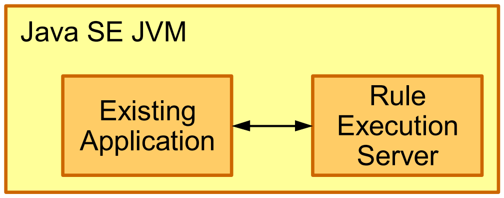


Figure 6-8 Running the Rule Execution Server in Java SE

By using this approach, the decision-making process remains within the same JVM so no remote calls are made, which can improve the performance of running the rules. The rules can be changed without having to compile, test and deploy new code, however there are some limitations when running the Rule Execution Server in a J2SE environment, which should be understood prior to choosing this pattern. See Chapter 5, “Identifying the patterns in general solutions” on page 51 for more information.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for running the rule-processing close to the existing application, consider this point also:

- When the Rule Execution Server is run inside a Java SE environment, certain features are not available by default. This includes hot deployment of new rule sets. This would mean that if a new version of a rule set was made available, it would not automatically be used. The Rule Execution Server would need to be told to reload the rule set. One way to achieve this would be to stop and start the application. When the new JVM was created, the Rule Execution Server could then load the new version of the rule set.

6.2.4 Integration with existing applications running in CICS

In this section, we look at integration with existing applications running in CICS.

Web services

In this pattern (Figure 6-9), a centralized Rule Execution Server exposes a web service interface, which the CICS application can call to run the rules. The server can expose the web service by using the web services (hosted transparent decision services) capabilities of the Rule Execution Server, or the web services can be specifically created to expose the rules.

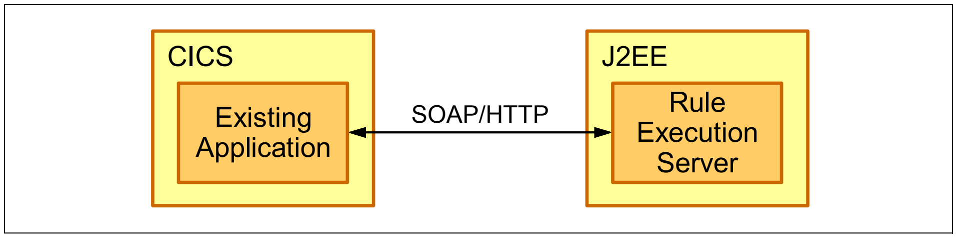


Figure 6-9 CICS web services integration

The web service calls use the HTTP protocol. CICS releases since V3.1 can invoke web services by using built-in CICS APIs. In this configuration, the full capabilities of the ILOG JRules product can be used by running the Rule Execution Server inside a J2EE server environment and calling it remotely from CICS.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, consider the following points:

- Web services uses standards based integration. The conversion from binary data to SOAP messages for the web service request is done by CICS.
- Overhead exists for an outbound web service invocation each time the rules are run.

JRules message-driven bean

In this pattern (Figure 6-10), a call is made to the Rules Execution Server by putting a message on a queue.

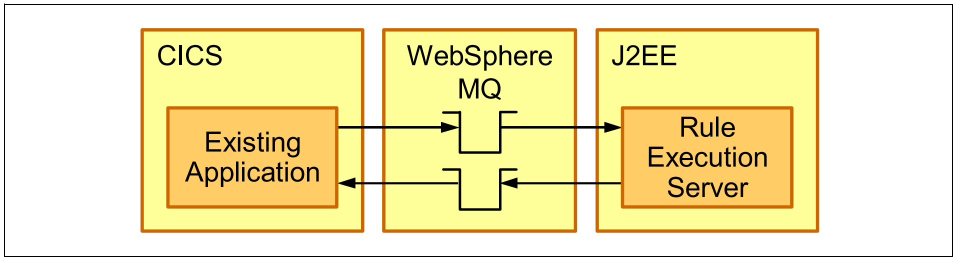


Figure 6-10 CICS message-driven bean integration using WebSphere MQ

In this pattern, to call JRules, the CICS application must render the variables into an XML format so that the JRules Rule Application can be invoked through the JMS message-driven bean interface. JRules then sends the response back in XML format, and the CICS application must be able to parse the XML to understand the response. The rendering and parsing of the XML in the CICS application requires additional code to be written in the application.

WebSphere MQ for z/OS is required for this pattern. It provides the messaging layer between CICS and the JRules Rule Execution Server.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, the following points should also be considered:

- JRules MDB uses standards based integration.
- The rendering and parsing of the XML in the CICS application requires additional code to be written in the application.
- Connections to the Rule Execution Server can be pooled.
- JRules MDB may potentially require custom code in the hosting environment.
- Overhead exists for XML marshalling and the call out over WebSphere MQ to the hosting environment to run the rules.

JRules Enterprise JavaBeans

CICS Transaction Server (CICS TS) V4.1 has support for session beans using the Enterprise JavaBeans (EJB) 1.1 specification. However, the announcement letter for CICS TS V4.1 states that this support is stabilized and that “It is IBM's intention to discontinue support for session beans in a future release of CICS TS.” Therefore, we would not recommend using EJBs as an integration pattern for CICS.

To read the full announcement letter, go to the following address:

http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&supplier=897&letternum=ENUS209-135

Rules for COBOL

In this pattern (Figure 6-11), COBOL code can be generated from rules, and used within an existing CICS TS application.

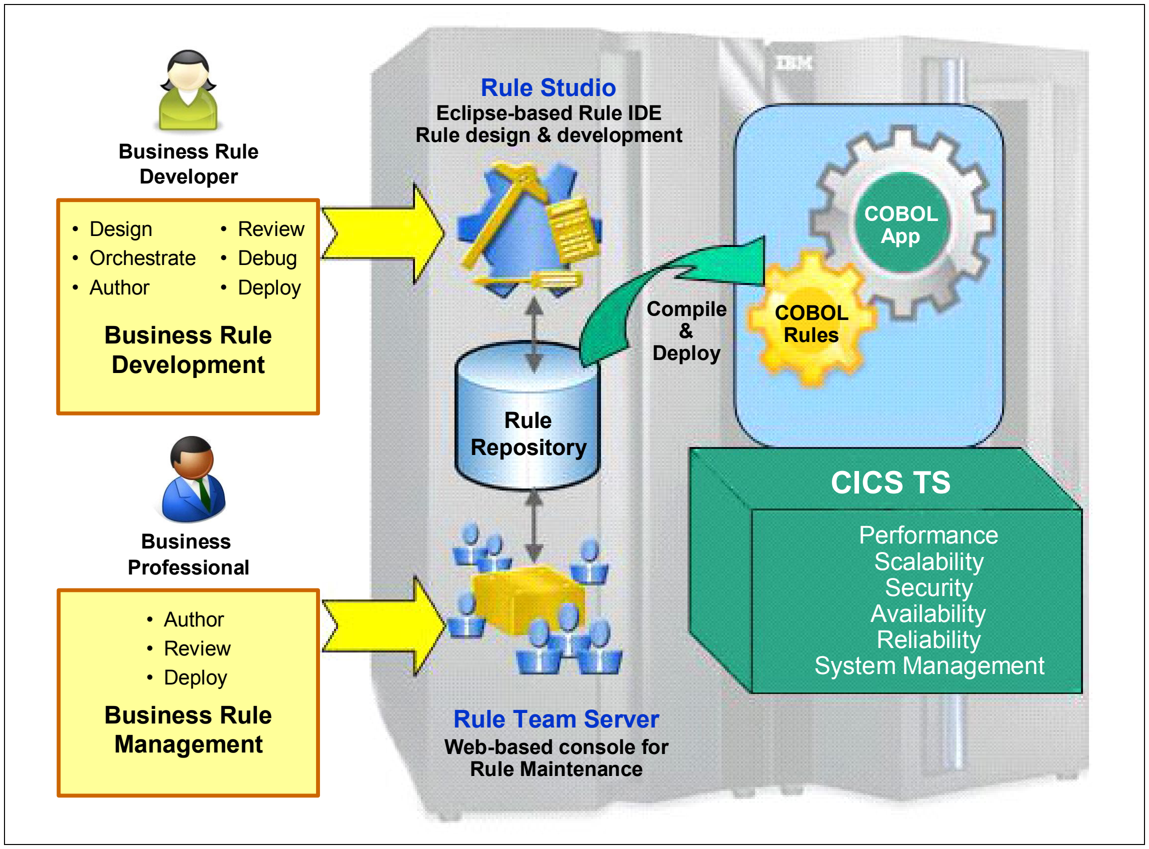


Figure 6-11 Using Rules for COBOL with CICS TS

The ILOG Rules for COBOL product provides a rules solution for users whose biggest concern is performance. Starting from a COBOL copybook, rules can be created and made available for the business analyst to work with. When the business analyst modifies the rules, COBOL code is generated that contains the new rules. This code then must be compiled and run through the deployment cycle to get the new code into production.

The advantages and disadvantages of this approach are as follows:

- The existing application is calling a COBOL routine to check the rules, so from a performance perspective the rule check will be fast.
- The business analyst can make the changes to the rules, so the risk of those changes being misinterpreted/incorrectly implemented by the IT department is removed.
- Rule changes result in the generation of COBOL source code, which means that the IT department still have to take that code and put it into production. Therefore, the values of hot deployment and being able to dynamically make changes to the rules without involving the IT department are lost.
- Rules for COBOL provides a subset of the JRules capabilities. This is partly as not all JRules features map nicely into COBOL code. Therefore, if the same rules need to be shared between Rules for COBOL and JRules, care needs to be taken over which features are used.
- The generation of the rules has to start from a COBOL copybook describing the layout of the data. If the same rules need to be shared between Rules for COBOL and JRules, the best approach is to start from the COBOL copybook when generating the data that the rules will work with.

JRules Java SE Rule Execution Server

In this pattern (Figure 6-12), the Rule Execution Server is run in a JVM that is managed by CICS TS, instead of in a J2EE server.

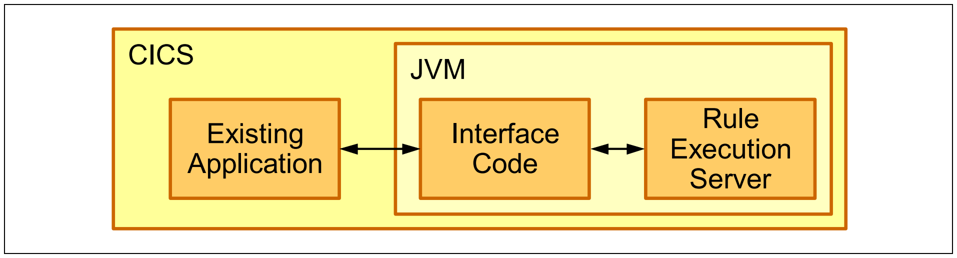


Figure 6-12 Running the Rule Execution Server inside a CICS managed JVM

As described in Chapter 5, “Identifying the patterns in general solutions” on page 51, the Rule Execution Server can be run in a Java SE environment, which CICS TS Java support provides. By using this approach, the decision-making process remains within CICS TS, same as the ILOG Rules for COBOL solution.

The advantage over the ILOG Rules for COBOL solution is that the rules can be changed without having to compile, test, and deploy new code. However, if the existing CICS application is not written in Java, a performance overhead exists when compared to the ILOG Rules for COBOL solution, because a switch is necessary, from the non-Java program to a Java program that is running inside a CICS TS JVM, to run the rules.

CICS provides a tightly controlled Java environment. If the Rule Execution Server does run in it, consider the following points:

- CICS Java applications can call the Rule Execution Server directly, passing Java objects containing the data to be used. Non-Java CICS applications need to determine the best approach to translate their data structures into a format that is suitable for use inside the Rule Execution Server (Java objects, or XML).
- On first use of a rule set, the rules must be loaded and parsed. For performance, ensure that this parsing is not needed for every request. Making the JVMs reusable and maintaining a static reference to the Rule Execution Server must ensure that the rule set can be cached and used for more than one request.
- CICS allows only one request at a time to execute in a JVM. Therefore, for this solution to scale you need a number of JVMs, which might need to be spread across CICS regions, depending on the size of each JVM. The maximum number of JVMs that can be run in a single CICS region is approximately 17, but is less if they need a lot of storage.
- When the Rule Execution Server is run inside a Java SE environment, certain features are not available by default, including hot deployment of new rule sets. For this reason, if a new version of a rule set is made available, it is not automatically be used. The Rule Execution Server must be directed to reload the rule set. One way to achieve this information is to purge the JVM pool. When new JVMs are created, the Rule Execution Server can then load the new version of the rule set.

6.2.5 Integration with existing applications running in IMS

In this section, we look at integration with existing applications running in IMS.

Web services

In this pattern (Figure 6-13), a centralized Rule Execution Server exposes a web service interface, which the IMS application can call to run the rules. The server can expose the web service using the web services (hosted transparent decision services) capabilities of the Rule Execution Server, or the web services can be specifically created to expose the rules.

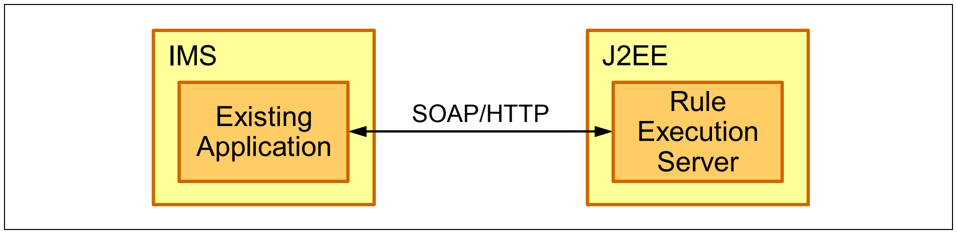


Figure 6-13 IMS web services integration

The web service calls use the HTTP protocol. An IMS offering named “IMS Enterprise Suite SOAP Gateway” enables IMS applications to make outbound web services calls. In this configuration, the full capabilities of the ILOG JRules product can be used by running the Rule Execution Server inside a J2EE server environment.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, consider the following points:

- Web services uses standards-based integration. The conversion from binary data to SOAP messages for the web service request is done by CICS.
- Overhead exists for an outbound web service invocation each time the rules are run.

JRules message-driven bean

In this pattern (Figure 6-14), a call is made to the Rules Execution Server by putting a message on a queue.

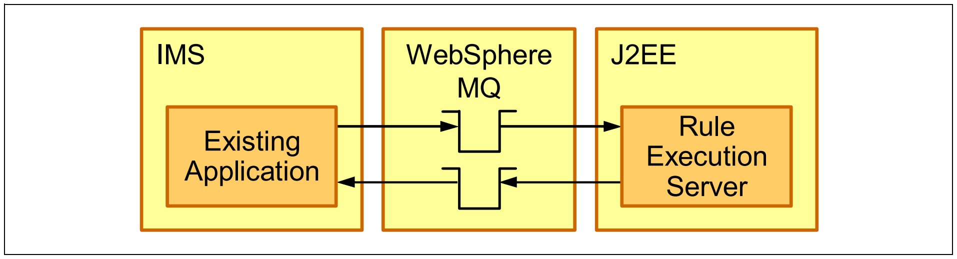


Figure 6-14 IMS message-driven bean integration using WebSphere MQ

IMS application programs that run in dependent regions of IMS Version 10 and IMS Version 11 can send outbound messages to request services or data, such as from a message-driven bean and receive responses in the same transaction during the same unit of work through IMS Connect and Open Transaction Manager Access (OTMA). The request for services or data outside the IMS installation is called a *callout request*.

You can issue a callout request from an IMS application and synchronously receive the response by using the IMS TM resource adapter. IMS TM Resource Adapter Version 10.3 is required.

In this pattern, to call JRules, the IMS application must render the variables into an XML format so that the JRules Rule Application can be invoked through the JMS message-driven bean interface. JRules then sends the response back in XML format, and the IMS application must be able to parse the XML to understand the response. The rendering and parsing of the XML in the IMS application requires additional application code to be written.

WebSphere MQ for z/OS is required for this pattern. It provides the messaging layer between IMS and the JRules Rule Application.

In addition to the advantages and disadvantages (see 6.2.1, “Advantages and disadvantages” on page 75) for using a centralized Rule Execution Server, consider the following points:

- JRules MDB uses standards-based integration.
- The rendering and parsing of the XML in the IMS application requires additional application code to be written.
- Connections to the Rule Execution Server can be pooled.

- JRules MDB may potentially require custom code in the hosting environment.
- Overhead exists for XML marshalling and the call out over WebSphere MQ to the hosting environment to run the rules.

JRules Enterprise JavaBeans

In this pattern, a call is made to the Rule Execution Server by making an EJB call from the existing IMS application.

Both local and remote EJB calls are supported by the Rule Execution Server. In this pattern, the calls are remote from the IMS Java environment to the J2EE Java environment in which the Rule Execution Server is executing.

IMS application programs running in dependent regions of IMS Version 10 and IMS Version 11 can send outbound messages to request services or data, such as from an EJB component, and receive responses in the same transaction during the same unit of work through IMS Connect and OTMA. The request for services or data outside of the IMS installation is called a callout request.

You can issue a callout request from an IMS application and synchronously receive the response by using the IMS TM resource adapter. IMS TM Resource Adapter Version 10.3 is required.

In this pattern, to call JRules, the IMS application must render the variables into a format suitable for use in Java, in the EJB. The EJB is then invoked. On return from the EJB, the returned data must be converted into a format that is suitable for the IMS application to understand.

In addition to the advantages and disadvantages discussed in Section 6.2, “Choosing the best technology pattern for the application environment” on page 74 for using a centralized Rule Execution Server, consider the following points:

- JRules EJB uses standards based integration.
- Overhead exists for the remote EJB call each time the rules are checked.
- Overhead exists for converting the IMS application data to or from a format suitable for use on the EJB call.

Rules for COBOL

In this pattern, COBOL code can be generated from rules, and used within an existing IMS application.

The ILOG Rules for COBOL product provides a rules solution that addresses performance issues. Starting from a COBOL copybook, rules can be created and made available for the business analyst to work with. When the business analyst modifies the rules, COBOL code is generated, containing the new rules. This code must then be compiled and run through the deployment cycle to get the new code into production.

The advantages and disadvantages of this approach are as follows:

- The existing application is calling a COBOL routine to check the rules. Therefore from a performance perspective, the rule check is fast.
- The business analyst can make the changes to the rules. Therefore the risk of those changes being misinterpreted or incorrectly implemented by the IT department is removed.
- Rule changes result in the generation of COBOL source code, which means that the IT department still must take that code and put it into production. Therefore, the values of hot deployment and being able to dynamically make changes to the rules without involving the IT department are lost.
- Rules for COBOL provides a subset of the JRules capabilities. This is partly because not all JRules features map nicely into COBOL code. Therefore, if the same rules need to be shared between Rules for COBOL and JRules, care needs to be taken over which features are used.

The generation of the rules must start from a COBOL copybook describing the layout of the data. If the same rules must be shared between Rules for COBOL and JRules, the best approach is to start from the COBOL copybook when generating the data that the rules will work with.

6.2.6 Integration with existing applications running in batch on z/OS

When running batch jobs on z/OS, for example jobs that run overnight, the patterns for integrating with a BRMS are limited by the needs of the application and the capabilities of the environment. Batch applications often need to complete by specific time deadlines. These deadlines can be imposed by regulations, or perhaps online services have been taken offline to enable the batch applications to have exclusive access to the resources they need, and therefore the batch applications must end by a certain time to enable the online applications to be available again when needed. Although developing a batch application that makes remote calls to a centralized Rule Execution Server might be possible, we do not see this as a common pattern because of the overhead of making the calls. The most common pattern we see for batch processing is using Rules for COBOL.

Rules for COBOL

The ILOG Rules for COBOL product provides a rules solution for users whose biggest concern is performance. Starting from a COBOL copybook, rules can be created and made available for the business analyst to work with. When the business analyst modifies the rules, COBOL code is generated, containing the new rules. This code must then be compiled and run through the deployment cycle to get the new code into production.

The advantages and disadvantages of this approach are as follows:

- The existing application is calling a COBOL routine to check the rules. Therefore from a performance perspective, the rule check is fast.
- The business analyst can make the changes to the rules. Therefore the risk of those changes that are implemented by the IT department being misinterpreted incorrectly is removed.
- Rule changes result in the generation of COBOL source code, which means that the IT department still must take that code and put it into production. Therefore, the values of hot deployment and being able to dynamically make changes to the rules without involving the IT department are lost.
- Rules for COBOL provides a subset of the JRules capabilities. This is partly because not all JRules features map nicely into COBOL code. Therefore, if the same rules need to be shared between Rules for COBOL and JRules, care needs to be taken over which features are used.

The generation of the rules must start from a COBOL copybook describing the layout of the data. If the same rules must be shared between Rules for COBOL and JRules, the best approach is to start from the COBOL copybook when generating the data that the rules will work with.

7

Business processes

This chapter contains the following topics:

- Business processes and rule management
- Decision patterns with processes
- Considerations for BRMS with BPM solutions
- Integration with IBM BPM solutions

7.1 Business processes and rule management

In this section, we look at business processes and rule management.

7.1.1 Introduction

In our example insurance scenario, the process has a sequence of activities for which four of them have decisions:

- Validation

  This activity is a request for a quote.

- Entitlement

  This activity checks that the requestor represents an acceptable risk before providing a quote for a particular product.

- Quote fulfillment

  This activity routes the quote request to the appropriate application to apply the underwriting rules and create a basic price quote and corresponding conditions.

- Price discounting

  This activity adjusts the price passed back to the customer to account for marketing promotions or other risks that cannot be taken into account by the product application.

In our scenario, the insurance company requires the ability to make regular changes to these rules that form the decisions, change the process flow, influence how human tasks are assigned and routed to optimize the process performance.

A business process can take many forms.

- It can be implemented as a legacy industry application, which has a process flow, business logic, business rules and decisions that are tightly coupled, limiting the ability to change the flow or rule logic without a complete redeployment.
- It can be a more modular J2EE application where the flow of execution and business logic and decisions are separate in the code, but any changes to a component still requires the complete application to be redeployed.
- It can be implemented as a loosely coupled process orchestration layer that invokes loosely coupled business services and decision services, which we now refer to as *business process management* (BPM) and which is what we

look at further in more detail. One major benefit is independent components, which can be updated to enable changes to be made more easily.

You should understand what a process is and its core constituents before we fully identify why a BRMS solution complements a BPM solution.

A *business process* is a collection of related, structured activities or tasks that provide a specific solution internally to a business or to external clients. In the context of the solution being described here, we are referring to business process in the context of an operational implementation, which has a specific starting point, a flow of activities across multiple applications or services, and one or more process endpoints that depend on the branches and decisions taken within the process.

Certain specific departmental business processes are narrow in scope and can be fulfilled by one solution provider that provides the process flow, business logic, and decisions for the entire solution to fulfill the business need.

Other processes span organizational functions within a line of business or across the enterprise. The activity flow integrates the organizational functions in the appropriate sequences and invokes business services and business decisions in each of the organizational areas that are required to implement the process.

When decomposing any process flow, consider the following questions:

- What is the flow of activities to implement the process?
- Which activities can invoke automated business services?
- Which activities require assisted services, with people being required for informational review, approval, or offline tasks?
- Which activities are decision points in the process, where the result of the decision is taken based on some business rules analyzing the data associated in the process instance? The result normally requires the process to handle the subsequent process activity flow differently in a separate activity branch.

7.1.2 Business process management

Business process management solutions can handle a loosely coupled solution: orchestrating business services and decision services.

Business processes can be modeled in many ways and through many tools, from simple flow diagrams to using an industry standard such as the Business Process Modeling Notation (BPMN), which captures the process workflow and activity sequences in a consistent manner that visualizes the process as a flowchart with a sequence of activities.

For processes that have process-flow branching rules and simple business rules, a BPM solution on its own is a good solution.

However, BPMN is not a standard designed to capture, express, and manage business rules in a natural language to aid business and IT collaboration. for certain business requirements, a BPM solution alone is not the optimal way to implement the solution when many business rules or complex rules and decisions exist.

Based on the four questions in 7.1.1, “Introduction” on page 96, for which the process is responsible, the BPM solution must be able to handle certain tasks:

- Be able to orchestrate the activities.
- Invoke the application and decision services.
- Manage the human task activities in the process flow.

Having a separate solution that captures, implements, and manages the business decisions and rules separately to complement the BPM solution is an enhanced solution scenario pattern that can optimize the process in respect to business agility and process streamlining. A business rule management system (BRMS), such as JRules, can fulfill these capabilities to complement the BPM solution.

BPM orchestrates and improves business processes:

- Is focused on flow orchestration.
- Has human orientation for activities within process.
- Crosses system and organizational boundaries.
- Has process-oriented transparency, driving awareness and improvement of business processes to an increased set or organizational stakeholders
- Has long and short running processes.

BRMS expresses and automates business decisions:

- Is focused on data orientation and what business information is used to make a decision.
- Encapsulates to a single boundary of a decision.
- Promotes decision and business rule re-use across BPM and other applications.
- Increases visibility of decisions driving critical business applications and processes.

Figure 7-1 shows the activity steps and decision points that are associated with a process.

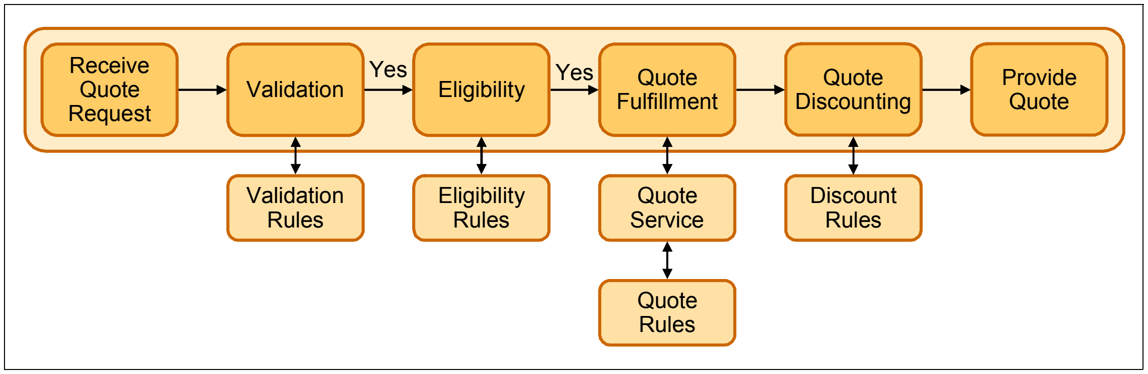


Figure 7-1 Insurance process activity steps and decision points

A process is flow-oriented and determines in what order the activities are run and what provider implements the rules. Business rules are data-oriented for making decisions when specific combinations of data are present within the current context for the solution.

Example 7-1 shows a business validation rule.

Example 7-1 Authored business validation rules

```
definitions
   set ‘name’ to the first name of the driver ;
if
   any of the following conditions is true :
      - the age of the driver is less than 16
      - the age of the driver is more than 100
then
   add validation error: “The driver’s age (“ + name + “) is beyond the
coverage range” ;
```

Using a combined BPM and BRMS solution also improves straight-through processing by automating certain manual tasks and altering the process activity flow to use automated application services, or business decisions with rules.

7.1.3 Solution patterns using BPM and BRMS

This section covers the multiple solution integration patterns for using a combined BPM and BRMS solution.

Be sure of the following starting points when you adopt a combined BPM and BRMS solution:

- No BPM or BRMS solution has been previously used.
- BRMS with business rules is being used, but not in combination with a process implemented in BPM.
- BPM is being used with some decisions that are implemented in BPM, but BRMS has not been used before.
- BPM and BRMS have both been used before, but not together.

For the scenario in this book, the assumption is that BPM was already in use and the quote fulfillment calculation was provided by a separate application. As shown in Figure 7-2, we have a process that is designed by the process owner and business analyst and is implemented by the process developer and application developer. Any rules that are implemented within the process flow in BPM or the application are implemented by these developers from the business requirements. The business teams do not have direct access to view or change the implementation of the rules directly.

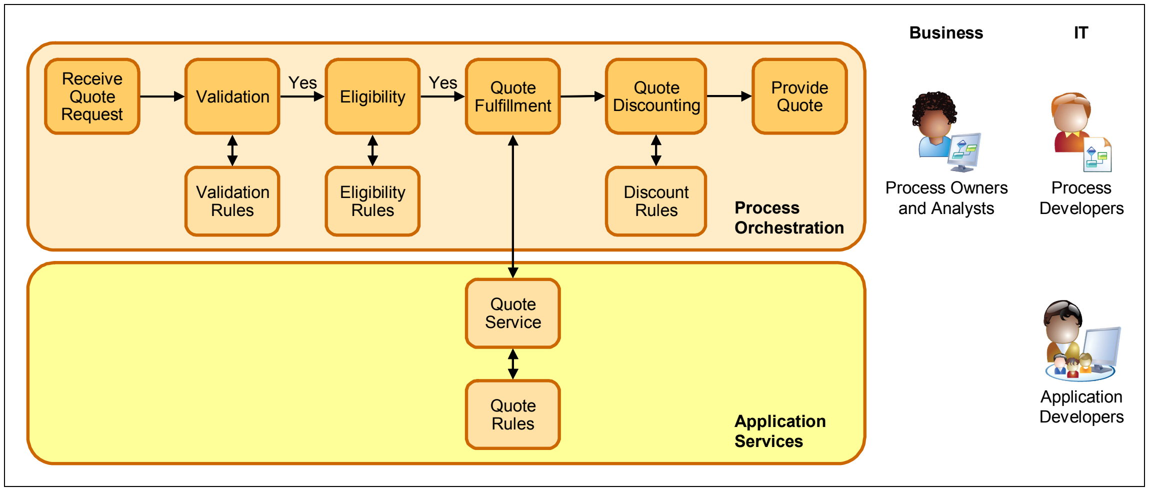


Figure 7-2 Starting point: Process and application have tightly coupled rules

The best practice is to identify the high-level decision points and make them visible in the process orchestration flow. This tactic enables the process model to be easily understandable and the main decisions points clearly visible. It also enables key performance indicator (KPI) process monitoring, if required, at each decision point to monitor process performance (for example, to calculate the percentage of valid insurance applications and eligible insurance applications compared to all Insurance applications).

BRMS contains a capability called *rule flow*, which is a method of sequencing of decision components within more complex decisions. Using the BPM process orchestration appropriately with the rule flow capabilities is key for an optimal design. See the detailed considerations in 7.3.3, "Assembly considerations: Using JRules rule flow with process orchestration" on page 125.

Adopting BRMS to add value to an existing BPM solution can be done incrementally, by gradually extending the capability used by incrementally adopting and implementing the patterns or selecting which best meets the current needs and implementing that pattern.

The primary solution patterns, which are described in this section, are as follows:

- IT uses BRMS to author and run rules to complement BPM
- BRMS for improving business collaboration and agility
- Central authoring and management of business rules

IT uses BRMS to author and run rules to complement BPM

Depending on the various characteristics around each of the process decision points, various decisions and rules can gain several immediate benefits from an IT-focused BRMS solution. Figure 7-3 illustrates a decision path that is implemented in BRMS.

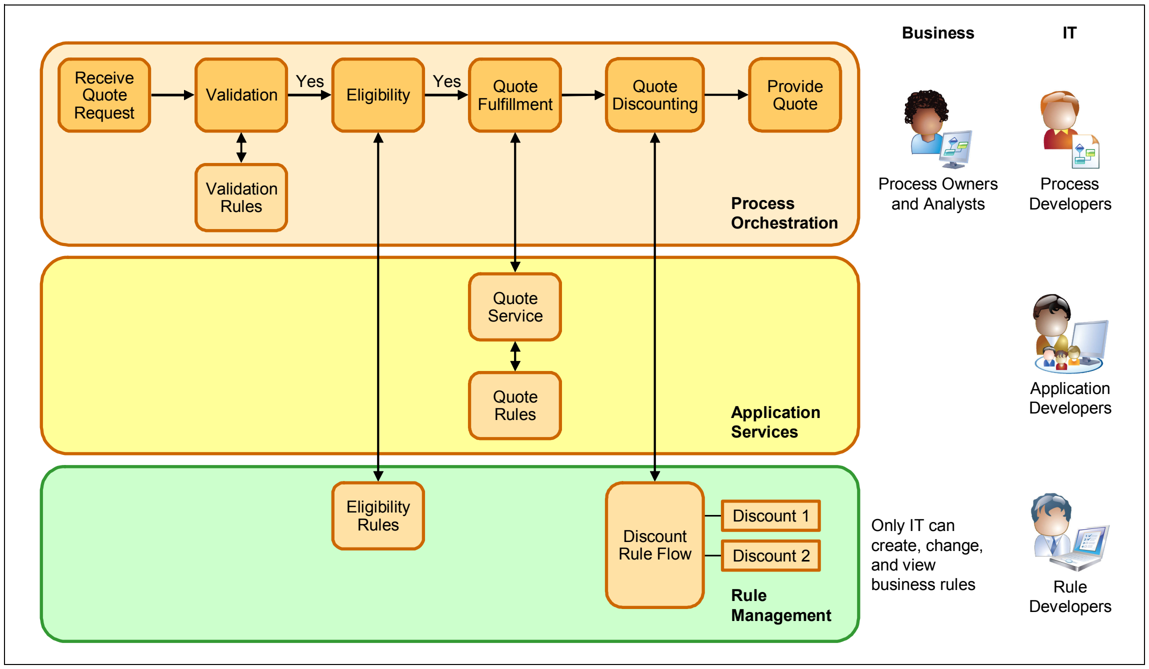


Figure 7-3 IT has selected certain decisions to be implemented in BRMS

This pattern is often used when some of the following requirements are identified for IT to enhance the process performance and agility. The two major differences in the pattern is that two of the decisions with rules that were previously implemented by the process developer in the BPM tools are now authored and deployed by a Rule Developer. The role focuses on creating, changing, and maintaining business rules to deploy to the BRMS solution.

In this pattern, WebSphere ILOG JRules, which includes the JRules Rule Studio tooling and JRules Rule Execution Server, is typically used.

Requirements for this pattern are as follows:

- Authoring, deploying, and managing medium to complex-level decisions for a process

  A business process reaches a decision point where it needs to make a medium to complex level of decision. These rules cannot be captured easily in a few simple sequential rules in the BPM products, such as entitlement, quote fulfillment calculation, or pricing in other examples.

- Where business decisions are complex and more specialized, a simulation and test environment is required to provide productive regression testing.

  The “what if” scenarios to understand the impact of changes of individual business rules on the overall decision result prior to the change that BPM-only solutions do not provide.

- A medium to complex decision service can reach a point where additional data is required.

  An example is changing the process flow so that an initial decision can be invoked, followed by an information retrieval activity, and then by the final decision activity.

- Changes to business rules more frequently than the process flow is normally changed and redeployed

  This requirement needs a separate life cycle to change the business rules and decision behavior, independently of the process flow to meet business needs.

- Improved streamlining of business processes and providing more consistent decisions by automating some of the decisions.

  For example, certain human tasks within a process can be automated by capturing certain business rules that were previously documented on paper and using them in a programmatic manner.

BRMS for improving business collaboration and agility

As an extended improvement step to the first pattern that is introduced in this chapter, additional value can be realized by focusing on business and IT collaboration and by changing various internal organizational responsibilities, thus creating greater agility and collaboration.

Figure 7-4 shows a pattern that enables policy managers and analysts to view or change rules.

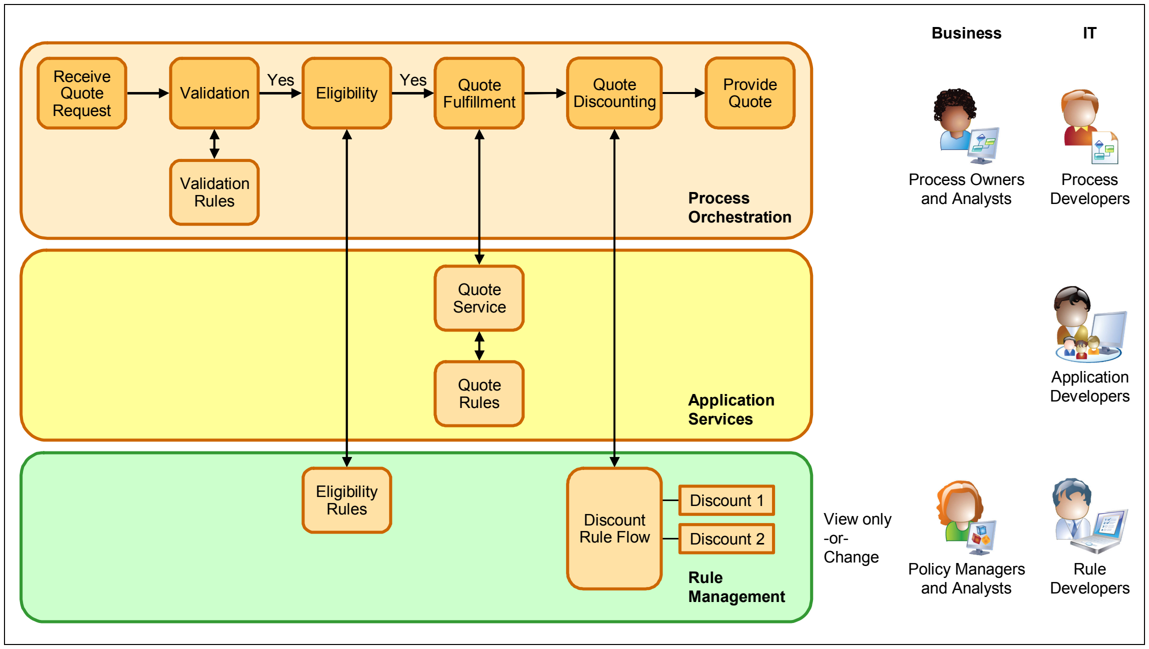


Figure 7-4 Enable policy managers and analysts to view or change rules

This pattern is often used when business rules and constraints frequently change and require updating. This updating can be a large part of ongoing IT maintenance for business processes, which are often simple updates and have to wait for IT resources to become available. Certain updates can be performed by business teams, allowing IT resources to focus on larger initiatives to improve business performance.

Tools

In this pattern, WebSphere ILOG JRules, which includes the JRules Rule Studio tooling and JRules Rule Execution Server, is typically used. (This pattern is introduced in Chapter 2, “Business rule management system (BRMS)” on page 13.)

Additional BRMS business tooling is also used:

- WebSphere ILOG Rule Server for Business collaboration to view and change business rules in a central repository.
- Add-on solutions to the Rule Team Server, including the following solutions:
  - WebSphere ILOG Rule Solutions for Office when business users used Microsoft Office 2007 for capturing business rules.
  - WebSphere ILOG Decision Validation Services for simulation, test and “what if” scenarios.

Requirements

Requirements for this pattern are as follows:

- More visible, understandable business rules across business and IT.

  Instead of the business rules being hidden in application code that is accessible only by the IT team, business rules are expressed in common semantics that are agreed upon and understood by both business and IT teams. This approach enables the business team to view the current business rules and clearly specify rule changes.
- Business team to change aspects of the business rules directly as needed and issue deployment requests to the IT team
- Complex simulation, test, and “what if” scenarios

  When business decisions are complex, more specialized simulation and test environments are required to provide productive regression testing. The “what if” scenarios are run to understand the impact of changes of individual business rules on the overall decision result prior to the change being implemented. BPM-only solutions do not provide that capability.

Central authoring and management of business rules

This pattern is for experienced users of BRMS when they do multiple projects across applications that use BRMS and BPM with BRMS.

The pattern demonstrates BRMS as an independent solution component and not simply an extension of a BPM-only solution. It has the ability to author and implement rules for both the process in BPM and the rules that are required for the quote fulfillment application, as shown in Figure 7-5 on page 105. This is an advantage of BPM and BRMS compared to BPM-only solutions.

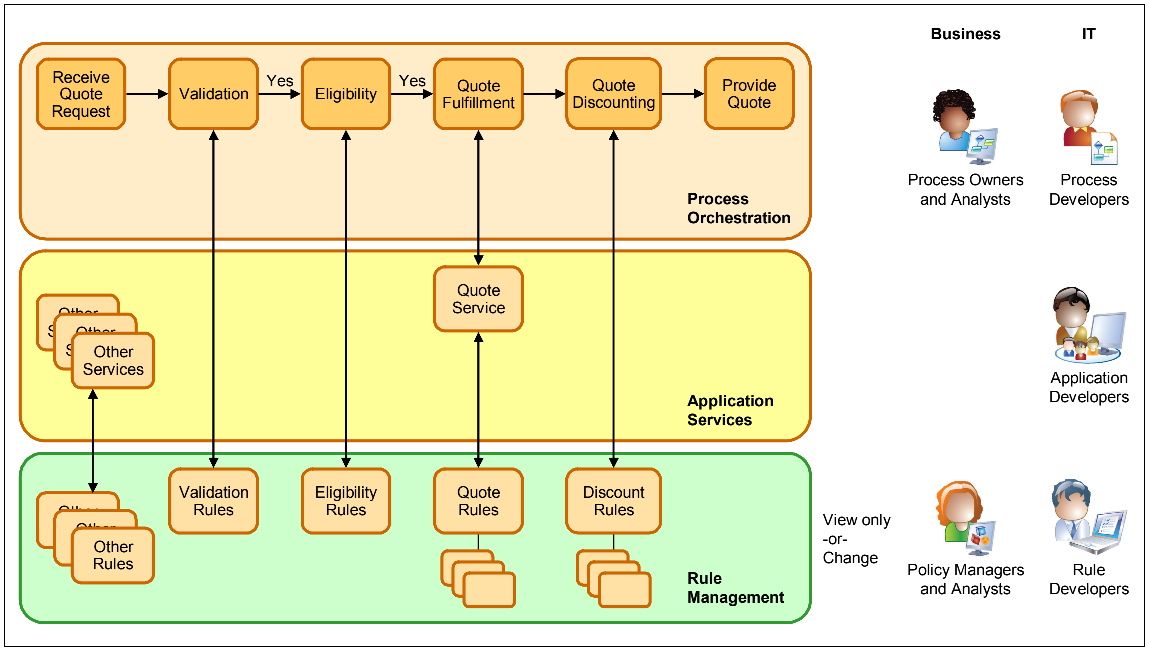


Figure 7-5 Centralize rule management pattern across BPM and applications

Using common business vocabularies and consistent usage of the BRMS tooling for authoring rules and deploying them, and having the ability to support central governance and change policies enables:

- Rule reuse across projects
- Reduction in the number of rules solutions
- Reduced IT costs and improved agility

The BRMS components in this pattern (the pattern described in Chapter 2, “Business rule management system (BRMS)” on page 13) are the same used in the second pattern, as listed in “Tools” on page 103. They are simply used extensively across the enterprise.

Requirements for this pattern are as follows:

- Process compliance and auditability

  Business decisions and rules in certain processes have to be auditable and governed tightly, which means that separating them and managing them separately from the process instances is a better approach for the business need.

- Reusing decisions or business rules across various processes or shared with applications across the enterprise

  Business decisions and rules must be defined as reusable business artifacts, where productivity gains can be obtained by sharing decisions and business rules across a line of business or enterprise. A good example is allowing multichannel solutions to require access to a common and consistent set of decisions and business rules.

7.1.4 Benefits of integrating BPM and BRMS

Using either BPM or BRMS as a stand-alone solution provides benefits to business solutions. However, when used together, the solution inherits the capabilities and advantages of both solution types, optimizing the benefits. As a result, all business solutions benefit from enhanced business agility, process streamlining, and performance benefits, leading to improved business results.

Improved business agility, and business and IT collaboration

By using a BPM solution, the ability to implement and deploy new processes and quickly change existing processes results in an enhanced time-to-value investment.

Combining BRMS with BPM enables even more productive and faster change; business rules can be changed more frequently and independently of the process flow logic. This combination further optimizes time-to-value of the solution.

Another aspect that increases business agility is when BRMS enables the business team to modify business rules after the rules are implemented by IT. This process circumvents the iterative cycles between business and IT to make minor changes to business rules; the business team can take responsibility for these changes with IT agreement.

As a result, some businesses see benefits in the shortened amount of time to agree and make changes, from months to days and even hours in some cases.

Improved process performance and streamlining

With a BPM solution, processes can have improved process flow of human tasks, and automated activities to improve process performance.

Combining BRMS with BPM enables automation of human tasks in the process, which improves performance. The combination can also simplify and streamline the decision logic within the process model by creating a separate automated decision in BRMS. This way has a contributory effect in reduced end-to-end process execution times and increases the percentage of processes that can be handled with straight-through processing.

Risk, compliance, and audit

This section describes reducing risk, guaranteeing regulatory compliance, and decision traceability and auditing.

Reduced operational risk

Externalizing business rules from the process reduces operational risk as the decision service can be standardized and optimized according to best practices, business policy, legislation, and other compliance regulations. If all channels in a multichannel operation use a centralized decision service, the decision outcome will be consistent and repeatable.

Guarantee regulatory compliance

Currently, much emphasis exists about meeting legislation and regulatory compliances for doing business. To avoid legal and financial consequences, businesses must ensure that these rules are followed. Regulations change over time and new regulations are introduced. Organizations must implement these changes quickly and roll out the changes to all affected departments. Externalizing all of these business rules from the process in a central BRMS repository offers a huge business benefit.

Decision traceability and auditing

When processes have a requirement to store auditable records for decisions that are made for specific transactional agreements, having access to the details about which rules were used with a decision to make the decision can be a major advantage. An example might be a banking process that asks why a loan was provided to a client who has defaulted and which policies and rules were used to reduce the risk and comply with regulations to make that decision.

Reduced IT cost

With an improved time-to-value on implementing and changing processes, the IT resource that is required to make the changes is reduced, which reduces cost. This approach offsets the cost of any additional software licensing and hardware needed for automating decisions through the reduction in time and effort of manual decision processes.

Improved business results

Improved processes and ability to respond to changing market needs at lower cost can result in additional business opportunities. An examples includes processes that affect core external business operations such as an order to cash, and insurance quotes.

7.1.5 Solution life cycles

When BPM and BRM solutions are used together as a combined platform to implement processes, you can expand the roles that are required for the joint solution. This way allows the components of the solution to have their own life cycles and rates of change that vary to align with required business needs.

In many business processes, the rate of change of business rules within a process changes far more frequently than the process activity flow. Another major factor to consider is roles within an organization. The role that is responsible for owning and implementing the processes normally differs from the role that defines and governs the business rules and policies.

Figure 7-6 illustrates the roles in a joint BPM and BRMS solution.

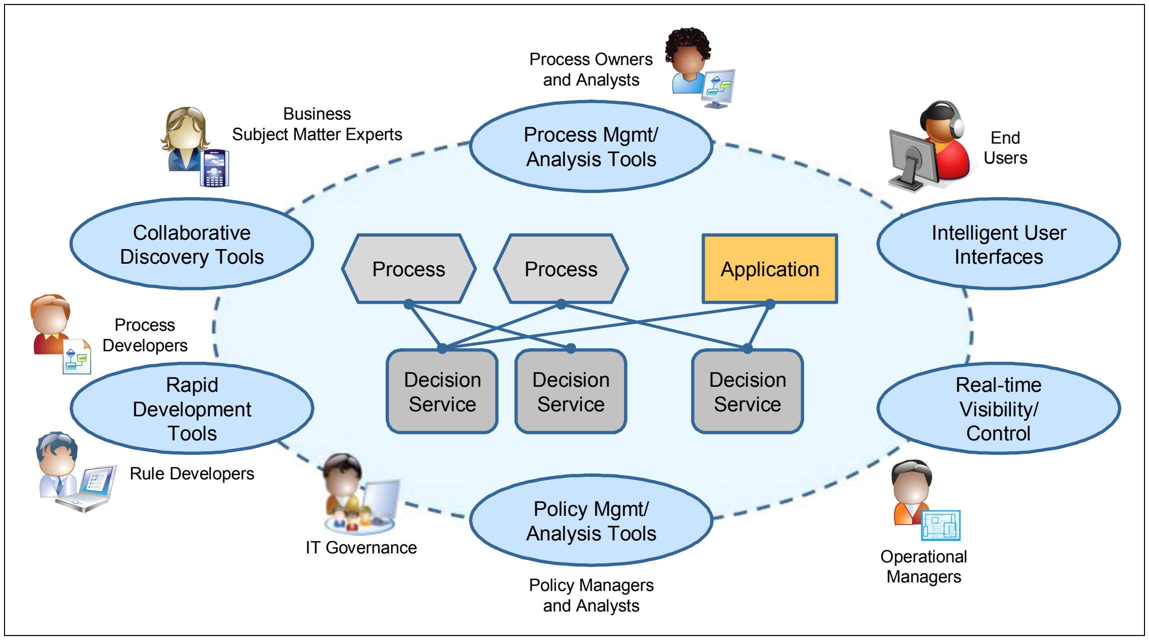

Process Owners and Analysts
Process Mgmt/ Analysis Tools
Business Subject Matter Experts
End Users
Collaborative Discovery Tools
Process
Process
Application
Intelligent User Interfaces
Process Developers
Decision Service
Decision Service
Decision Service
Rapid Development Tools
Real-time Visibility/ Control
Rule Developers
IT Governance
Policy Mgmt/ Analysis Tools
Operational Managers
Policy Managers and Analysts

Figure 7-6 Roles for joint BPM and BRMS solution

In small projects, the same person may model and assemble the process flow and also define simple rules for the process context.

When the business focuses on capturing its enterprise business policies and rules and assesses which ones are applicable to major decisions (such as pricing, quotes, and eligibility) the organization doing this will differ from the process owners, as shown in Figure 7-7 on page 110.

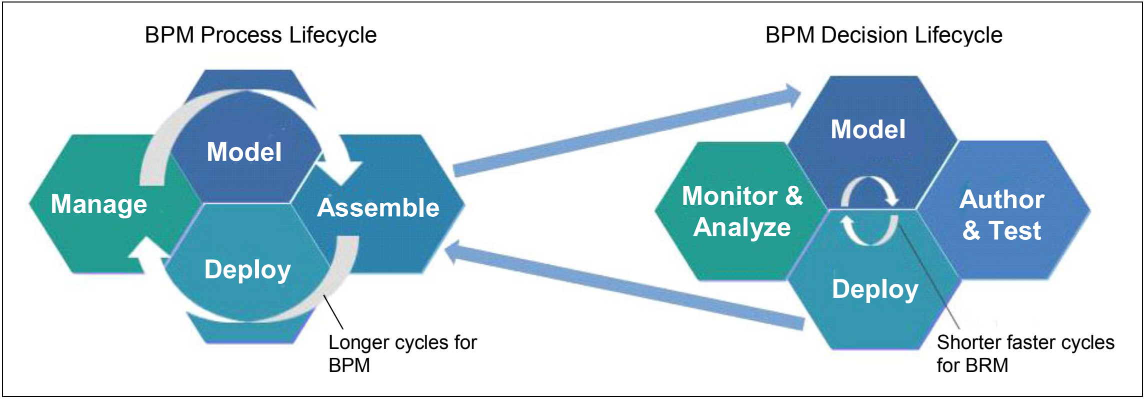


Figure 7-7 Separate BPM and business rule management life cycles

The left side of the figure shows a normal four-stage SOA development service life cycle, which can be used for processes. The right side shows a separate business rule and decision life cycle.

For many processes, the rate of change of the business rules is more frequent to meet new business requirements than the less frequent changes to process flow logic and activities.

Existing processes and decisions can be changed independently with a final regression test to ensure that the complete solution meets its business requirements.

For new processes, a normal expectation is that the processes are modeled first, so that the decision points in the process flow can be identified more easily.

At the process assembly phase, real services are required to implement the modeled process activities. This phase is when the BRMS solution needs to provide an appropriate decision service to meet the process needs.

A decision service can be reused if already available or modeled where the rules are authored, tested. and combined into the decision service, which can be immediately deployed. When complete, the decision service can be assembled with all other services that the process will use to orchestrate its activity steps.

7.2 Decision patterns with processes

Within the process activity flow, certain decision points must be identified: where the result of a decision might influence what action or branch the process flow undertakes. In examining the decision patterns identified in Chapter 6, “Existing applications” on page 71, we assess which patterns are applicable for the combination of BPM and BRMS.

Figure 7-8 shows the general decision pattern.

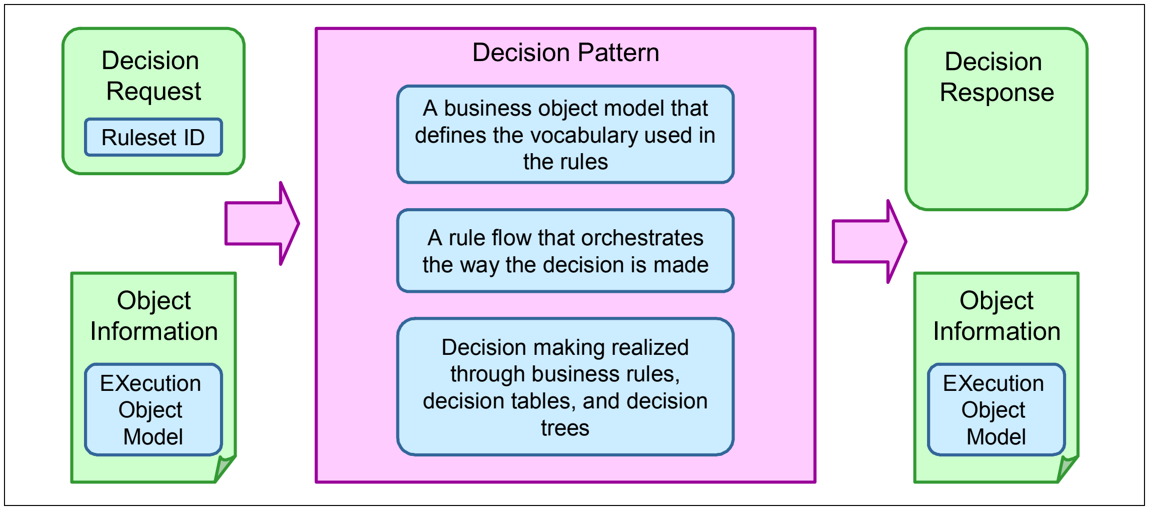

Decision Request
Ruleset ID
Object Information
EXecution Object Model
Decision Pattern
A business object model that defines the vocabulary used in the rules
A rule flow that orchestrates the way the decision is made
Decision making realized through business rules, decision tables, and decision trees
Decision Response
Object Information
EXecution Object Model

Figure 7-8 General decision pattern

Decisions have a unique identifier and an execution object model that describes signature of the identifier. The decision encapsulates a combination of business rules to implement the decision.

The BRMS solution from IBM has a unique rule flow capability for managing the order of medium to complex decisions and a business object model for defining the vocabulary for describing the business rules in a manner that is understandable by both business and IT. Simple rules engines that are embedded in BPM products do not offer these capabilities.

After the decision has been executed and multiple rules have been fired, the decision returns its response to the process with the appropriate output return values set in the execution object model.

7.2.1 Validation pattern

The process can invoke decisions in the BRMS solution for validation of different data which has been passed to the process. This pattern normally provides a “go-or-no-go” response and returns the decision result.

An assumption in all these cases is that the process analyzes the result and failures fields and takes some action as to how to best handle the next set of appropriate activities in the flow.

Figure 7-9 shows a validation decision pattern.

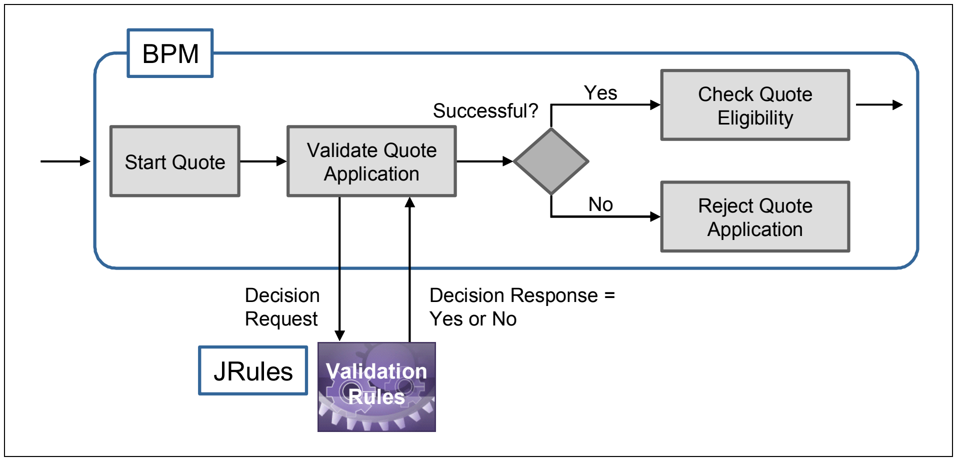


Figure 7-9 Validation decision pattern

The decision that is implemented with JRules can be either a simple rule flow and single rule set, or a more complex rule flow and rule set pattern.

Typical process decision types with this pattern are as follows:

- Validation of party or item information
- Eligibility of a party to a service or process
- Fraud detection

Validation of party or item information

In an *order to cash* process, validation of the order information is commonly done by using this validation pattern, for example, order part number, color, and quantity. The item that is ordered might be incorrect, might have been replaced by a new product, or might be discontinued.

Any failures must be accompanied with the specific information fields that failed the validation check to allow the process to route the order request to the appropriate handling point. Examples of handling points include reject order if product is unknown, assisted service if the part has been superseded, and determine whether the client wants new version.

Eligibility of a party to a service or process

In a *banking account opening* process, check the eligibility of the party to open the account based on the input data from the account that is opening request, other banking records retained by the company, or external data enquiries that might be required prior to agreeing to open the account.

Any failures must be accompanied with reasons about why the request failed, for example, the client already has an account of this type, the client does not have a valid address, or the client has a credit rating problem.

Fraud detection

In a *banking account opening* process, perform an anti-money laundering (AML) check to validate the person's identity before opening an account. Any failures must identify which part of the check failed for further investigation.

7.2.2 Classification pattern with process routing pattern

In this section, we look at the classification pattern with the process routing pattern.

Client classification in service and process flow routing

In a process that has separate process execution paths based on the type of a business item, the classification pattern should be used. In the example of an insurance claims process, different classifications or types of claims, such as windshield replacement, minor bodywork damage, major repair, and so on, might exist. In processes involving many clients, the clients might be classified for how they are to be treated. The clients are sometimes referred to as Gold, Silver, Bronze classified clients, and might have different levels of service based on paying different rates to have differential service.

In this case, use a decision service to determine and return the classification information. When the classification information is received by the process, use this flow logic inside the process to make a decision on how to best route the request.

When the classification information is received by the process, use this flow logic inside the process to make a decision about how to route the support service request to an appropriate automated service or assisted service for higher risk, higher value, or more critical problems.

Figure 7-10 shows a classification decision pattern.

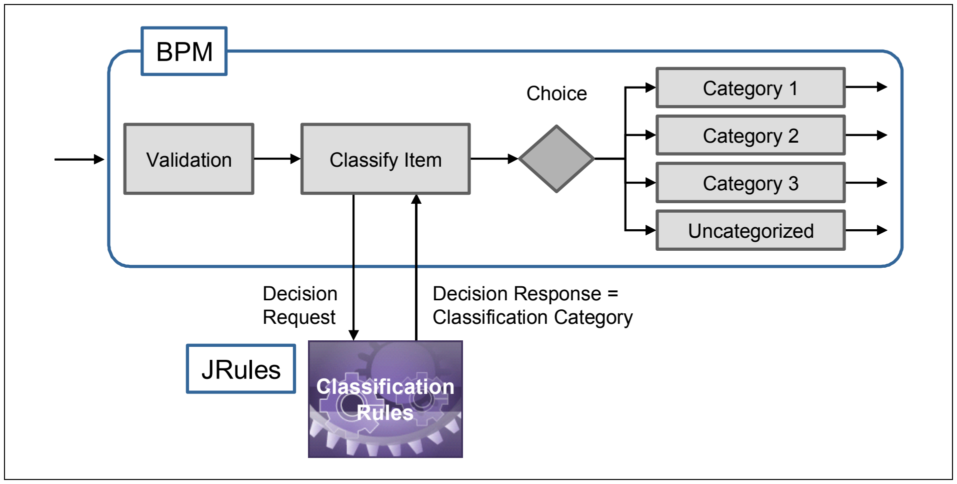


Figure 7-10 Classification decision pattern

The decision implemented with JRules may be either a simple rule flow and single rule set, or a more complex rule flow and rule set pattern.

7.2.3 Calculation pattern

Rules are often used to define algorithmic calculations that can be changed to meet business needs, for example, pricing and discounting rules. Within this pattern, the process is not expected to route the process flow to multiple branches, because only a derived information result is being returned, unless the calculation itself has a failure.

Pricing, billing, and entitlement calculation

Pricing and billing calculations often must consider various complex and interrelated business requirements and rules to form a combined set of rules, which determines all aspects to be considered when calculating a unique entitlement or price. With many influencing business rules, the expectation is high that the rules influencing this consideration will change frequently. Externalizing this consideration in a decision service separate from the process provides increased business agility and reduced effort to change. Examples of this rule include based entitlement calculation (pension payments), benefit payments, and item prices.

Figure 7-11 shows a calculation decision pattern.

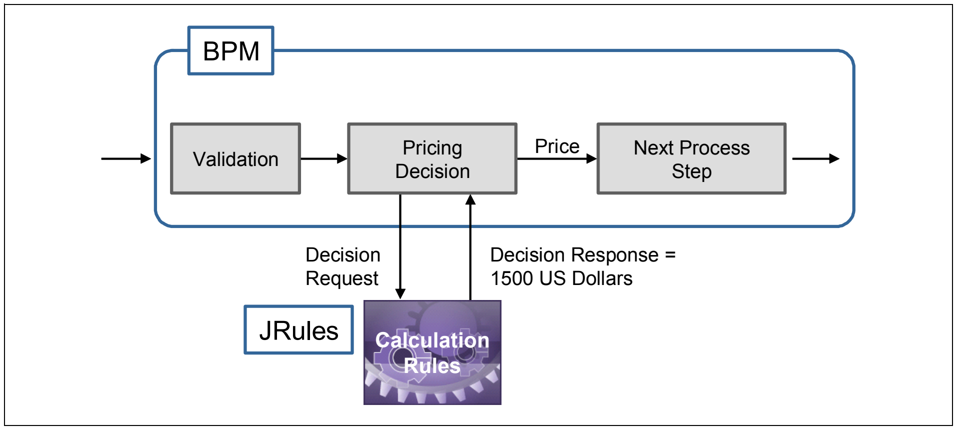


Figure 7-11 Calculation decision pattern

For prices, billing marketing incentives might also have to be considered for early payment incentives or customer loyalty schemes.

The decision implemented with JRules can be either a simple rule flow and single rule set, or a more complex rule flow and rule set pattern.

Insurance quote

On an insurance quotation or underwriting process, after the basic validation checks have been done, a calculation is required that is based on multiple information inputs from a client against many internal business rules. This information determines what the unique quote should be based on: age, status, previous insurance history, and many other factors.

7.2.4 Scoring and selection pattern and process routing

In a new credit card application process, one process activity is to obtain the requestor's current credit score. This decision can be based on information from external credit agencies and a combination of specific business rules within the business.

After the decision returns the score, the process flow activity can implement a process branch selection based on the returned score value. For example, if the returned score is less than 300, a refusal response is returned. If the returned score is in the range of 300 - 500, an acceptance application for normal credit card is returned. If the score is greater than 500, consideration for premium credit card offerings is returned.

Figure 7-12 shows a scoring decision pattern.

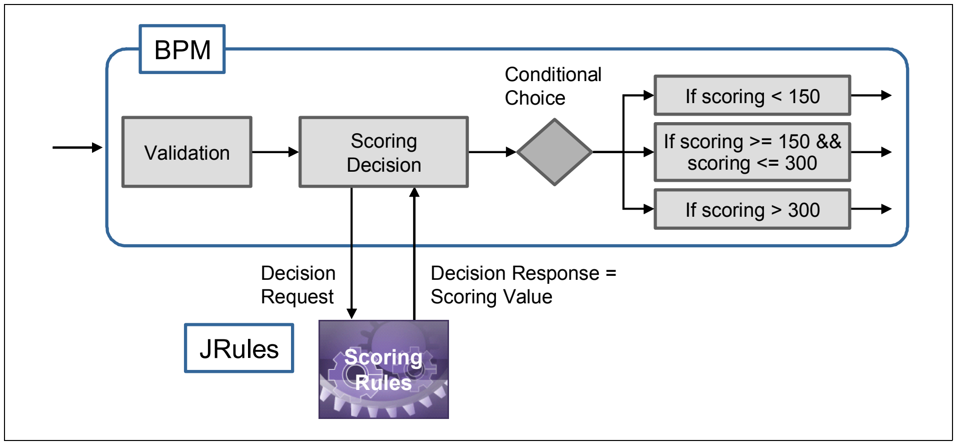


Figure 7-12 Scoring decision pattern

In many cases, the decision variability that is encapsulated by the decision service can be combined with another activity with service selection so that the process execution flow does not need to branch.

Service selection routing can be achieved by a request to a service registry that supports dynamic service binding. In this case, the conditional choices that are used to select various services can be encapsulated and would not be visible in the main process flow, as shown in Figure 7-13.

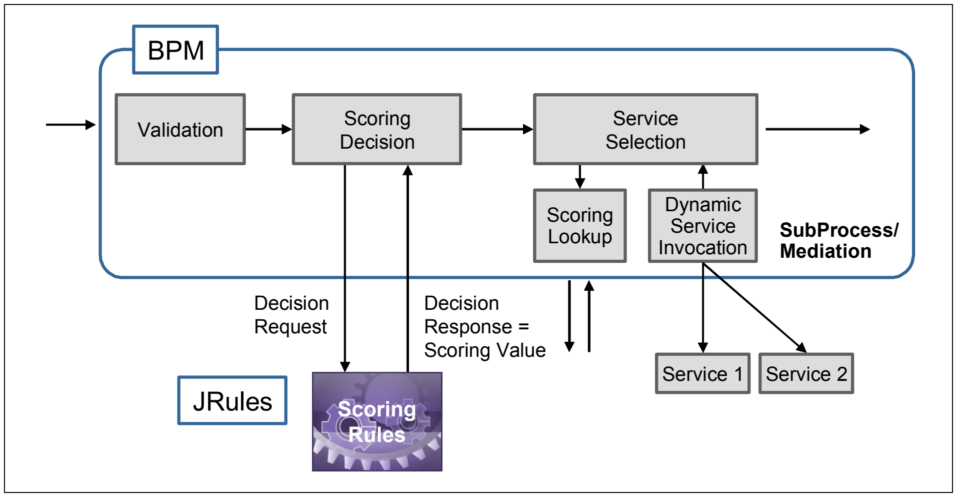


Figure 7-13 Scoring decision pattern and dynamic selection without multiple branches

The scoring decision that is implemented with JRules can be either a simple rule flow and single rule set, or a more complex rule flow and rule set pattern.

7.2.5 Compliance pattern

Today, the emphasis is strong for meeting legislation and regulatory compliances for doing business. To avoid legal and financial consequences, businesses must ensure that these compliance rules are followed. The regulations change over time and new regulations are introduced.

Organizations must implement these changes quickly and distribute them to all affected departments. Therefore, externalizing all these business rules from the process in a central BRMS repository offers a huge business benefit, especially when the rules are reused in multiple applications and processes.

7.2.6 Decision patterns within the decision: calculation pattern

Rules are often used to define algorithmic calculations that can be changed to meet business needs, for example, pricing and discounting rules.

The decision can have multiple steps, with each step using a separate pattern, as shown in Figure 7-14.

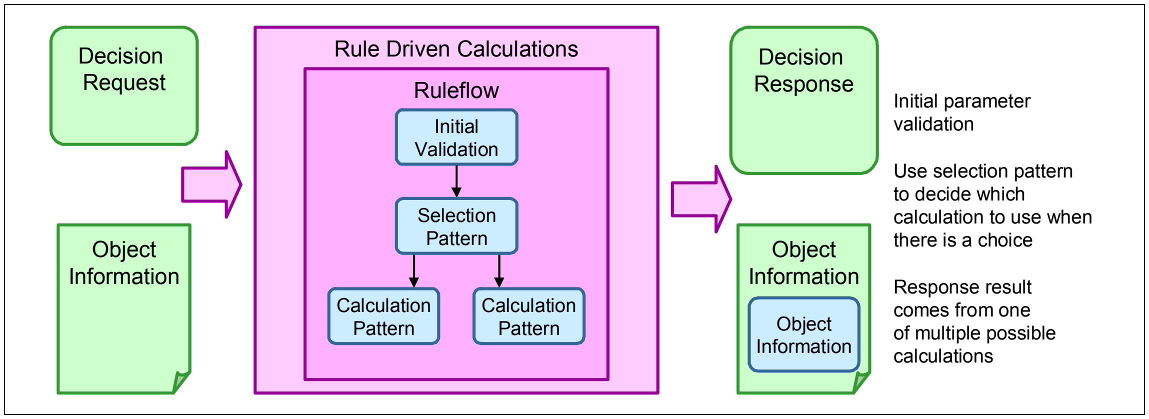


Figure 7-14 Rule flow using decision patterns within a calculation decision pattern

Pricing and billing calculations must often consider various complex and interrelated business requirements and rules to form a combined set of rules. This set can determine all aspects to be considered when calculating a unique entitlement or price. Using rule flow, and having a subset of business rules to run specific conditions or contexts makes the decision easier to manage and, in some cases, can be more efficient.

The decision patterns that are implemented in rule flow inside the decision service can change without the BPM process decision pattern changing.

7.3 Considerations for BRMS with BPM solutions

The BRMS solution from IBM, WebSphere ILOG JRules v7, can be integrated with many BPM solutions. See Figure 7-15.

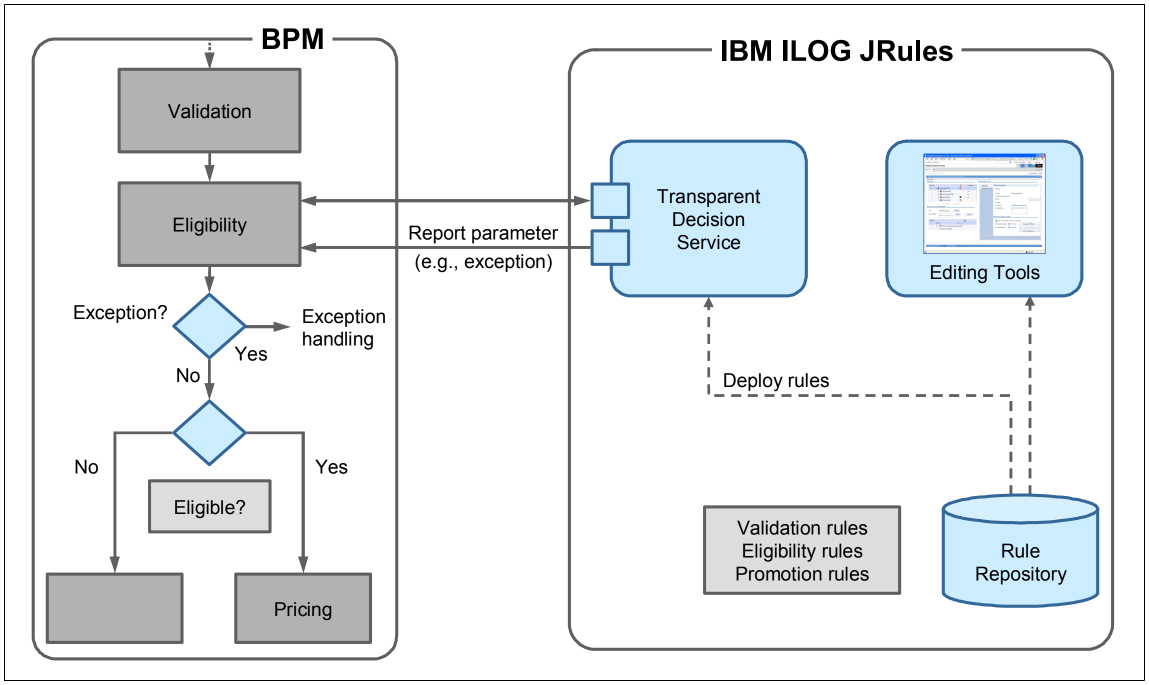


Figure 7-15 Process activity flow invoking decisions

Before going into detail about how to use JRules decisions with the BPM runtime process, consider all BPM aspects starting from the dual life cycles to identify all aspects of integration, as shown in Figure 7-16 on page 119.

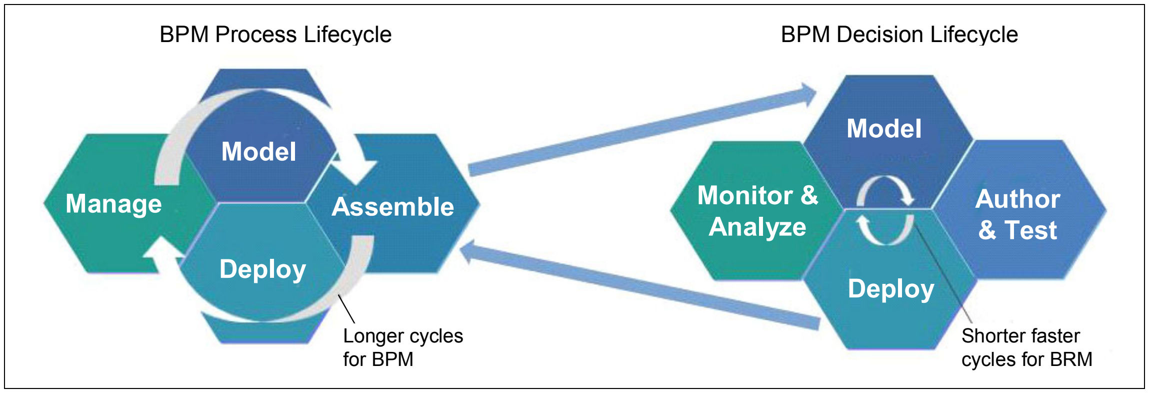


Figure 7-16 Dual life cycle

For new projects, where rules and decisions might change at a various rate from the process flow, a dual life cycle is advantageous to make targeted changes, when required, to enhance agility and reduce the amount of change required.

During the process assembly phase where the process activities are bound to specific services to realize the implementation any new services are also being created. Decision services have their specific rules authored and tested during this phase in parallel to process assembly. After the decision services have been fully tested and completed, they decision services can be bound to the appropriate decision points in the process activity flow and the input and out data mapped where required to finalize the process assembly phase. The whole process should be tested before the solution is fully deployed. This allows the process and decision services to be managed and monitored during normal operations.

To obtain the best agility from such an implementation it must be modeled and assembled so that the following items can occur:

- Simple changes to a decision service (implemented by business rule changes within it requiring a new version of the decision service) require no changes to the process.
- Simple process activity flow changes that must have a new version of the process require no changes to the decision services.

Only when the data object information in the decision pattern interface changes in the decision service is when the process and decision service both have to change.

Taking a top-down approach, we can start at the modeling phase and move through each phase to identify key aspects of the integrated solution.

7.3.1 Modeling considerations

When modeling a business process and the process requires decisions, multiple important aspects must be considered.

Process modeling

Consider the following questions about process modeling:

- Which process model activities are decision points?

  Identify how many decisions are within a process flow or subprocess. That is key in the modeling phase.

- Which decision patterns are required for decision points?

  Identify what decision patterns are required, for example, validation, scoring, routing, and calculations. This identification can help identify any additional logic in the process that is required to make process branching decisions based on the decision response. The output object model structure will depend on the decision pattern chosen because each has unique requirements.

- What business object models are required in the process model?
- How will the interface be handled between the process business object models and the decision service interfaces?
  - If decisions are being used in one process context uniquely, they can use the same business object models that the process uses to minimize mapping interfaces in the process, if it provides the correct business information to make the decision.
  - If business decisions and rules are expected to be reused in multiple processes and other applications, the business object model for the decision may be different from that used within the process. In this case, each time the process invokes a decision, a mapping activity is required both for the decision request and the decision response.
  - If there are existing industry-focused common schemas in the business that the process and decisions can share, these are a good starting point for reuse.
  - If the industry schemas are complex, verbalizing and authoring the rules in the authoring stage for business rules can be difficult. In some cases, a subset of the data structure can be passed, which contains the information that is required to make the decision.

In the example provided in Chapter 10, “Integrating WebSphere Process Server with JRules” on page 239, a common schema is used in the process and the decision.

Designing agility into processes which make decisions

Often, all the previous considerations are managed well, but this consideration is sometimes overlooked in process design.

A well designed process that is optimized for agility will focus on optimizing the interface between the process activity and the decision service, but does I not contain any references to a specific decision technology or decision service endpoint address. It also does not contain any business object mapping that is required for a specific decision or rule technology.

By encapsulating the technology, endpoint and object mapping outside of the main process activity flow, the process is kept simpler and is less likely to change because of changes in business decision services. If the technology is changed, changes to version or decisions are deployed to separate endpoint servers.

Modeling a process for encapsulating invocations to specific decision services in a subprocess or service mediation is a best practice for designing agility into processes and optimizing the independence of the process and decision life cycles.

If, at the modeling phase, decision technology specifics are part of the process model design, the process changes when the decision changes. In this case, both must be changed and redeployed together.

Decision modeling in JRules

For each unique decision that is required by the processes, JRules must provide a unique rule set ID at the assembly phase.

The decision request, response, and business object modeling, based on the decision patterns required by the process, are the main aspects of the decision modeling.

As described in “Process modeling” on page 120, these decisions may be aligned to meet the unique needs of a process where the business object models match closely to the process models, or the business object models are designed to meets the needs of making the decision reusable by multiple processes and applications.

Figure 7-17 shows an example of decision-pattern modeling.

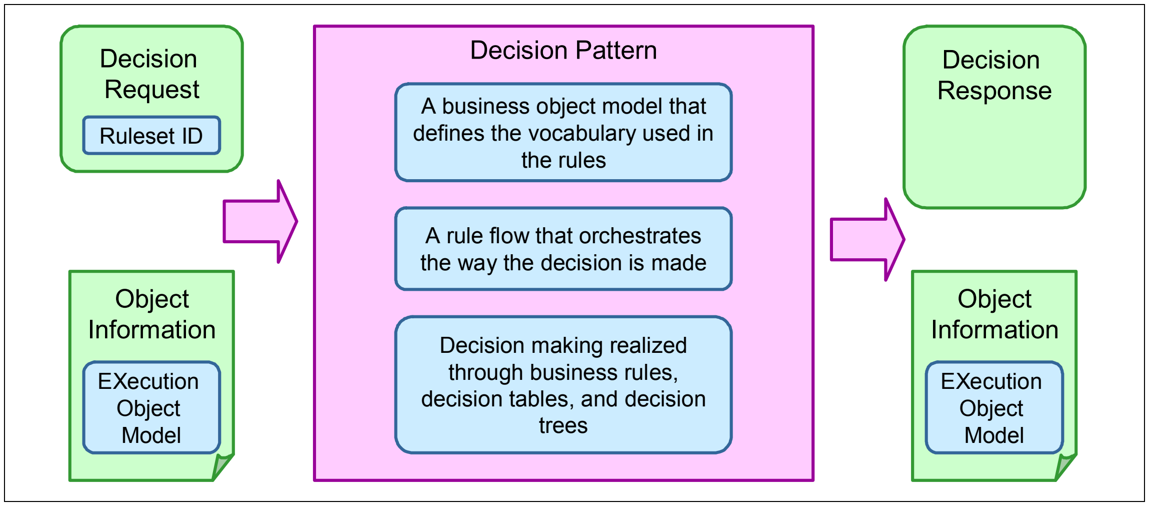

Decision Request
Ruleset ID
Object Information
EXecution Object Model
Decision Pattern
A business object model that defines the vocabulary used in the rules
A rule flow that orchestrates the way the decision is made
Decision making realized through business rules, decision tables, and decision trees
Decision Response
Object Information
EXecution Object Model

Figure 7-17 Modeling the decision pattern object models and vocabulary

The final operation in the modeling phase is refining the business object model to agree on a common vocabulary to be used for how to articulate the business rules prior to them being authored. This approach has the benefits of having consistency for the business and IT teams that use the vocabulary, and improving productivity in the Assembly phase and in the ongoing change phases.

7.3.2 Assembly considerations: JRules decisions or BPM decisions

Each process is unique in its needs and must be assessed in its own right. Processes with multiple decisions might use different decision patterns and might be implemented using various technologies that depend on the type of decision, how frequently it changes, and who is expected to make changes to it.

Consider the following questions when determining whether to use internal BPM rules or JRules:

- What type of decision pattern required?
- Are there requirements that roles in the organization other than the process owner or developer are expected to change the decisions and rules?
- Do any decisions and rules change more frequently than other aspects of the process?
- Do any specific governance, audit, compliance requirements regarding decisions and rules exist?

- Are there specific requirements for decision services or rules to be shared with processes or other applications?
- Are enterprise rules required to be deployed to multiple applications and platforms (Java, COBOL, .NET)?
- Are there specific requirements for Decision Services to be modeled, simulated and tested independently of the processes that consume them?

Decision pattern type and rule complexity

Depending on the decision type, complexity, and other factors, a choice must be made regarding how the decision is implemented. An initial starting point prior to taking the other factors into consideration are as follows:

Use rules in BPM for the following items:

- Validation patterns of with simpler combinations of rules
- Process routing decisions
- Human task assignment decisions

Use JRules for the following items:

- Validation patterns with more complex combinations of rules
- Classification and routing pattern where process does the routing
- Scoring, selection, and routing pattern where the process does the routing
- Rule-driven calculations
- Compliance patterns

Frequently, process decisions for validation can start small and over time grow more complex because of additional requirements being added. As complexity increases, migrating some rules that were initially implemented in BPM rules to JRules might be a requirement.

In addition, as Figure 7-18 shows, even when a decision is implemented in JRules, the process is sometimes required to provide decision logic to evaluate the results from a decision (indicated by Yes or No in the validation pattern).

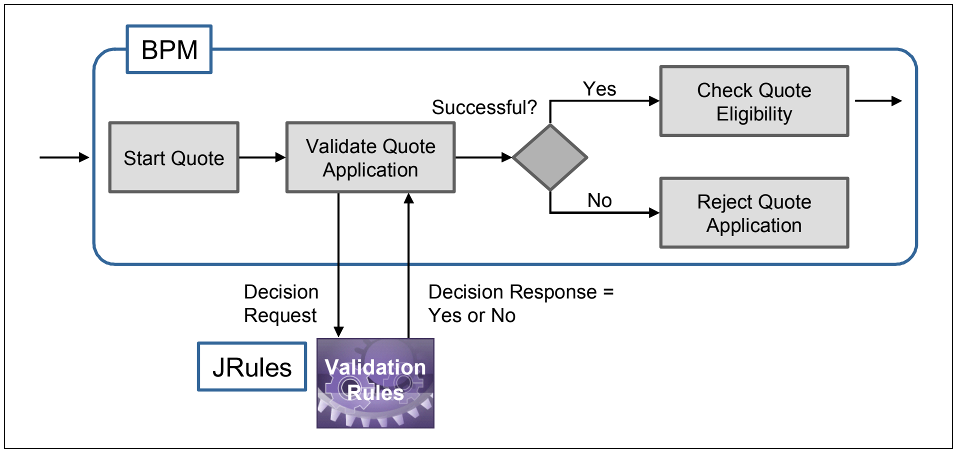


Figure 7-18 Validation decision pattern

What roles are required to change business rules

Traditionally, all business requirements are documented for policy and rule changes and then provided to IT and process owners to implement. JRules enables other roles to be able to view and change specific rules.

Use JRules to improve time-to-value and reduce ongoing minor IT updates. Policy managers and business analysts who specialize in pricing, risk, eligibility, and compliance are required to change rules specific to their domain of skill and knowledge by using JRules business tools.

Decisions and rules that change more frequently than process

Some processes have business decisions and rules that change more frequently than the process activity flow. In those cases, we can simplify the process and reduce the costs for process maintenance updates by updating the decisions services more frequently.

This approach has agility benefits when managed as a dual life cycle, as shown previously in Figure 7-16 on page 119.

Use JRules when the business needs a decision to be updated and maintained at a different rate from the core process that invokes it.

Specific governance, audit, and compliance rule requirements

Certain industries must comply with specific regulations and might be audited and demonstrate proof of compliance. Tight controls exist concerning who has the ability to modify the rules and decisions in processes and applications.

Using JRules to centrally manage the decisions and rules that have to comply in a consistent enterprise manner reduces the need and effort that is required for multiple processes and applications solutions to comply in the same manner.

Sharing decisions and rules in processes and applications

Some common business policies and constraints might apply in separate business functions, processes, and applications. Businesses can reduce effort and increase agility by centralizing and sharing these, minimizing effort and expense.

Using JRules can provide central sharing of decisions and rules in a well-governed manner. This approach enables the consuming processes and applications to indicate specific versions or the latest versions of these decisions, if required.

Decisions and rules deployed in multiple environments

Many businesses run various parts of their infrastructure in separate environments, such as J2EE Application Servers, Mainframe, and .NET servers. In certain cases, various common business functions have separate implementations because of mergers and acquisitions. Also, separate channels to a common business function might have different performance and availability needs. However, they often share common decisions so that each channel, business unit, and geography provides the same consistent result to clients. Use JRules to centrally host the decisions and rules that are shared in this way.

7.3.3 Assembly considerations: Using JRules rule flow with process orchestration

Before finalizing any detailed assembly pattern, we consider process orchestration and the BRMS rule flow capability.

Considering rule flow capabilities with BPM solutions

A best practice is to identify the high-level decision points and make them visible in the process orchestration flow. This practice can help make the process model easier to understand and the main decisions point clearly visible. It also enables KPI process monitoring, if required, at each decision point to more easily monitor process performance. An example might be calculating the percentage of valid insurance applications and eligible insurance applications compared to all insurance applications.

The BRMS rule flow capability enables sequencing of decision components within more complex decisions.

The patterns for rule flows in BPM solutions are as follows:

- Simple rule flow
- Multipart decisions
- Decisions with Different Context

Rule flow pattern with BPM 1: simple rule flow

This rule flow is only a single step, which invokes a single rule set to implement the decision.

Rule flow pattern with BPM 2: multipart decisions

In this pattern, for example, if eligibility has three parts or components for the check to validate various aspects of the insurance quotation request, each can be implemented as a unique rule set. The rule flow can be used to sequence this check, and can be extended without the need to change the process orchestration flow. See Figure 7-19.

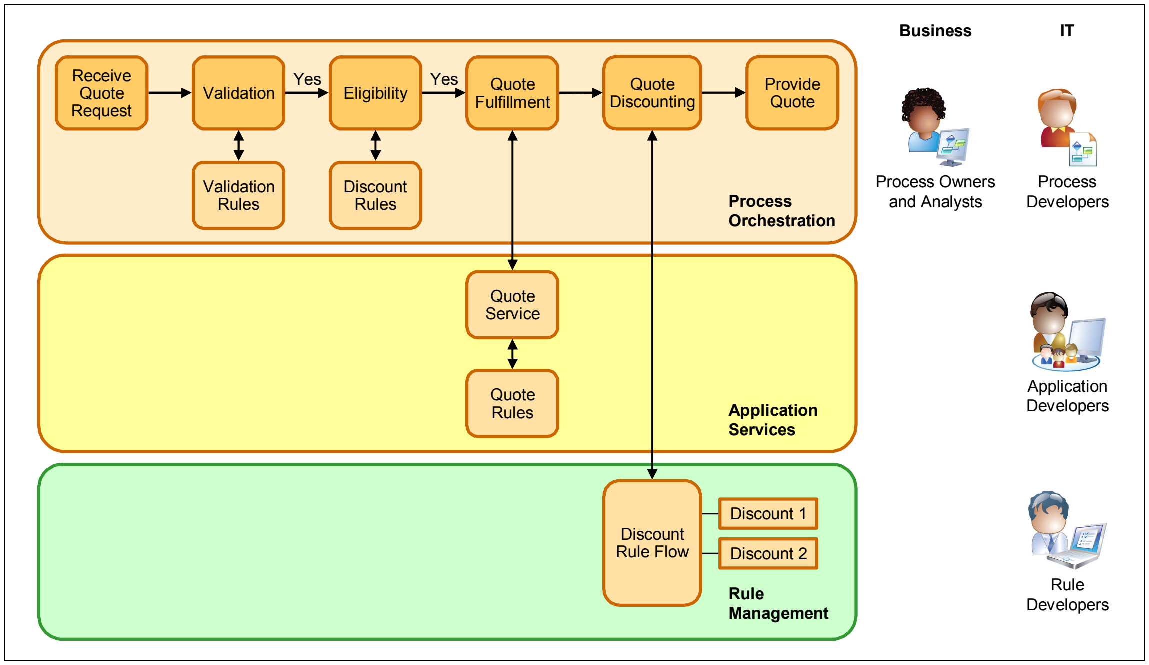


Figure 7-19 Using process orchestration flow and rule flow for a multipart decision

Looking at the rules aspect in more detail, Figure 7-20 demonstrates an example of a two-part decision regarding discounting, as follows:

- Using sequential rules for some initial discount validation, regarding the subset of people who are eligible for the discount.
- Invoking the calculation aspect of discounting using the RETE calculation engine inside JRules.

Performing the detailed validation for discounting only in the rule flow does not conflict with the general validation and eligibility checks that are invoked from the process directly. The rule flow can also invoke a decision table within a rule flow stage.

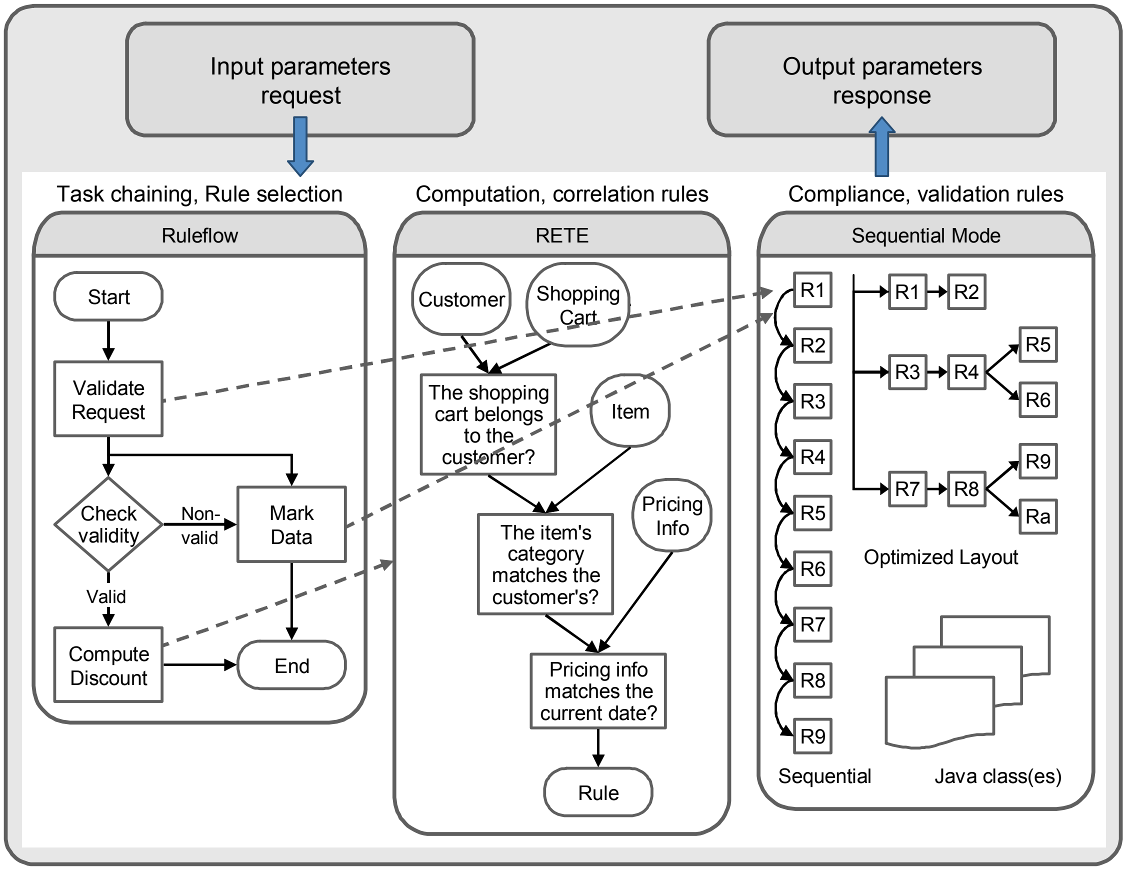

Input parameters request
Output parameters response
Task chaining, Rule selection
Computation, correlation rules
Compliance, validation rules
Ruleflow
RETE
Sequential Mode
Start
Validate Request
Check validity
Non-valid
Mark Data
Valid
Compute Discount
End
Customer
Shopping Cart
The shopping cart belongs to the customer?
Item
The item's category matches the customer's?
Pricing Info
Pricing info matches the current date?
Rule
R1
R2
R3
R4
R5
R6
R7
R8
R9
Ra
Optimized Layout
Sequential
Java class(es)

Figure 7-20 Multipart rule flow can invoke multiple JRules rule types

Rule flow pattern with BPM 3: decisions with different context

Many businesses have common processes, but with minor changes based on the context in which the process is being run. If we examine the insurance example, an insurance quotation process can be available for car insurance, buildings insurance, pet insurance, or health insurance. Depending on how the high-level process is designed, one possibility is to have a consistent and consolidated process orchestration flow with a generic insurance quotation process that invokes a generic, context-sensitive decision.

In this case, the rule flow can initially validate what the context of the decision is, which is passed from the process, and then invoke the rule sets and rules that align with that context. In this case, extending the context is easier, without requiring additional processes.

In certain complex cases, a subprocess may be used to handle some of the context choice, combined with multiple decisions, which are context-sensitive where major differences occur.

Uses of rule flow with BPM that are not best practices

The following two designs are either overuse or underuse rule flows:

- Minimizing the number of process orchestration steps to the extent that distinct process activity steps are implemented within the rule flow and are not visible at the process orchestration level

  This design minimizes the ability to Monitor process performance at these steps because they are hidden and makes it harder to understand what activity is actually being executed.

- Instead of using BRMS rule flow, a subprocess in BPM is created and calls multiple small decision steps, which could have been done in rule flow

  This design increases the overhead of calling between the process orchestration layer and BRMS multiple times in each process, unnecessarily.

7.3.4 Assembly considerations: JRules decision service integration and binding options for BPM

When assembling the process that needs external decisions, hosted and managed by JRules, two main architectural decisions must be made:

- Whether JRules decision services are invoked directly from the process or invoked indirectly from the process
- Which interfaces to use to bind the decisions to the process.

JRules has the following integration patterns that are commonly used by BPM solutions:

- Web service support using JRules hosted transparent decision services (HTDS) and custom web services patterns
- J2EE synchronous Rule Execution Server patterns (local and remote EJB and POJO patterns)
- J2EE asynchronous Rule Execution Server patterns (POJO pattern)

The direct invocation pattern options are shown in Figure 7-21.

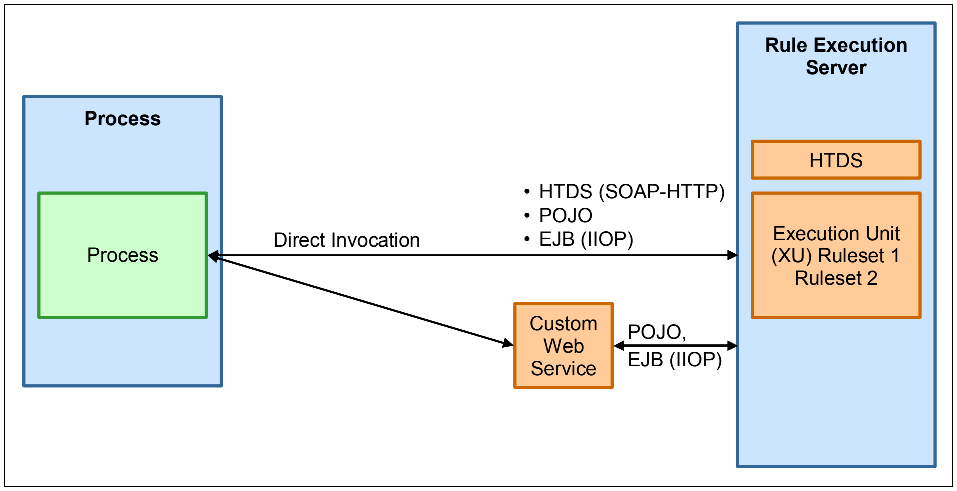


Figure 7-21 Process with directly invoked decision services

Although certain integration options, such as HTDS, EJB, and custom web services, can support the JRules Rules Execution Server running on a separate system, the process still has the following items:

- Any business object mapping between the process and decision service
- Various decision service endpoint information, such as endpoint address in HTDS-generated WSDL

In certain production scenarios, especially where the Process and Rule Execution Server are both installed on the same system, this may not matter and this design optimizes solution performance. However, when decision services have changes in the rules, or the servers they are running on changes, the process has dependencies on this change and must also change.

Using the indirect invocation pattern, all processes to decision service business object mapping, service endpoint information can all be encapsulated within the subprocess or service mediation patterns. Additionally, if the decision requires additional information that is not available in the process, this pattern can easily retrieve that information without the process requiring access directly. See Figure 7-22.

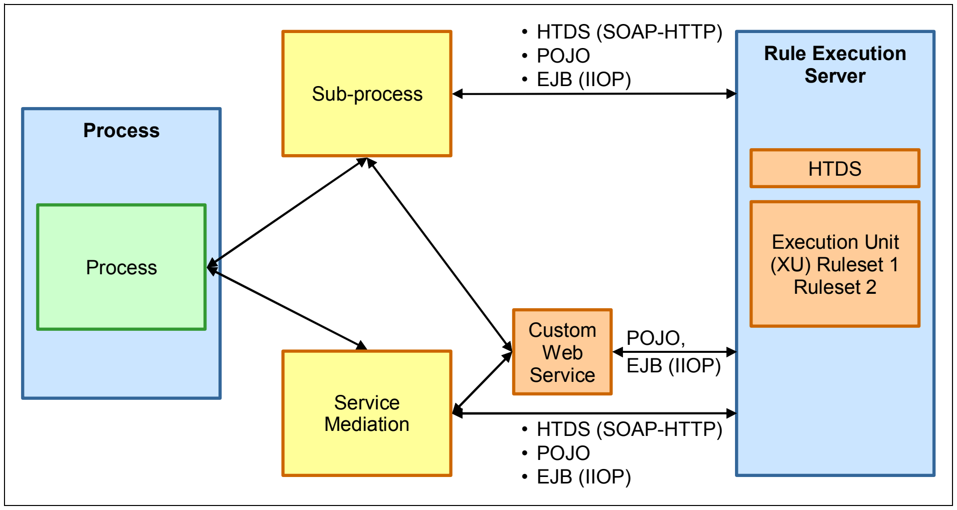


Figure 7-22 Process with indirectly invoked decision services

Where process and decision agility and process simplification are near the top of the business priorities, one of these patterns must be deployed.

Table 7-1 summarizes the capabilities of these integration patterns.

Table 7-1 Comparison of capabilities of integration patterns

Capability (pattern)	**Direct invocation**	**Service mediation**	**Subprocess**
Technical service with static service reference to the decision on Rule Execution Server	Yes	No	No
Business service that is agile and has the ability to virtualize service endpoint	No	Yes	Yes
Encapsulate the process to decision data mapping outside the main process to increase agility and minimize main process changes	No	Yes	Yes
Enrich information that is not in process prior to invoking decision	No	Yes	Yes
Dynamic lookup of service registry to match the best service endpoint to invoke	No	Yes	Yes
Top-down architecture modeling of both service, process names, and interfaces for decision services to use	No	Yes	Yes[a]

a. Might require some custom Java programming.

After understanding which option best meets the needs of the decision requirements, a choice of JRules interface integration pattern is required.

All BPM products support the web service integration pattern that provides either local or remote decision-coupling by using industry standards. The most common interface for implementing web services by JRules HTDS, which can be automatically generated from rule applications that are already deployed to JRules Rule Execution Server.

HTDS is an enterprise application that provides enpoints to the web service within RES. It is a service wrapper for the Session Bean and facilitates the binding to SOAP.

The data schemas that are supported with this integration patterns are as follows:

- XML schema
- XML schema with Java Utility classes that supplement the XML elements that have specific behaviors.

For process solutions that require direct invocation between the process flow and decision services, the web services that are generated by JRules can be directly imported by the BPM process assembly tools for the appropriate decision activities and invoked directly from the main process flow.

Note: A limitation with HTDS in JRules v7.1 is that it has one service signature for each JRules rule set.

The indirect invocation integration patterns with service mediations or subprocess, support the following scenarios:

- A process or subprocess has multiple decisions, and one business decision is encapsulated in a service mediation or subprocess.
- A top-down, model-driven approach is required and the designed decision service interface at the process level is required to be mapped onto the technical HTDS decision interface in the service mediation or subprocess.

Using J2EE Rule Execution Server API on both EJB and POJO JRules interfaces

The implementation of these interfaces requires that the processes create external services that wrap the RES execution API and provide options for multiple rule session providers (POJO invocation, local EJB, and remote EJB). Choose the session provider that best matches the deployment requirements:

- Select POJO invocation if you want the RES to run in-process with the process instance.
- Select remote EJB if you have an external instance of RES running in a separate application server, and decision support for transactions is required.

The following data schemas are supported using these integration patterns:

- XML schema
- Java schema

For process solutions that require close coupling between the process flow and decision services, any mapping of object models between the process and decision service must be done within the process.

The indirect invocation integration patterns with service mediation or subprocess support the scenarios where business object mapping and any service naming changes is done within the service mediation or subprocess.

7.3.5 Deployment considerations

How the Rule Execution Server that hosts the decision service and the business rules is deployed is a factor in the architectural design and performance of the overall solution.

Of the three main deployment options of JRules v7, only the following options are applicable:

- J2EE Rule Execution Server installed locally with process or applications
- J2EE Rule Execution Server installed on separate systems to where processes or applications are running

When setting up a demonstration or development system, having the BPM and BRMS solutions co-residing is easier to manage and test.

When larger production systems are required in a cluster of machines, BPM and BRMS solutions residing on separate systems is more common, especially where BRMS might also be hosting decisions for other processes and applications.

WebSphere ILOG Team Server and WebSphere ILOG Decision Verification Services can be installed on the same server as BPM solution or on a remote server.

7.3.6 Monitoring considerations

After the processes, decisions, and business rules are deployed and running, process performance can be monitored by using the process monitoring tools, and the business rules can be monitored by using JRules tools.

7.4 Integration with IBM BPM solutions

This section describes integration options for the following BPM solutions:

- WebSphere BPM, which includes WebSphere Process Server v7
- IBM FileNet® Business Process Manager v4.5
- WebSphere Lombardi Edition v7.1

This book does not describe when to select which BPM offering. The book does focus on how JRules adds value, when the BPM solution is chosen, and can be easily integrated to create an easy-to-use combined platform.

7.4.1 Integration with WebSphere Process Server-based solutions

IBM WebSphere Process Server is a high-performance business process automation engine to help form processes that meet your business goals.

Built on open standards, it deploys and executes processes that orchestrate services (people, information, systems, and trading partners) within your service-oriented architecture (SOA) or non-SOA infrastructure. It helps increase efficiency and productivity by automating complicated processes that span people, partners, and systems. It helps cut costs by enabling flexible business processes with reusable assets, reducing the need to hard-code changes across multiple applications.

WebSphere Process Server v7 is the runtime deployment component of the WebSphere BPM solution. Additional solution components are required to implement the solution patterns and the four-phase process life cycle, shown in Figure 7-23.

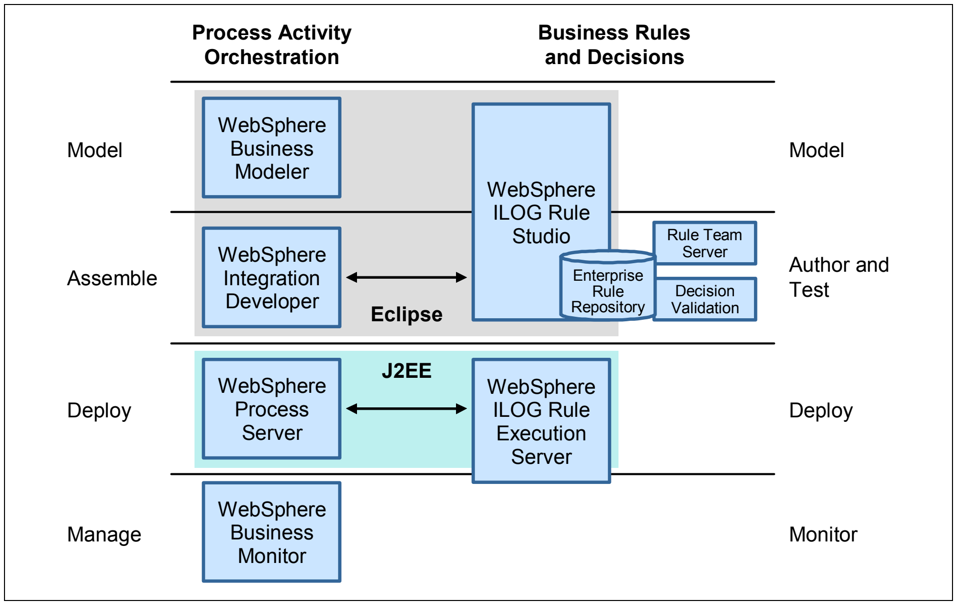

Process Activity Orchestration
Business Rules and Decisions
Model
Assemble
Deploy
Manage
Model
Author and Test
Deploy
Monitor
WebSphere Business Modeler
WebSphere Integration Developer
WebSphere Process Server
WebSphere Business Monitor
WebSphere ILOG Rule Studio
Enterprise Rule Repository
Rule Team Server
Decision Validation
Eclipse
J2EE
WebSphere ILOG Rule Execution Server

Figure 7-23 Joint solution tools and runtime components

Figure 7-23 on page 134 shows that the BPM solution has a specific product that is used in each of the four phases of the life cycle:

- WebSphere Business Modeler and WebSphere Business Monitor

  These modeling and managing life cycle tools are primarily focused on the business process analyst. This focus can minimize the gap between business requirements and IT design, implementation, and monitoring real performance.

- WebSphere Integration Developer and WebSphere Process Server

  These tools are used in the Assemble and Deploy phases. These are IT based tools and runtime products focused on Business Process Execution Language (BPEL) and SCA standards for process orchestration.

On the business rules and decisions side, JRules Rule Studio can fully model, author, and test the required decision interfaces-based and business rules and deploy the decisions to Rules Execution Server. Business users can view, change, and test the business rules with Team Server and Decision Validation Services,

External tools for creating enterprise business objects that both BPM and BRMS can share are optional, as is external monitoring of business rule performance.

This combination supports all three solution patterns that are described in 7.1.3, “Solution patterns using BPM and BRMS” on page 99, and all decision patterns that are described in 7.2, “Decision patterns with processes” on page 111.

Solution pattern 1

IT uses BRMS to complement BPM with rule authoring and deployment. This pattern is sometimes used as the entry pattern to evaluate the combination of BPM and BRMS. The required combination of product components are as follows:

- WebSphere Integration Developer and WebSphere Process Server are the minimum required BPM components.
- WebSphere Business Modeler and WebSphere Business Monitor can be used by both business and IT users and may be present in the pattern.
- WebSphere ILOG JRules Rule Studio and Rule Execution Server is required.

Solution pattern 2

This pattern uses BRMS to improve business collaboration and agility. This pattern is the most common; both IT benefits and business collaboration and agility benefits can be achieved.

The required components are as follows:

- Similar product components as in pattern 1.
- WebSphere ILOG Team Server for Business collaboration to view and change business rules in a central repository.
- In addition, the add-on solutions for Team Server are as follows:
  - WebSphere ILOG Rule Solutions for Office, when business users used Microsoft Office 2007 for capturing business rules.
  - WebSphere ILOG Decision Validation Services for simulation, test and what-if scenarios.

Solution pattern 3

This pattern uses central authoring and management of business rules. This pattern is often used after multiple joint BPM and BRMS projects or where a BRMS solution is already established and is extended to centrally manage decisions for processes that are orchestrated in BPM and other applications. This pattern builds on the deployments from the other patterns and increases the usage of them without adding any new product components.

Because the BPM and BRMS solutions work well together, positioning JRules and its four-phase life cycle at the same time is useful. See Figure 7-23 on page 134.

Modeling considerations

The modeling tool of choice for solutions with WebSphere Process Server is WebSphere Business Modeler. It documents process activity flow for processes by using industry standards such as BPMN and helps to more easily identify which activity steps are business decisions.

When a decision point occurs in a process, the two methods to designate the activity step in the process model that invokes a decision are as follows:

- For decisions implemented in JRules, a general activity is selected and comments are added to identify it as a decision that will be associated with a JRules decision during the assembly phase. With this method, JRules business rules tools can be used to model the decision interface in JRules Rule Studio and author the business rules in a natural language, prior to assembly in JRules Rule Studio and Rule Team Server.
- For decisions implemented locally in WebSphere Process Server, a special business rule activity is selected in the workflow step. This method allows the WebSphere Process Server decision interface to be defined and simple business rules authored in a programatic manner within the modeling tool.

Depending on the decision type, complexity, and other factors, a choice is required as to how the decision is implemented. In many cases, the choice for implementation technology for the decision is better made during the assembly phase, where more details about this process are provided.

Figure 7-24 demonstrates multiple decisions in WebSphere Business Modeler, implemented as general process activities, and shows the following information:

- Validation decision activity is a validation pattern.
- CheckEligibility decision activity is a validation pattern.
- ApplyPromotion decision activity is a rule-driven calculation pattern.

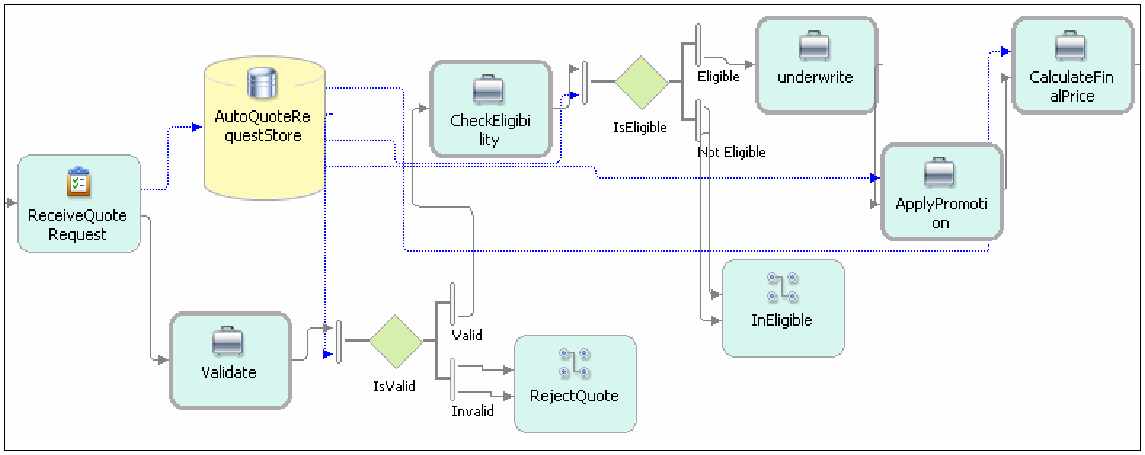


Figure 7-24 Process model of process

Business object models

WebSphere Business Modeler can create simple object models. However, in most projects with levels of complexity, be sure to use external tools to create the business objects and then import them into WebSphere Business Modeler. WebSphere Integration Developer is one tool that can be used for this task.

Designing agility into processes that make decisions

WebSphere Process Server supports modeling a process to encapsulate invocations to specific decision services in a subprocess, or service mediation as a best practice for designing agility into processes, and optimizing the independence of the process and decision life cycles.

The process can benefit from being simplified by moving any business object mapping that is required for specific decisions into the subprocess or service mediation, where it can be changed if the decision changes its interface. This pattern also abstracts any technology or service endpoint specifics from the generic process model.

The required subprocesses can be modeled in WebSphere Business Modeler as microflows or short running processes. Any service mediations can be created as services during the assembly phase in WebSphere Integration Developer.

Assembling and deploying considerations

The assembly tool of choice for solutions with WebSphere Process Server is WebSphere Integration Developer. This tool can take the documented process activity flow model in BPMN, from a modeling tool such as WebSphere Business Modeler that, when exported using BPEL standards, can be used for process assembly and run time.

With the decision points and decision types that are previously identified in the Modeling section, the role of WebSphere Integration Developer is to select and bind the decision activity to the most appropriate decision implementation at the BPEL level.

Figure 7-25 shows an example of a BPEL process in WebSphere Integration Developer.

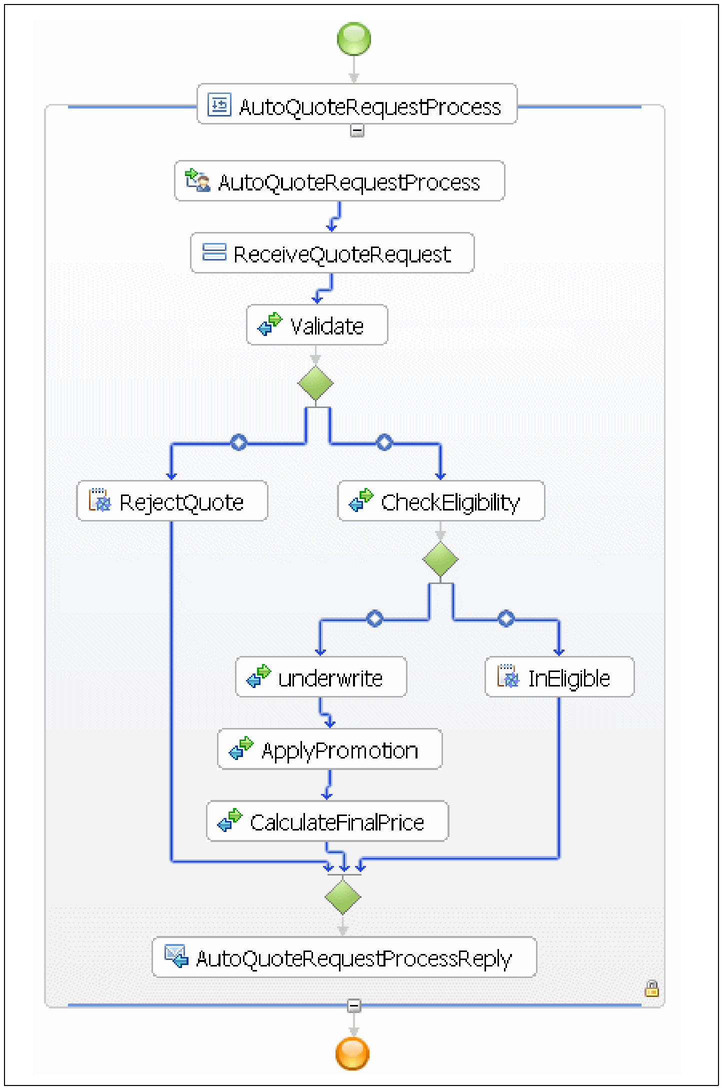


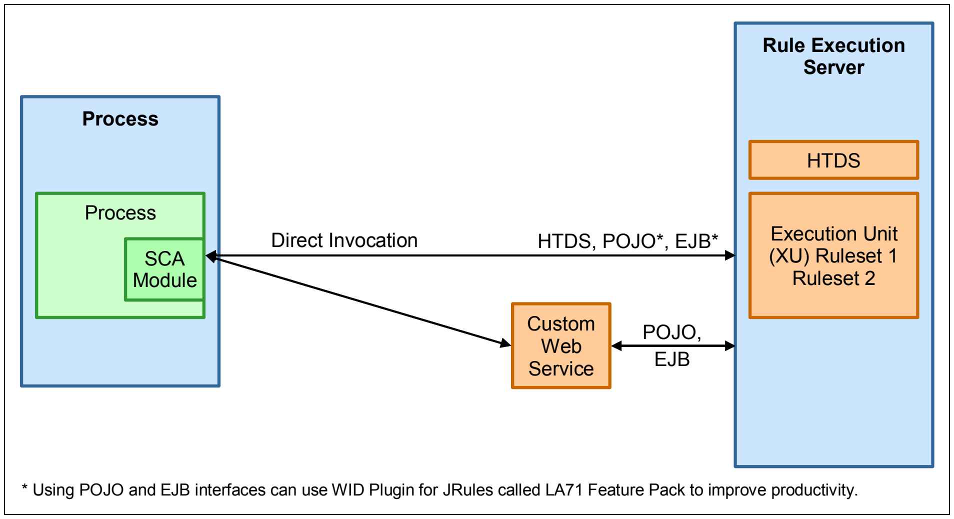
Figure 7-25 BPEL process representation in WebSphere Integration Developer

Decision implementation choices

Depending on the decision type, complexity, and other factors, a choice might be required regarding how the decision is implemented: either in WebSphere Process Server Rules or in JRules. This criterion is provided in 7.3.2, “Assembly considerations: JRules decisions or BPM decisions” on page 122.

JRules integration and binding options

WebSphere Process Server can support both a direct invocation (Figure 7-26) and an indirect invocation (Figure 7-27) pattern for decisions.



Process
Process
SCA Module
Direct Invocation
HTDS, POJO*, EJB*
Custom Web Service
POJO, EJB
Rule Execution Server
HTDS
Execution Unit (XU) Ruleset 1 Ruleset 2
* Using POJO and EJB interfaces can use WID Plugin for JRules called LA71 Feature Pack to improve productivity.

Figure 7-26 Direct invocation of decisions from process using SCA

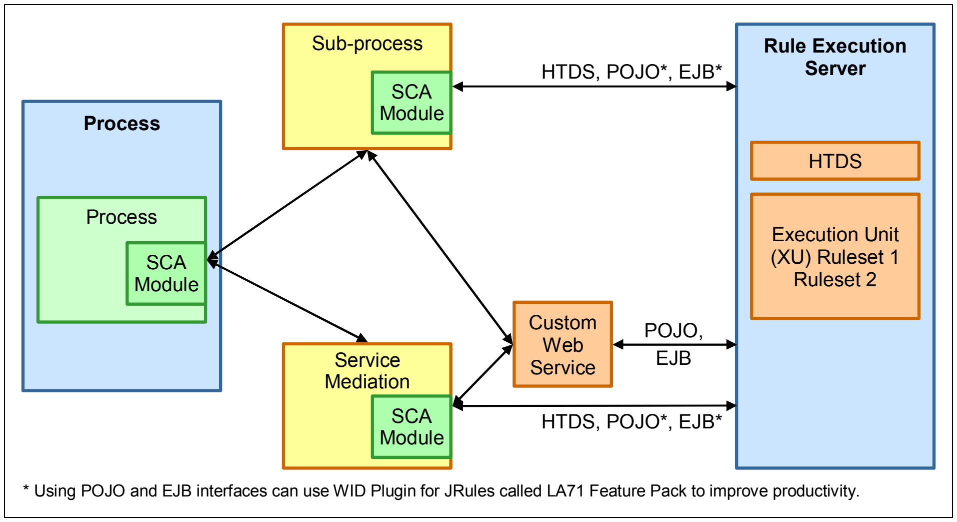


Figure 7-27 Indirect invocation of decisions from process using SCA

For both direct and indirect invocation patterns, use WebSphere Integration Developer as the main tooling for assembly.

JRules offers two integration options for WebSphere Process Server projects:

- ► Web services
- ► Binding for JRules interfaces, which is created with SCA modules

Web services

JRules decisions may be deployed to WebSphere ILOG JRules Rule Execution Server (RES), where they may be reused by any standard SOAP client. With the ready-to-use features in WebSphere Integration Developer, a developer can more easily import a WSDL from RES and use it within a business process.

WSDL is provided by HTDS, which is an enterprise application that provide the service end-point within RES. It acts as a service wrapper to the JRules session bean and facilitates the binding to SOAP.

Custom web services that invoke decisions in JRules can be integrated in a similar way. See Figure 7-28 (in the figure, WID is WebSphere Integration Developer).

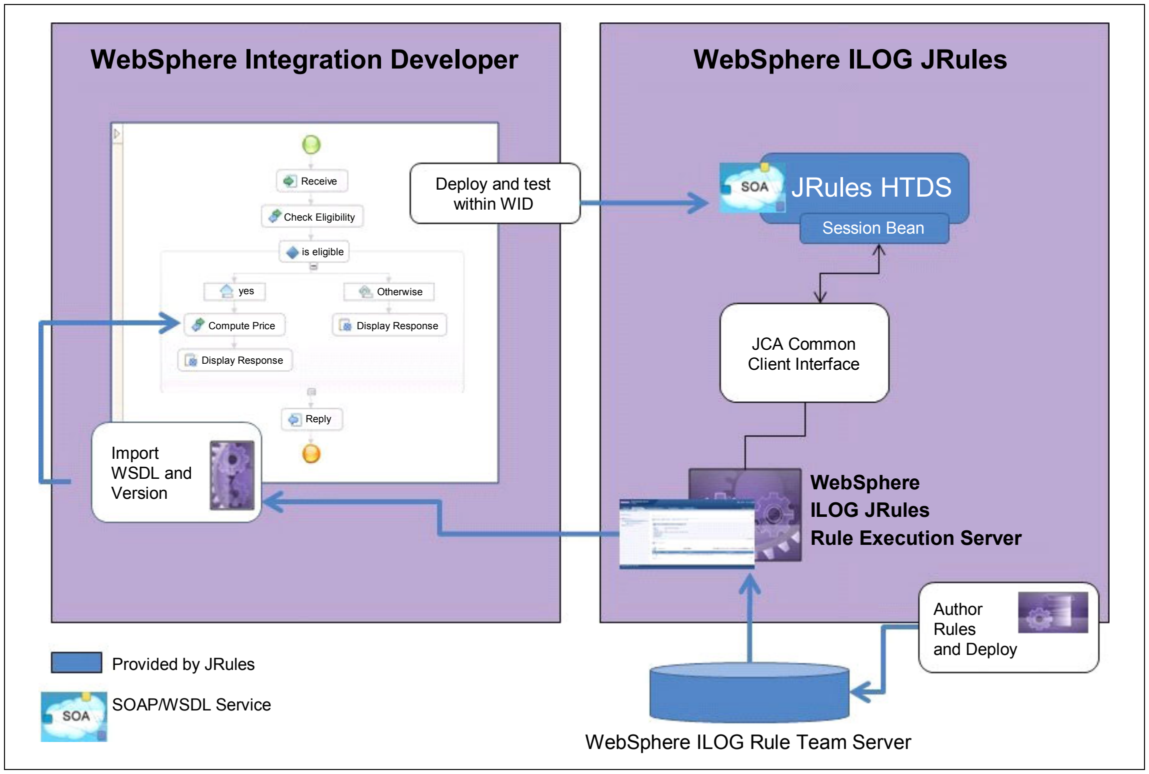


Figure 7-28 Using JRules through web services

The WSDL import process can generate many of the required WebSphere Integration Developer project artifacts, including the interface and business objects (datatypes). JRules provides two options for WSDL:

- Get HTDS WSDL for this rule set version

  This option provides the WSDL for a specific version of the decision. Use this version if you do not want the decision to change for existing long-running process instances.

- Get HTDS WSDL for the latest rule set version

  This option always provides the latest version of a decision. All existing and new process instances will take advantage of dynamic changes to the decision.

Decision service SCA module using internal JRules interfaces

The implementation of the service component wraps the RES execution API and provides options for multiple rule session providers from JRules. The interface options from the RES execution API are POJO invocation, local EJB, and remote EJB. This implementation is in an optional WebSphere Integration Developer plug-in that is provided by JRules, called LA71, which is an optional capability.

The plug-in can run a wizard that creates a WebSphere Integration Developer module, which encapsulates the JRules EJB or POJO invocation.

Choose the session provider that best matches your deployment requirements. For example, you can use the following session providers:

- POJO invocation, if you want the RES to run in-process with the process instance
- Remote EJB, if you have an external instance of RES running in the data center, which is more likely for production scenarios
- Local EJB, if you want to avoid serialization overhead but still need transaction support

Finally, you must use one of the EJB options if the business decision participates in a transaction. See Figure 7-29 on page 143 (in the figure, WID is WebSphere Integration Developer).

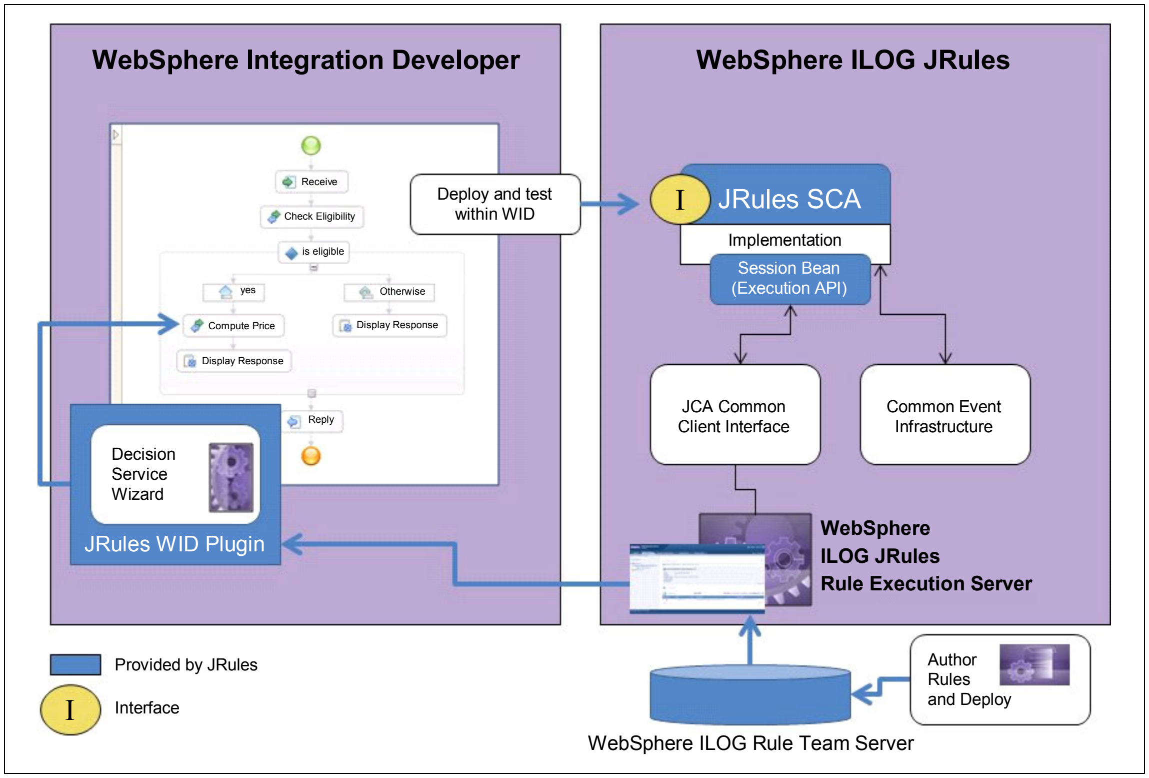


Figure 7-29 Using JRules through Decision Service Wizard plug-in from LA71

Choosing JRules integration options

Benefits and limitations exist in all choices. This section summarizes the benefits and limitations.

Benefits of using Webservices through JRules HTDS

The benefits are as follows:

- Web services are generated directly from JRules tools, which can be imported directly into WebSphere Integration Developer.
- WebSphere Integration Developer provides native support to import JRules web services.
- No additional integration code is required.
- Industry standard interfaces are used for integration.
- Offers loose coupling of process and decision services.
- WebSphere Process Server and RES can be located on the same system or on separate systems.

Limitations of using web services through JRules HTDS

For more complex processes with multiple decisions, HTDS in JRules v7 has a limitation in its use of namespace, and multiple decisions are not possible within the main process flow. However, this limitation can easily be circumvented by implementing the decision service as a microflow in WebSphere Integration Developer which can be invoked from the main process activity flow.

Benefits of using LA71 with Remote EJB JRules interface

The benefits are as follows:

- The wizard is simple to use for generating the SCA modules that are required for the decisions.
- The remote EJB is useful for production scenarios where WebSphere Process Server and RES will not be located on the same system.
- No limitations on multiple-decision use exist within a process such as web services and HTDS.

Limitations of using LA71 with Remote EJB JRules interface

Binding hard-codes a unique RES address. Code must be customized for deploying the SCA module to other environments.

Benefits of using LA71 with POJO JRules interface

The benefits are as follows:

- The wizard is simple to use for generating the SCA Modules required for the decisions.
- POJO is easy to use for simple installation scenarios.
- Supports the Java Object Model.
- No limitations on multiple-decision use exist within a process such as web services and HTDS.

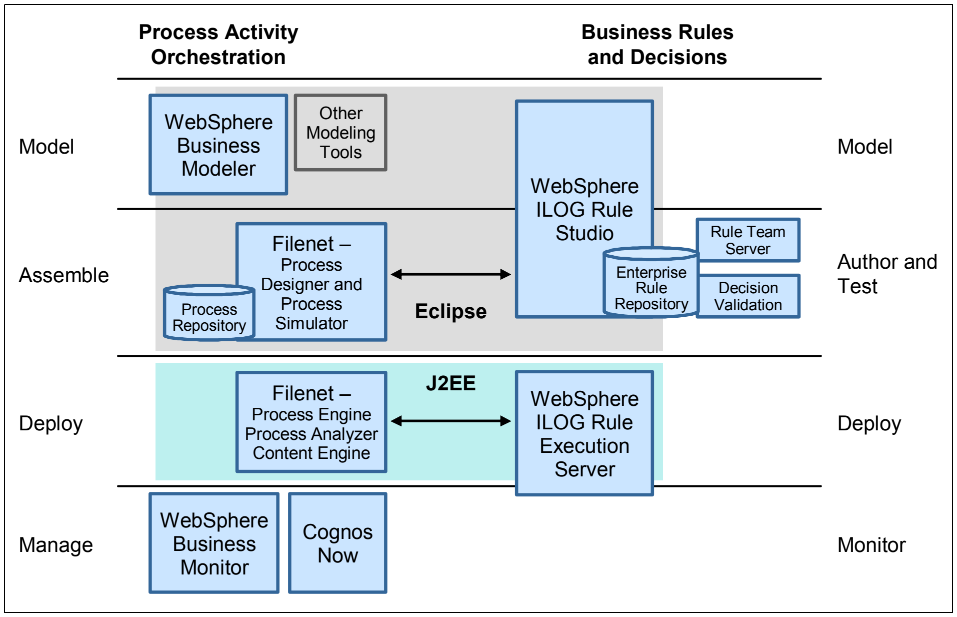
Limitations of using LA71 with POJO JRules interface

POJO is not transactional like EJB is.

7.4.2 Integration with IBM FileNet Business Process Manager

IBM FileNet Business Process Manager (Figure 7-30) manages workflow among people and systems for content and case-based processes:

- It helps organizations increase process performance, reduce cycle times, and improve productivity and decision-making by addressing the creation, management, and optimization of case-based processes to improve business outcomes.
- It provides comprehensive content-centric process management capabilities through included process design and simulation tools, smart electronic forms, rapid application development frameworks, and process monitoring dashboards.
- It integrates with records management and content management, including Content Manager Enterprise Edition and Content Manager OnDemand.



Process Activity Orchestration
Business Rules and Decisions
Model
Assemble
Deploy
Manage
Model
Author and Test
Deploy
Monitor
WebSphere Business Modeler
Other Modeling Tools
WebSphere ILOG Rule Studio
Rule Team Server
Enterprise Rule Repository
Decision Validation
Filenet – Process Designer and Process Simulator
Process Repository
Eclipse
J2EE
Filenet – Process Engine Process Analyzer Content Engine
WebSphere ILOG Rule Execution Server
WebSphere Business Monitor
Cognos Now

Figure 7-30 FileNet diagram

The modeling and managing life cycle tools are more focused on business use than IT, and are used to minimize the gap between business requirements and IT design, implementation, and monitoring real performance.

On the business rules and decisions side, Rules Studio uniquely models the required decision interfaces for the process as an IT role. Authoring, changing, and testing business rules can be done by both IT and business roles.

BPM assembly tools normally require the decision services to be implemented and deployed to WebSphere ILOG Rule Execution Server prior to them being assembled into the decision flow.

At run time, processes deployed to FileNet invoke decision services in Rule Execution Server.

External tools for creating enterprise business objects, which both BPM and BRMS can share, is optional, as is external monitoring of business rule performance.

This combination supports all three solution patterns in 7.1.3, “Solution patterns using BPM and BRMS” on page 99 and all decision patterns described in 7.2, “Decision patterns with processes” on page 111.

Solution pattern 1

IT uses BRMS to complement BPM with rule authoring and deployment. This pattern is sometimes used as the entry pattern to evaluate the combination of BPM and BRMS. The required combination of product components is as follows:

- FileNet Process Designer and Process Simulator are required in combination with the Process Engine and Process Analyzer for the minimum BPM components.
- WebSphere Business Modeler and Cognos® Now or WebSphere Business Monitor can be used by business and IT users and can be present in the pattern.
- WebSphere ILOG JRules Rule Studio and Rule Execution Server are required.

Solution pattern 2

BRMS is used to improve business collaboration and agility. This pattern is the most common, where both IT benefits and business collaboration and agility benefits can be achieved. Components are as follows:

- Similar product components as in pattern 1.
- WebSphere ILOG Team Server for business collaboration to view and change business rule in a central repository.
- In addition, the add-on solutions for Team Server are as follows:
  - WebSphere ILOG Rule Solutions for Office when business users used Microsoft Office 2007 for capturing business rules.
  - WebSphere ILOG Decision Validation Services for simulation, test and what-if scenarios.

Solution pattern 3

This pattern uses central authoring and management of business rules. It is often used after multiple joint BPM/BRMS projects or where a BRMS solution is already established and is extended to centrally manage decisions for processes that are orchestrated in BPM and other applications. This pattern builds on the deployments from the other patterns and increases the usage of them without adding any new product components.

The modeling and managing life cycle tools are more focused on business user use than IT. This pattern is used to minimize the gap between business requirements and IT design, implementation, and monitoring real performance.

On the business rules and decisions side, Rules Studio uniquely models the required decision interfaces for the process as an IT role. Authoring, changing, and testing business rules can be done by both IT and business roles.

BPM assembly tools normally require the decision services to be implemented and deployed to WebSphere ILOG Rule Execution Server prior to them being assembled into the decision flow.

At run time, processes deployed to FileNet invoke decision services in Rule Execution Server.

External tools for creating enterprise business objects which both BPM and BRMS can share is optional, as is external monitoring of business rule performance.

Modeling considerations

Process modeling is implemented in the business; process models that are captured in tools such as WebSphere Business Modeler can be exported to FileNet.

Business Modeler documents the process activity flow for processes that use industry standards like BPMN and helps to more easily identify which activity steps are business decisions.

For decisions that are implemented in JRules, a general activity is modeled and it can be associated with a JRules decision during assembly phase. This method allows JRules business rules tools be to be used to model the decision interface in JRules Rule Studio and author the business rules in natural language and prior to assembly in JRules Rule Studio, Rule Team Server.

Business object models

WebSphere Business Modeler can create some simple object models. However, for most projects with some levels of complexity we recommend using other tools to create the business objects and then import back into WebSphere Business Modeler.

Designing agility into processes which make decisions

FileNet Business Process Manager supports subprocesses if a process wants to encapsulate invocations to specific decision services. It can invoke a service mediation, running in an external ESB solution that can encapsulate a decision service.

Assembling and deploying considerations

The assembly tool of choice for solutions with FileNet is FileNet Process Designer. This tool can import a process model from a modeling tool such as WebSphere Business Modeler when exported using theXML Process Definition Language (XPDL) standards for process assembly and run time. The process flow can be created directly into Process Designer if no modeling stage was done.

With the decision points and decision types previously identified, the role of FileNet Process Designer to select and bind JRules decisions into the process activity flows at the appropriate points.

Decision implementation choices

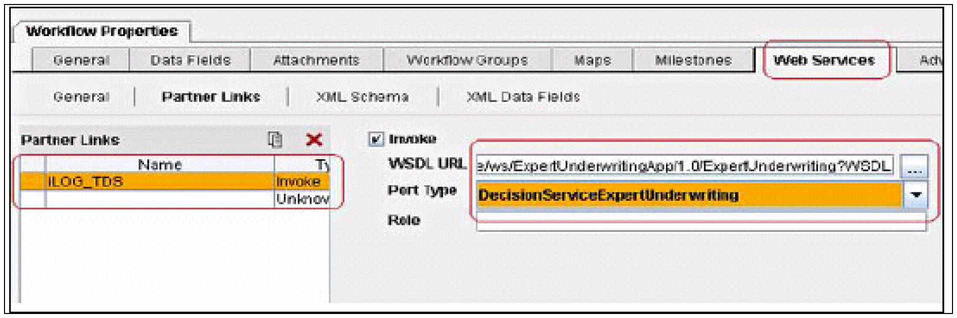
Depending on the decision type, complexity, and other factors a choice is sometimes required regarding how the decision is implemented: either in BPM or in JRules. This criterion is provided in 7.3.2, “Assembly considerations: JRules decisions or BPM decisions” on page 122.

Available integration decision binding options

JRules offers only one commonly used binding option for FileNet projects: web services using XML data schema. Custom integration if other JRules interfaces can be achieved by using custom developed services to the rule session interfaces.

Web services

JRules decisions may be deployed to WebSphere ILOG JRules Rule Execution Server (RES), where they may be reused by any standard SOAP client. Using default features in FileNet Process Designer, a developer can easily import a WSDL from RES and use it within a business process after configuring the partner link to point to RES. See Figure 7-31 on page 149.



Partner Links

Name	Ty
ILOG_TDS	Invoke
	Unknov

Invoke: ✓

WSDL URL: e/ws/ExpertUnderwritingApp/1.0/ExpertUnderwriting?WSDL

Port Type: DecisionServiceExpertUnderwriting

Role

Figure 7-31 Using web services to integrate rules with FileNet

WSDL is provided by HTDS. HTDS is an enterprise application providing the service endpoint within RES. It acts as a service wrapper to the JRules session bean and facilitates the binding to SOAP

The WSDL import process can generate many of the required WebSphere Integration Developer project artifacts, including the interface and business objects (datatypes). JRules provides two options for WSDL:

- Get HTDS WSDL for this rule set version

  This option provides the WSDL for a specific version of the decision. Use this version if you do not want the decision to change for existing long-running process instances.

- Get HTDS WSDL for the latest rule set version

  This option always provides the latest version of a decision. All existing and new process instances will take advantage of dynamic changes to the decision.

7.4.3 Integration with WebSphere Lombardi Edition

WebSphere Lombardi Edition provides a unified BPM environment for collaborative process improvement. It can help process owners and business users, and IT to more easily collaborate and engage directly in improving their business processes.

It is well suited to implement business processes exhibiting a high degree of user involvement and frequent change. It has a single, unified environment for process design, execution, monitoring, and optimization and graphical workspace for rapid composition and collaboration. See Figure 7-32 on page 150.

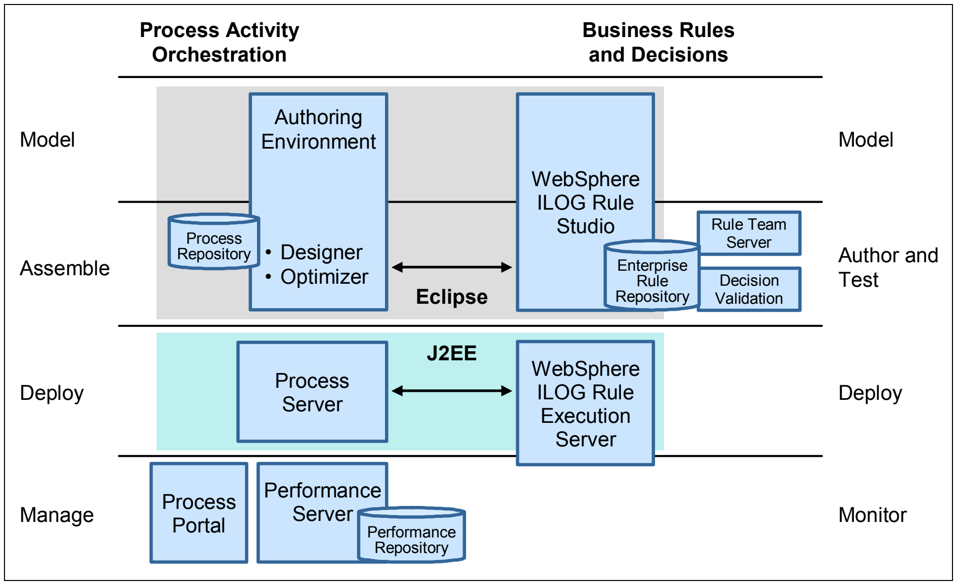


Figure 7-32 Joint Solution tools and runtime components

The modeling, assembly, and managing life cycle tools are more focused on the business user than IT in the Lombardi tooling, which differs in both WebSphere Process Server and FileNet. This tool is used to minimize the gap between business requirements and IT design and enable business process owners in business teams to rapidly change and redeploy the process.

On the business rules and decisions side, Rules Studio uniquely models the required decision interfaces for the process as an IT role; however, authoring, changing, and testing business rules can be done by both IT and business roles. Because the Lombardi Edition tooling has a more business focus than having a matching business focus with JRules, including Rule Team Server is expected to be part of any joint solution to author and change rules more.

Lombardi Edition authoring tools require the decision services to be implemented and deployed to WebSphere ILOG Rule Execution Server so that the WSDL files are created for the decision services prior to them being imported and assembled into the decision flow.

At run time, processes that are deployed to WebSphere Lombardi Edition Process Server invoke decision services in JRules Rule Execution Server.

This combination supports all three solution patterns and all decision patterns that are described in 7.4.2, “Integration with IBM FileNet Business Process Manager” on page 145.

However, because Lombardi Edition can enable and make process orchestration easy for business users, patterns 2 and 3 are normally used.

Solution pattern 2

BRMS is used to improve business collaboration and agility. This pattern is the most common where both IT benefits and business collaboration and agility benefits can be achieved:

- Lombardi Edition is a combined solution with multiple components.
- WebSphere ILOG JRules Rule Studio and Rule Execution Server are required.
- WebSphere ILOG Team Server is required for business collaboration to view and change business rule in a central repository.
- In addition, the add-on solutions for Team Server are as follows:
  - WebSphere ILOG Rule Solutions for Office when business users used Microsoft Office 2007 for capturing business rules.
  - WebSphere ILOG Decision Validation Services for simulation, test, and what-If scenarios.

Solution pattern 3

The central authoring and management of business rules pattern is often used after multiple joint BPM/BRMS projects or where a BRMS solution is already established and is extended to centrally manage decisions for processes that are orchestrated in BPM and other applications. Note the following information:

- This pattern builds on the deployments from the previous patterns and increases the usage of them without any new product components.
- WebSphere Lombardi Edition has multiple components that are required to implement the process life cycle that has been identified.
- Because the BPM and BRMS solutions work well together it would also be useful to position JRules and its multi phase life cycle at the same time.

Modeling and assembling considerations

WebSphere Lombardi Edition Authoring Environment supports BPMN and allows a process to be modeled directly. It also has a separate tool named Blueprint which can support early modeling phases.

This tool can simplify the modeling and assembly phase prior to deployment, and uses industry standards such as BPMN for modeling the activity flow. This tool can help you to more easily identify which activity steps are business decisions.

When a decision point occurs in a process, the designation of the task is the same at that decision point. See Figure 7-33.

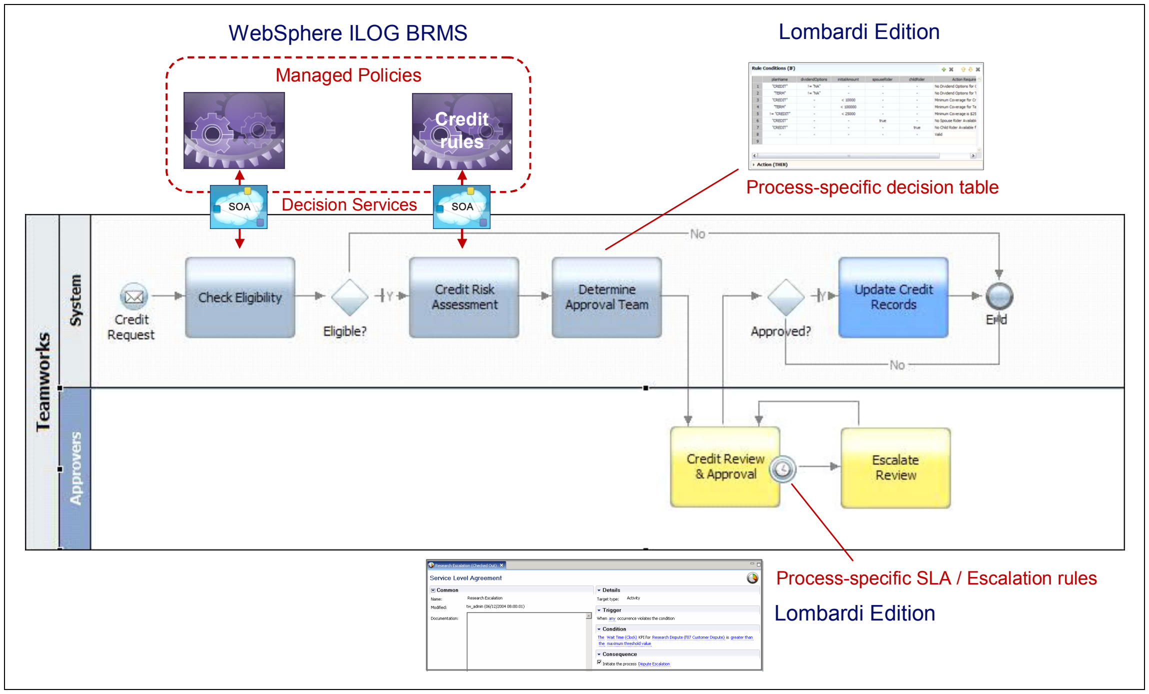


Figure 7-33   Example process using both JRules decisions and internal rules

Decision implementation choices

Depending on the decision type, complexity and other factors a choice is required as to how the decision is implemented. This criterion is provided in Section 7.3.2, “Assembly considerations: JRules decisions or BPM decisions” on page 122.

Figure 7-33 demonstrates that some SLA rules and human task assignment rules can be implemented within the process context and other Eligibility and credit rules can be invoked from JRules.

Business object models

Lombardi Edition has the capability to create process object models in XML format. The normal assumption is that Lombardi tooling is used for the process model. As such, a decision is required as to whether the decisions in JRules used the Model generated from Lombardi Edition or from an external tool.

Designing agility into processes which make decisions

WebSphere Lombardi Edition supports modeling a process to encapsulate invocations to specific decision services in a subprocess as its standard integration practice for all external services. This pattern also abstracts any technology or service endpoint specifics from the generic process model.

It can call a service mediation running on a separate ESB which encapsulates a decision service if required.

Available decision integration and binding options

Lombardi Edition does not recommend any direct invocation patterns for decisions. Instead, it recommends an indirect invocation, using subprocess as default, as shown in Figure 7-34.

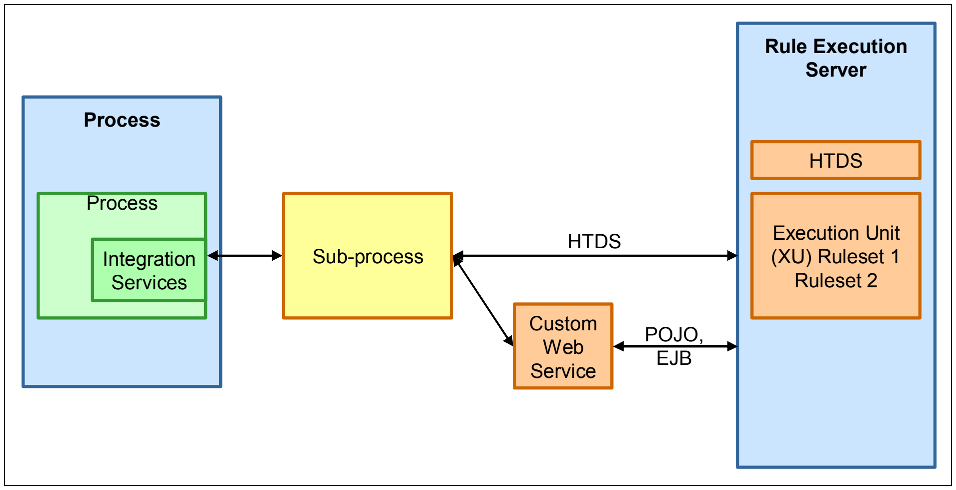


Figure 7-34 Lombardi Edition invoking business decisions from a subprocess

To keep the process activity streamlined and simple, a good design practice is to encapsulate service invocations to external systems. See Figure 7-35.

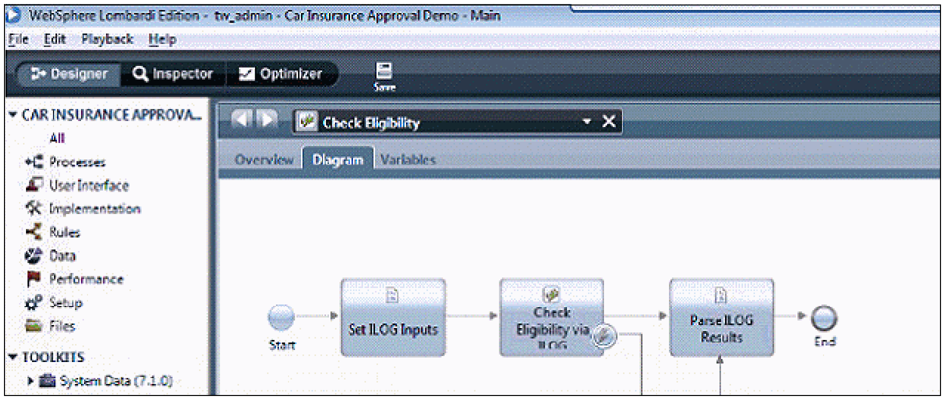


Figure 7-35 Encapsulating a JRules eligibility decision in Lombardi Edition

JRules offers two binding options for Lombardi projects that can be invoked as Lombardi Edition integration services (Figure 7-36):

- Web services
- Integration with JRules interfaces

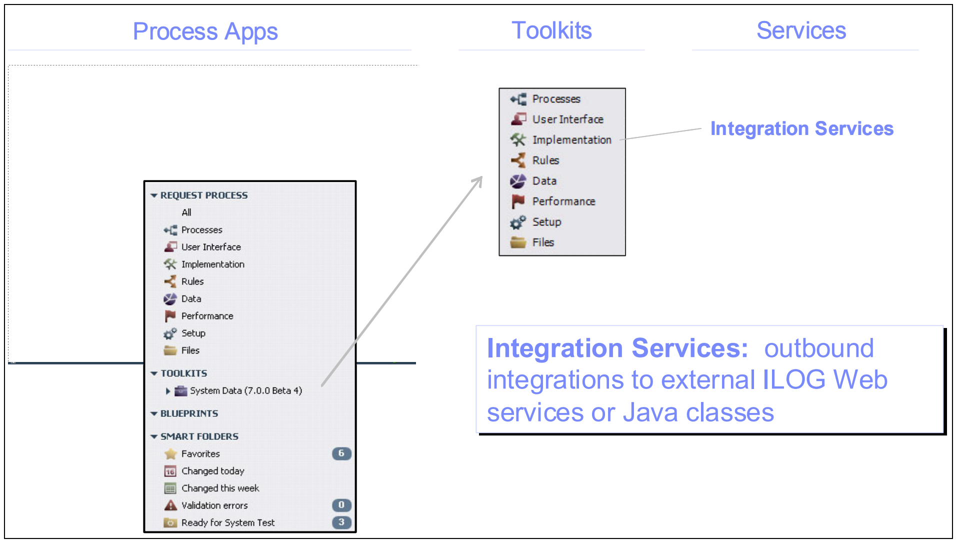


Figure 7-36 Invoke JRules decision services as Integration services

Web services

JRules decisions may be deployed to WebSphere ILOG JRules RES, where they may be reused by any standard SOAP client. Using default features in Lombardi Edition Authoring Environment, a developer may easily import a WSDL from RES and use it within a business process.

WSDL is provided by hosted transparent decision services (HTDS) in JRules. HTDS is an enterprise application providing the service end-point within RES. It acts as a service wrapper to the JRules Session Bean and facilitates the binding to SOAP.

Custom web services which invoke decisions in JRules can be integrated in a similar way.

The WSDL import process generates many of the required WebSphere Integration Developer project artifacts including the interface and business objects (data types). JRules provides two options for WSDL:

- Get HTDS WSDL for this rule set version

  This option provides the WSDL for a specific version of the decision. Use this version if you do not want the decision to change for existing long-running process instances.

- Get HTDS WSDL for the latest rule set version

  This option always provides the latest version of a decision. All existing and new process instances will take advantage of dynamic changes to the decision.

Custom integration using POJO JRules interface

The Lombardi integration service can be implemented to access POJO invocation if you want the RES to run in-process with the process instance.

This manual process requires the JAR files to be deployed to support the solution, which is additional coding effort. Using POJO is probably more applicable when JRules and Lombardi Edition are running on the same system. In summary, where JRules is used on separate systems for larger production scenarios, web services is expected to be the normal choice.

8

Connectivity infrastructure and messaging

This chapter contains the following topics:

- Rules within the connectivity infrastructure
- Decision patterns
- Integration pattern considerations
- Integration patterns in IBM products

8.1 Rules within the connectivity infrastructure

In this section, we look at rules within the connectivity infrastructure

8.1.1 Introduction

Connectivity layer (or connectivity infrastructure) middleware components are used to handle the complexities of application connectivity or integration. These components can provide services ranging from feature-rich enterprise service buses (ESB) and message brokers to basic messaging services. This chapter focuses on ESB products within the connectivity layer (for example, WebSphere Message Broker, WebSphere Enterprise Service Bus, and the WebSphere DataPower® Integration Appliance) that can integrate with WebSphere ILOG JRules and use its business rules. We describe typical scenarios, patterns, and considerations.

The question of where business rules (sometimes called *business logic*, in ESB terminology) should be implemented is always an important discussion during solution architecture and design. Many enterprises use an ESB pattern within their architecture to support the requirement for either traditional Enterprise Application Integration (EAI) or a service-oriented architecture (SOA). In early solutions that used an ESB, a common approach was to find that business rules became embedded within the ESB message flows (also called *mediation flows* and *message processing flows*). The reason is because the message flows deliver the integration between separate applications within the enterprise and is therefore an "easy" and identifiable place at which to make business decisions. However, over time this became accepted as bad practice because embedding such logic within the ESB connectivity layer can inhibit flexibility in the overall architecture. If such embedded business rules needed to be updated it would require the message flows to be updated, tested, and redeployed, which is a potentially lengthy process even for minor changes. Because these systems are often responsible for handling high volumes of potentially business critical messages flowing through the enterprise, then this represents a high risk IT change.

By using a combination of an ESB and a WebSphere ILOG JRules, decoupling the integration or connectivity logic from any necessary business rules is now possible. This way means that the foundation of the connectivity layer can remain unchanged; business rules are changed in a fast and flexible manner, which lowers risk and increases business agility.

Figure 8-1illustrates ESB rule processing.

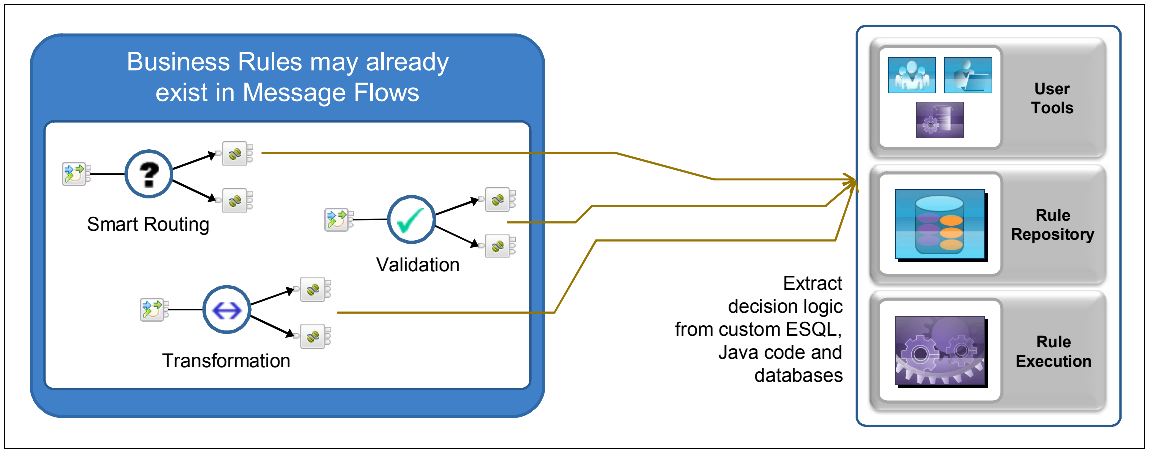

Business Rules may already exist in Message Flows
Smart Routing
Validation
Transformation
Extract decision logic from custom ESQL, Java code and databases
User Tools
Rule Repository
Rule Execution

Figure 8-1 ESB rule processing overview

8.1.2 Connectivity and business rules life cycles

The life cycle for the message processing flows that are deployed within our connectivity infrastructure are naturally under the control of the IT department (for example, integration specialists). By using business rules appropriately, we can separate the connectivity processing flow life cycle from the business rule life cycle. This separation means that when a business policy changes, the rule modifications that are made in support of that business change can be redeployed independently of the message processing flows. The result is the gaining of similar benefits to other rule based applications, for instance, an increase in flexibility and the ability to adapt as quickly as possible to certain changes in policy at the business level.

When exploiting WebSphere ILOG JRules in your connectivity infrastructure, the business analyst can make changes to the rules and test the changes, thus removing the requirement on the IT department to make code changes. The results are as follows:

- Reduced workload for the IT department
- Reduced need for the business analyst to communicate the required changes

  Such a change has an inherent risk that the changes might be misinterpreted, or made incorrectly.
- Increase in flexibility, for example, reduced dependencies on the IT department

- The ability to adapt as quickly as possible to certain changes in policy at the business level
- Increased effectiveness of the business analyst.

  The reason is that business rules can be iteratively prototyped and tested before deployment by the business analyst and without the need to involve the IT department.
- Reduced rule testing costs (time, software, infrastructure)

  The reason is because the run time that is used in rule prototyping and testing is identical to the deployment run time.

8.1.3 General design considerations

Consider the high-level architecture or design when you start a project that involves integration of an ESB with a BRMS.

When to use BRMS

Depending on existing or proposed ESB business logic type, complexity and other factors, choose how the decision is implemented. Table 8-1 describes best practices for each choice.

Table 8-1 BRMS best practices

When to use JRules	When to use ESB
The rule management life cycle must be independent of the ESB development life cycle (implies that the frequency of change differs between a decision and the ESB flows).	The rules belong to the life cycle of the message processing flow.
Rules are authored and maintained by business users.	Rules are authored and maintained by the same user that owns the message processing flows.
Decisions contain many rules.	Decisions contain few rules.
Decisions require independent testing and simulation by business users.	Decisions are tested and simulated at the same time as the message processing flows.
Decisions require inference or contain complex data structures.	Data used for decisions is limited to routing and basic evaluation.

ESB and BRMS data models

The data model of the enterprise architecture (EA) often includes messages processed within the connectivity layer. This model is often unlikely to match the BRMS data model. The differing concerns and the historical lack of separation of the data models may have resulted in inconsistent models. When designing a data model for an enterprise, consider the separation of business and connectivity data. In addition, understanding what information should be exposed, and modeled, for use at the BRMS level is an important facet introduced with BRMS.

The requirements for data models between the ESB and BRMS environments might differ. The ESB processes messages that flow between applications and other components. As such, these messages are likely to conform to data models that are owned at the EA level or line of business level. In addition, these messages might also contain infrastructure data that might be owned by the IT department.

The BRMS data model itself and the data to be exchanged between the ESB and BRMS must be designed in a manner which is consistent and reasonable given the existence of the other EA models. The BRMS data model must not be designed in isolation, because this might inhibit exploitation within the connectivity layer. Where existing rules, perhaps developed in isolation from the existing connectivity infrastructure implementation, are used by ESB then mismatches in data models might become apparent, which would then require some re-examination of the rule interfaces and models. Simply assuming that the ESB message flow implementation can adapt to the existing rule is not good practice.

When the data model of an enterprise's business rules does not match that of the connectivity layer, messages flowing through the ESB will likely contain much more information than required by the rule to be invoked. The results are the following options when passing data from the ESB to the BRMS (and vice versa):

- Pass all data

  This option reduces data model development concerns and simplifies message flow development, although it can cause performance issues if the BRMS is used to process very large messages.

- Pass a subset of data (only the data required by the BRMS)

  Although this option performs better, the data model transformation must be considered and that message flow development is more complex.

Message flow and rule development collaboration

Many integrations between the ESB and BRMS use existing assets. For example, a new version of a message flow is developed to use an existing rule set, or a new rule set is developed to replace an existing service that is used by a message flow. Where new message flows and rule sets are developed to form a new solution, collaboration between ESB and BRMS developers is important so that interfaces are designed to fit with each development team's design guideline development cycles and practices.

Contract between the ESB and BRMS

Changing the rule set signature can be costly. You must clearly define the contract between the ESB and BRMS before any other consideration or integration.

A good practice is to validate the contract using Decision Validation Services module, which can guarantee that the contract is respected. The Decision Validation Services can be used either directly in Rule Team Server, from Eclipse, or in batch mode (samples are delivered in the WebSphere ILOG JRules distribution for more information).

8.2 Decision patterns

This section describes decision patterns applicable for use with connectivity infrastructure and messaging products and goes into greater detail than the generic BRMS decision patterns that are introduced in Chapter 7, “Business processes” on page 95.

In the connectivity layer, we must often process, transform, and route messages with minimal overhead and latencies. Integration of WebSphere ILOG JRules within the connectivity infrastructure is therefore more likely to be suited to scenarios in which the rules are considered to be fine-grained or at the micro-decision level. This section describes high value patterns for the integration of the ESB and WebSphere ILOG JRules.

8.2.1 Message routing

Using WebSphere ILOG JRules to influence message flow routing decisions enables the business users to directly participate in the authoring and change management that is associated with routing decisions. Frequently, the choice of destination can have business implications in terms of the cost of processing or the ability to maintain service level agreements. Changes to these routing rules might be required to keep the business agile and competitive in the marketplace.

For example, if we have a customer of class “Gold,” we provide express shipping of their order. Rules that are based in WebSphere ILOG JRules can be used to determine the class of the customer and the service that should be used for shipping.

The rule has to return some data in the output that will form the basis of the routing decision. Do not return technical infrastructure data because it creates an unwanted close coupling between the connectivity layer and the rule, and exposes the business analyst to infrastructure considerations.

The most likely approach here is for a use case that involves static routing between a known set of endpoints (or routing targets). The ESB and the rule implementations must agree on the defined set of targets. To effect the routing, the ESB implements a mapping of a name that has business meaning and the infrastructure endpoint that represents it.

One of the many features of an ESB is its ability to route messages between services or application components that are based on certain criteria, such as the content of either metadata (for example, header information) or the business content (for example, the payload). Often, the choices for where a message should be routed are founded on business-related factors. For example, the choice of target service or application might have business implications in terms of the cost of processing or the ability to maintain service-level agreements (SLAs). Being able to make changes to these rules quickly, outside the IT life cycles, is highly attractive.

Figure 8-2 illustrates ILOG JRules message pattern routing.

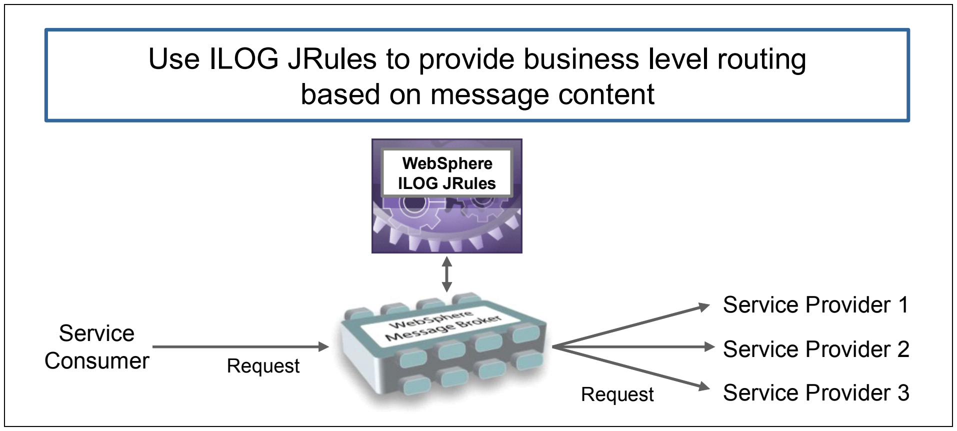


Figure 8-2 Message routing pattern

Scenario

A large electronics outlet can receive orders from customers anywhere in North America. When an order is placed and a payment received, a message is sent to the ESB, which routes the request to a fulfillment partner. The wholesale cost charged by each partner can change frequently. In addition, the shipment cost is a factor of the distance between the partner and the customer. Therefore, a desirable approach is to route the order to the most cost-effective partner, which maximizes their profit.

To deliver this solution, the ESB message flows are developed and the supplier selection rule is modeled in WebSphere ILOG JRules. At run time, the order request and the appropriate parts of the order request are used as input to the rule. In its output, the rule returns an indication of the partner to be used and the ESB acts on this information to perform the routing.

Design considerations

The key objective is to decouple the connectivity layer from the business rules so the rule is unaware of any data or concept which is in the infrastructure domain. The rule must deal only with abstract or real business concepts and not IT related concepts. For example, for the rule to return the name of an MQ queue to which the message should be routed is not advisable; this is clearly knowledge about deployed IT infrastructure and that can change in various environments. It is not something a business user would be expected to be familiar with. Instead, the rule should return *abstract* routing information, such as partner identification. The ESB message flow can then perform the necessary mapping to the target endpoint.

The consideration implies that the ESB message flow must perform a mapping between the “routing target” that is returned by the rule and the physical endpoint. At a simplest form, this way could be a static mapping implemented within the message flow logic. However, this method would require that the message flow to be changed when a new endpoint is added or removed. A more flexible and agile solution can be considered:

- Use WebSphere ILOG JRules to define the connectivity infrastructure routing mapping (possibly implemented by a Decision Table). This rule can be owned and modified by IT and not the business.
- If an SOA registry such as IBM WebSphere Service Registry and Repository, is being used in the architecture, the message flow can perform a lookup of the registry at run time to select the endpoint based on the output of the rule.

The message flow should be designed to cope with an unexpected routing target being returned from the rule. The flow should handle invalid routing labels accordingly, for example, with error handling or default routing logic.

8.2.2 Message enrichment

Another feature of an ESB is the ability to enrich messages with additional information to provide additional data, which cannot be supplied by the client application and which can only be obtained by the ESB interacting with other services, applications, data, or “business logic.” This approach is known as *message enrichment* or *message augmentation*. Several of these situations might be appropriate for exploitation of a BRMS, as follows:

- When an existing enrichment implemented is in your connectivity layer and is derived from what can be described as “business logic” implemented within the ESB flow itself. You might recognize this example when you find you frequently need to modify the same logic in the processing in response to business policy changes.
- When the ESB interacts with an external data source that acts as a type of rule (for example, you might use a database table representing a set of outcomes or outputs for a set of given inputs). This example can be regarded as analogous to a WebSphere ILOG JRules decision table. Again, if this data source must be changed frequently, this situation might represent where BRMS can be exploited.

In these cases, you may find that the change you made to your ESB business logic is often minor, but has a significant amount of impact in terms of testing and operational deployment,

Information might often be required in the message for the final consumer and cannot be supplied by the client. This information might be derived from the supplied message using business rules. For example, an element of the final price might have to be calculated individually for each customer based on demographics, as in the following example:

```
Customer Location = Iowa, therefore tax = X%
```

Messages being processed within the connectivity infrastructure can be enriched with data, which is output from a rule which is invoked from the message flow. The rule input is derived from either the message (or part of it) being processed by the message flow. The rule output is used to update the message as it passes through the message flow. For example, we can use some data or context within the message to look up in a decision table to determine the tax rate to be applied to an insurance premium, based on the country where the insured party resides.

Although message enrichment does perform message modification, this is not regarded as the same as the use case for transformation (or mapping) of the message. Enrichment is where the message formats are the same but modification of a subset of the fields in the message is needed, as shown in Figure 8-3 on page 166.

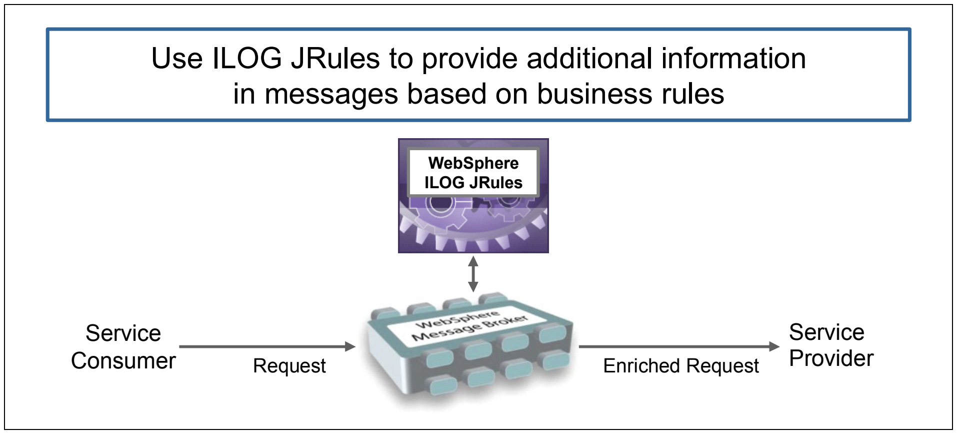


Figure 8-3 Message enrichment pattern

Scenario

An insurance company can receive requests for motor insurance quotes, through phone or web, from customers anywhere in North America. The company uses two ESBs, as follows:

- A web channel ESB for processing quotes submitted directly by a web customer
- A call center ESB for processing quotes submitted by call center employees

When a quote is submitted (by a customer or call center employee), a message containing information about the vehicle and driver is processed by the message flows of the ESB. The message flows adjust parts of the quote submission (for example, vehicle value) based on rules that take current vehicle valuations and historical insurance claims into consideration. When the values are adjusted, the quote submission is passed to backend quote systems, which calculate the premium for the quote. The quote submission rules and the data on which they are based can change frequently. The best approach is to use the most up to date rules and data to avoid costly mis-valuations. In addition, the rules are required by both ESBs and therefore a centralization of rules helps ensure that the customer quotes match regardless of whether they use the web or phone (call center) channel.

To deliver this solution, the ESB message flows are developed and the adjustment rules are modeled in JRules and deployed in a centralized rule engine. At run time, appropriate parts of the quote submission are used as input to the rule set. The rule set returns in its output a quote submission containing adjusted values, which the ESB then passes to backend quote systems, which calculate the premium.

Design considerations

The design considerations are as follows:

- Enrichment scenarios typically use identical message formats for input and output. Although this use often simplifies the development of message flows and rule sets, it can be appropriate for the message formats for the input and output to the rule engine to differ. In cases where they differ, the ESB message flow can be used to transform the output from the rule.
- Rules that are used to enrich messages must be self-sufficient and not connect to another application, such as a database. If orchestration between separate systems and processes is required, consider integration with a BPM application.
- Invalid rule set output parameter content can cause errors in subsequent system components. This issue usually indicates that the schema is not rigid enough. When deployed schemas are limited and inflexible to change, you consider validating the output from the rule set. For example, the rule set output can contain an enumeration type that conforms to the schema, but contains values that are invalid for a subsequent part of the message flow.

8.2.3 Intelligent data mapping

One of the roles of the connectivity layer is to provide features for transformation of the structure of messages. ESB message flows generally can transform messages at a structural and content level by using various mapping tools and features. It is more challenging where there is no direct mapping for the *content* between the source and target of the message transformation. In such cases, a level of business knowledge is sometimes needed to complete the required mapping. It might not be easy for the business to specify requirements to IT for how the mapping is to be implemented. These factors suggest that a business rules-based approach to the fine-grained mapping detail can be considered. With this approach, IT can develop and build the connectivity infrastructure independently of the rules modeling and refinement,

Certain mapping requirements may therefore be best satisfied by using a combination of appropriate mapping features in both the ESB and the BRMS, as shown in Figure 8-4 on page 168.

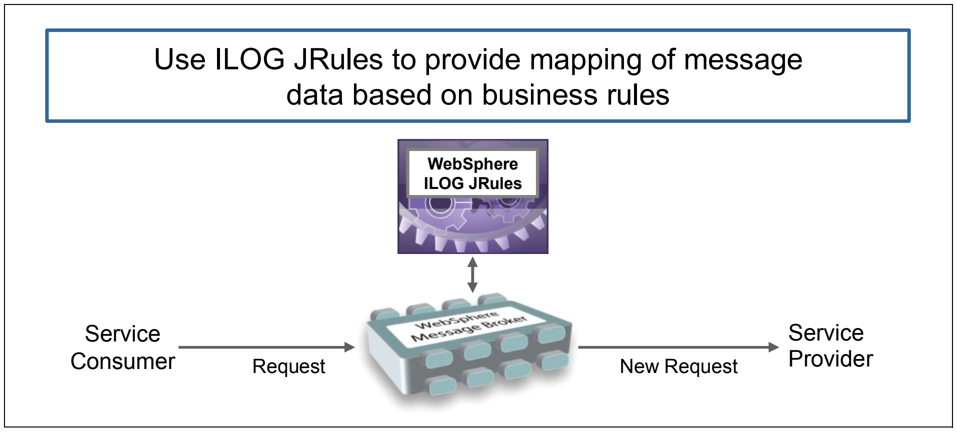


Figure 8-4 Intelligent data mapping pattern

Scenario

A company that operates a network of gyms uses a pricing system which uses an ESB for routing and transformation of pricing information. The pricing information is delivered from the head office through the ESB to the gym locations. The ESB then maps a general pricing message format into gym pricing message formats, the content of which can be tailored for individual gyms. There are multiple gym pricing messages formats, to cover the various gym services, such as membership fees, individual visits, equipment hire, and so on. Special offers such as loyalty programs and discounts are also available. General pricing information is mapped to gym pricing based on business factors.

The gyms experience periods of peak demand and want to spread the demand through each day and year by offering timely special offers. The ability to deliver timely special offers results is increased customer satisfaction, because of less queues for equipment and the actual discount benefit and a reduction in staff costs and rota pressures. The special offers are created by business analysts based on business costs, geographical information, and time factors such as day of the week and time of year.

To deliver this solution, the ESB message flows are developed and the special offers rules are modeled in WebSphere ILOG JRules. At run time, the general pricing input messages are used as input to the BRMS. The BRMS outputs appropriate messages for gym pricing and special offers which the ESB then passes to the gyms.

Design considerations

The design considerations are as follows:

- Mapping can be implemented in either the ESB, the BRMS, or a combination of the two, depending on considerations such as the requirement for frequency of rule update and whether the rule set relates to business or IT-related factors.
- Invalid rule set output parameter content can cause errors in subsequent system components. Usually this consideration indicates that the schema is not rigid enough. Where deployed schemas are limited and inflexible to change, consider validating the output from the rule set. For example, the rule set output might contain an enumeration type that conforms to the schema, but contains values that are invalid for a subsequent part of the message flow.

8.2.4 Business level message validation

Business rules can augment the standard ESB processing to provide a business context to the message validation. The two aspects to message validation are as follows:

- Ensure that messages are structurally correct.

  For example, are XML tags in the correct position? Are any constraints such as datatypes respected? A role of the message flow is to perform such validation by using techniques such as XML schema validation, which rejects messages that are incorrectly formed. This aspect generally requires no "logic" or "rule" to be implemented within the connectivity layer; instead the message flow is generally configured to validate input, output, or both.

  This type of message validation is best performed within the ESB infrastructure or message flows themselves where native support for schema, or other message model validation exists. The WebSphere ILOG JRules also carry out schema validation of rule input parameters.

- Ensure that the content of data is correct and valid from a business perspective.

  This consideration goes further than the content constraints, mentioned previously, to support the validation of business data in data fields and the whether a data field is valid in relation to other fields in the message. We term this business-level message validation. Avoid implementing such business rules within the ESB flow itself. Instead, consider delegating these validation rules into WebSphere ILOG JRules.

Figure 8-5 shows the ILOG JRules message validation process.

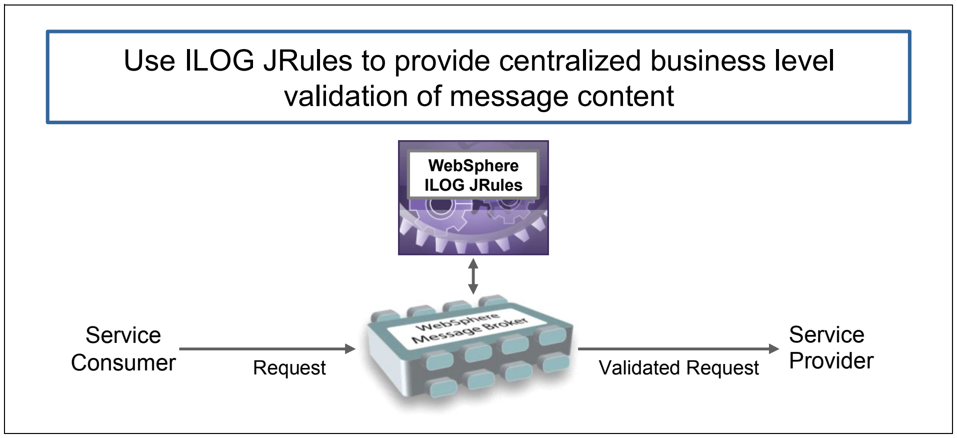


Figure 8-5 Message validation

Scenario

A large insurance company has clients throughout a wide geographical area with a large number of variations in clauses on policies. To reduce costs, the company is centralizing its claim processing centers. Claims are routed from the data centers in each region to a central claims unit. Claim requests are sent to the ESB, which resides in that central claims unit. To reduce the cost of processing invalid claims, the company wants to reject invalid claims at the earliest possible opportunity.

To solve this problem, an appropriate rule set is modeled in WebSphere ILOG JRules, which takes as input the originating claim center location ID and the claim itself, and returning an error output if the claim is invalid for the originating region. The existing ESB message flow is modified to invoke the rule for each message containing a claim and the message is routed back to the originating region.

Design considerations

The design considerations are as follows:

- The decision of whether to apply structural (schema) validation in the ESB to messages is often done at an architectural level. That is, the decision is generally not associated with a specific message flow between two applications. For example, we might use an ESB gateway to validate all messages that originate from outside our “trusted” IT infrastructure, such as those that are received from the Internet or from our trading partners.What we are saying is that the decision to apply validation of messages that are input to, and output from, rules, are generally be guided by a higher architectural principle. An exception to that might be during development or integration testing when we want to catch invalid messages.

Additionally, if we, as the IT department, regard BRMS as not being in our IT “trusted” domain, we might want to strictly perform message validation.

- The validation strategy must be consistent between the connectivity infrastructure and the rule set. If structural and constraint validation against a schema is being done in the ESB, it must not be performed within BRMS. Do not mix validation techniques that are performed in the ESB and BRMS. The most likely strategies are as follows:
  - Use schema validation in the ESB if you have an over-arching architectural principle that requires it.
  - Perform any validation of infrastructure information (headers) within the ESB itself.
  - Delegate any required validation of business information within message content or payload to WebSphere ILOG JRules.
- Regardless of input method, for example, POJO, the WebSphere ILOG JRules rules engine can perform schema validation on rule sets with XML based input parameters. If rule set input parameters are not structured correctly, the rule set execution will fail. Example 8-1 shows several error messages that can occur.

Example 8-1 Rule set execution error message examples

```
ERROR ERR019: in source ?, after line 10, before line ?, Internal
exception: For input string: "demo"
ERROR RTERR011: in source ?, after line 3, before line ?, Unexpected
or unknown XML element: VehicleIdentificationNumber
```

8.2.5 Enable virtualization of BRMS services

An ESB is often chosen to offer the *virtualization* of services in a implementation of an SOA. In this pattern, we observe the notion of exposing, or virtualizing, BRMS services over an ESB. Although this notion is already present in native WebSphere ILOG JRules using web services (hosted transparent decision services) or web services (custom), the ESB can be leveraged to virtualize any type of rule implementation. This way can be used, for example, to expose SOAP web services to application clients within the enterprise, which must invoke rules, regardless of how the rule is implemented. This way can further decouple the application client from the rule implementation and hence simplify the architecture.

WebSphere ILOG JRules Services can be virtualized in exactly the same ways as any other service. The use of service virtualization can be used to perform the following tasks:

- Provide a policy enforcement point decoupled from the BRMS system.

  An example is providing transport security or providing logging for audit purposes. This pattern means that policy related artefacts (message flows) can be reused for multiple services, for example, BRMS, transaction servers, web services, databases, and so on.

- Provide a protocol and data transformation services.

  An example is a J2EE application providing an inflexible rule service through a custom message format over HTTP is replaced by an WebSphere ILOG JRules deployment. Existing rule requestor clients can be used without modification by using an ESB to transform the protocol and data, for example, custom message format over HTTP, to that required by the new Rule Execution Server, such as JMS over WebSphere MQ.

- Hide the true location of rule providers (services) from the rule requestor.
- Allow for the seamless additional or removal of rule providers with no impact on the rule requestors.
- Support architectures where rules are invoked from client application requestors, such as home-grown applications or business processes such as BPEL.

Figure 8-6 shows how ESB can virtualize BRMS services.

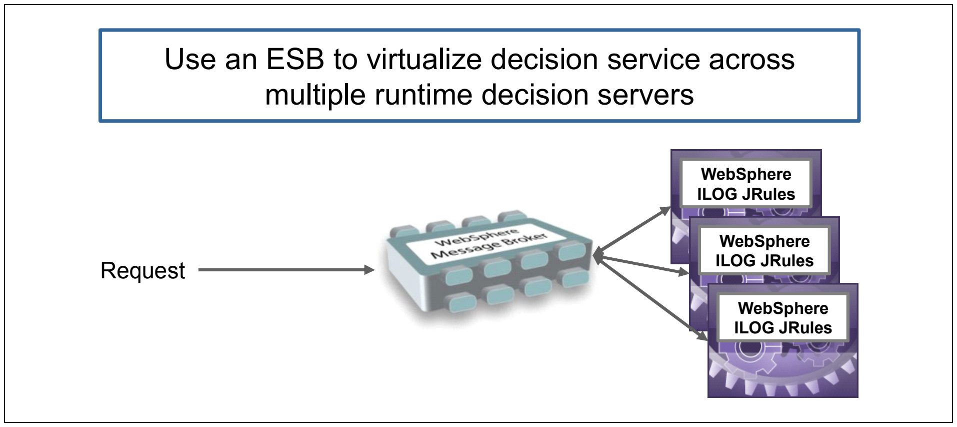


Figure 8-6 Virtualization of BRMS Services

Scenario

A car manufacturer has deployed a BRMS in a data center. The rule services that are provided by the BRMS are used by rule requestors in several external company locations, ranging from manufacturing locations, to head office and regional sales offices. The data center also contains systems that provide other services such as CICS web services for accounting and COBOL applications for client sales services. The car manufacturer introduces a new security policy for all intra-company data transmission. Although many systems (including the BRMS) offer features that adhere to the new security policy, the car company chooses to decouple the security by implementing the security policy enforcement in a common system such as the ESB.

Rule requestors send requests over a secure transport to the ESB, which in turn forwards the request to the BRMS. Replies are delivered from the BRMS to the ESB, which in turn sends them over a secure transport to the originating rule requestors.

Design considerations

The design considerations are as follows:

- If the ESB provides a consolidated connection point for multiple clients or for multiple BRMS (and other services), the roadmap of volumes and values of messages must be well understood. The reasons are of performance (Can SLAs be achieved?) and availability (What happens if the single connection point is offline for maintenance?).
- The ESB must not rely on specific rule input and output message formats so that, for instance, if a new rule is deployed the ESB does not need to be updated.
- The request and response are likely to be processed by separate ESB message flows and therefore be performed asynchronously. As with any asynchronous pattern that is implemented within an ESB, a necessary task might be to store some context from the inbound message to allow correlation of requests and replies.

8.3 Integration pattern considerations

This section describes integration patterns that are applicable for use with ESBs, supplementing the generic BRMS integration patterns introduced in Chapter 5, “Identifying the patterns in general solutions” on page 51.

Several implementation alternatives are available for the integration of the ESB and WebSphere ILOG JRules. This section describes the considerations of each implementation alternative in addition to the advantages and disadvantages described in Chapter 5, “Identifying the patterns in general solutions” on page 51.

This section also concentrates on how the ESB integrates with the WebSphere ILOG JRules rules engine.

Note: ESBs can exploit other WebSphere ILOG JRules tools such as the Rule Team Server and the Decision and Validation Service.

8.3.1 Web services: hosted transparent decision services

In this pattern, ESBs invoke web services and Rules Execution Server deployed into a J2EE environment. For example, Figure 8-7 shows WebSphere Message Broker invoking web services deployed in WebSphere Application Server.

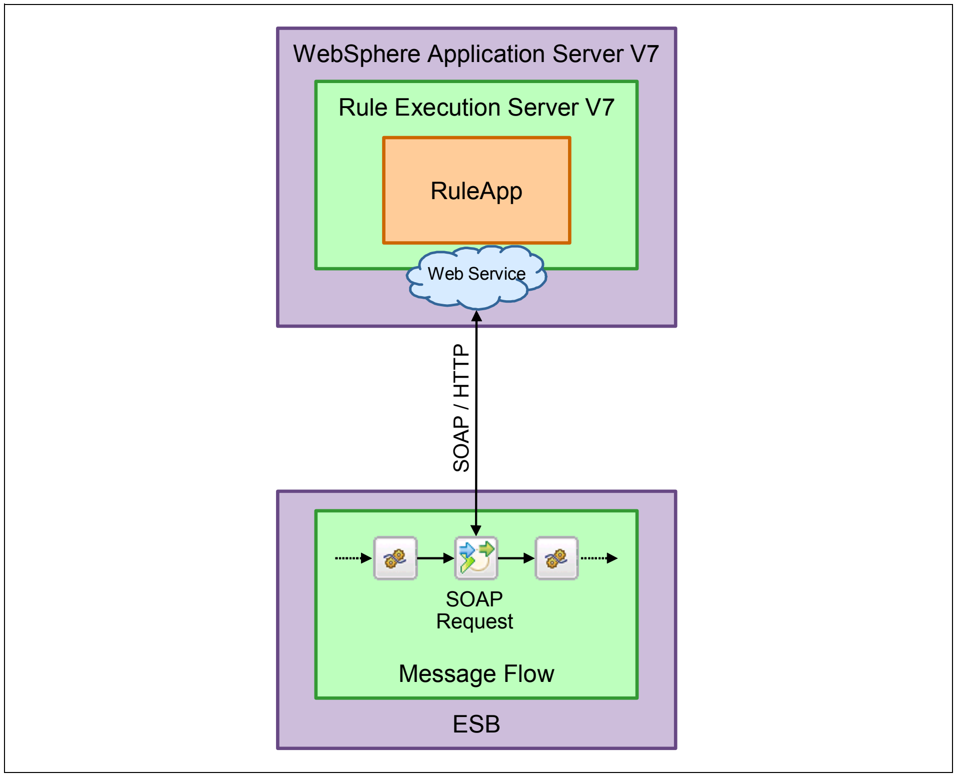


Figure 8-7 Web services (hosted transparent decision services) integration

Consider the following information:

- Easy integration is possible from message flows using WSDL exported from WebSphere ILOG JRules. The RES console provides WSDL at the click of a link. The WSDL can be imported into the tooling environment for the ESB.
- If the rule does not preserve and return the input message in its OUT or INOUT parameters, a necessary step might be to store the input message to allow for its reconstruction after the rule has been invoked.
- Web services can be invoked synchronously or asynchronously. If you are concerned about the implications of synchronously invoking rules, ESB offers asynchronous variants that can be used instead. When rules are invoked asynchronously, state information can be stored within the ESB to correlate requests and replies to and from WebSphere ILOG JRules.

8.3.2 Web services: custom

In this pattern, ESBs invoke web services and Rules Execution Server deployed into a J2EE environment.

Although generic reasons exist for choosing between web services (custom) and web services (hosted transparent decision services), as described in Chapter 5, “Identifying the patterns in general solutions” on page 51, the considerations that are specific to ESB are similar:

- Easy integration from message flows using WSDL that is exported from WebSphere ILOG JRules. WSDL is made available through links in readme files in Rule Studio web services (custom) projects. The WSDL can be imported into the tooling environment for the ESB.
- If the rule does not preserve and return the input message in its OUT or INOUT parameters, a necessary step might be to store the input message to allow for its reconstruction after the rule has been invoked.
- Web services can be invoked synchronously or asynchronously. If you are concerned about the implications of synchronously invoking rules, use the asynchronous invocations. When rules are invoked asynchronously, state information can be stored within the ESB to correlate requests and replies to and from WebSphere ILOG JRules.

8.3.3 J2EE asynchronous custom web services patterns: MDB

In this pattern, ESBs invoke web services and Rules Execution Server deployed into a J2EE environment through JMS and a message-driven bean (MDB).

For example, Figure 8-8 shows WebSphere Message Broker invoking an MDB deployed in WebSphere Application Server.

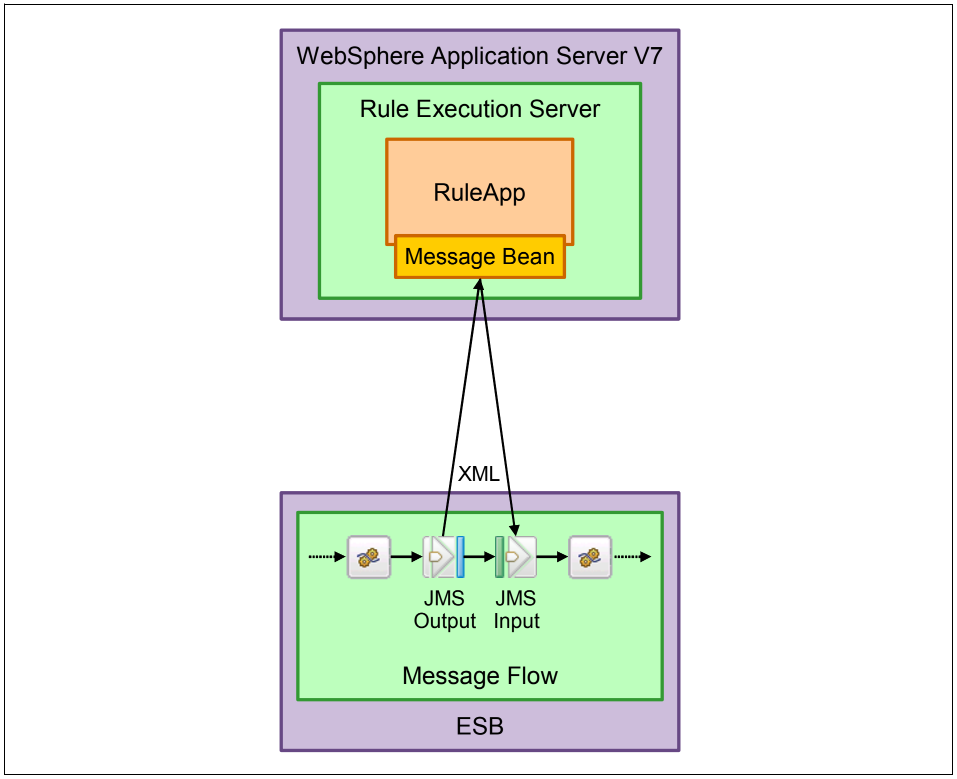


Figure 8-8 MDB integration

Consider the following information:

- JMS offers a “messaging-centric” approach, which might be suitable for integrating solutions that already exploit message queuing, such as those involving WebSphere MQ or the default messaging provider in WebSphere Application Server.
- When using this approach, the message flow has to build the message required for input to the rule and put the message on the JMS queue. The response to the invocation of the rule is returned to a JMS queue and processed by a separate message flow. The rule invocation is therefore

performed asynchronously. As with any asynchronous pattern that is implemented within an ESB, a necessary step is to store some context from the inbound message to allow correlation of requests and replies.

8.3.4 JRules Java SE rules engine

In this pattern, ESBs invoke web services and Rules Execution Server that are deployed within a Java SE rules engine. This means that the RES can be embedded in Java run time of ESBs. For example, Figure 8-9 shows a Rule Execution Server that is integrated at the Java level inside a WebSphere Message Broker JavaCompute node.

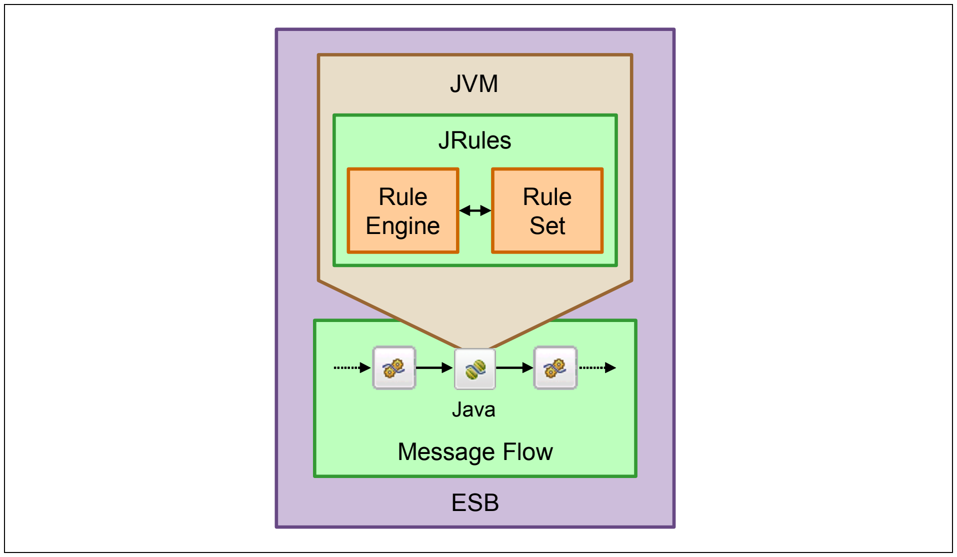


Figure 8-9 Embedded RES in the WebSphere Message Broker JavaCompute node

Consider the following information:

- Using an embedded RES is a more complex solution because it is not natively supported by ESBs and requires the development of Java code. However, this approach enables an integration solution to be designed that offers highly flexible, efficient, and optimized integration patterns than are possible by using native features of the ESB.
- Integrations that use this approach might offer the lowest latency and best overall performance, although no performance analysis or studies have been included within the scope of this book.

8.3.5 J2EE synchronous custom web services patterns: SCA

In this pattern, ESBs invoke an SCA component that is deployed into an SCA run time. For example, Figure 8-10 shows WebSphere Message Broker invoking an SCA component, deployed in WebSphere Application Server.

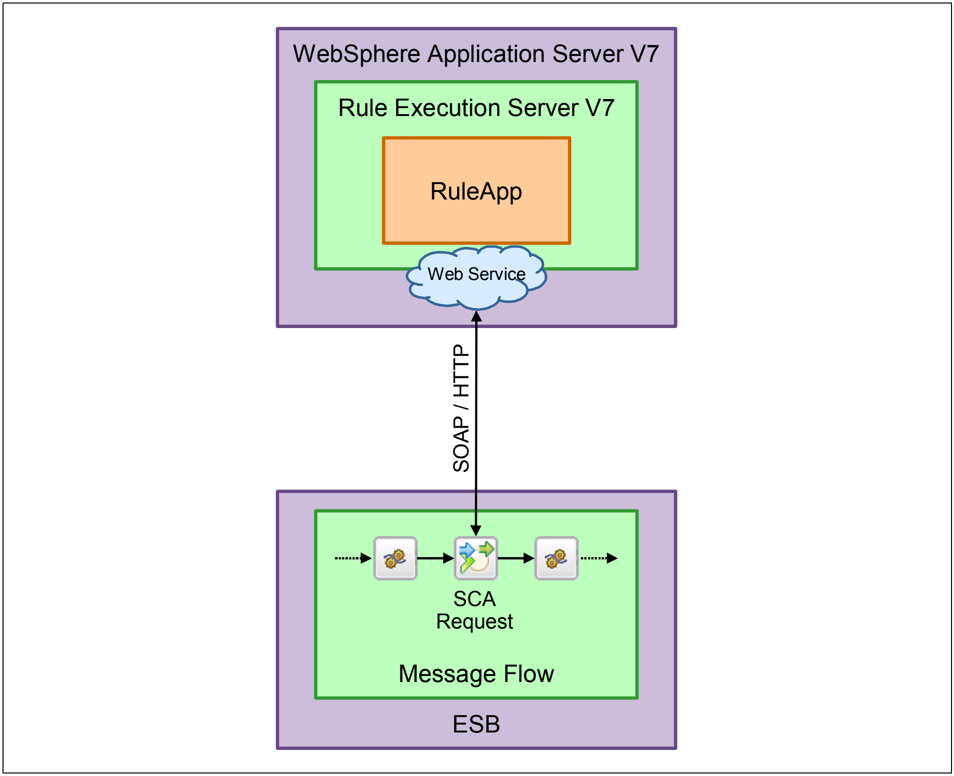


Figure 8-10 SCA integration

Consider the following information:

- This integration option is useful where ESBs offer SCA invocation features.
- This type of integration might be appropriate where rules are currently, or expected to, be heavily exploited in the BPM space and there is scope for reusing rules in the ESB.
- SCA components can be invoked synchronously or asynchronously. If you are concerned about the implications of synchronously invoking rules, use the asynchronous invocations. When rules are invoked asynchronously, state information can be stored within the ESB to correlate requests and replies to and from WebSphere ILOG JRules.

8.4 Integration patterns in IBM products

As described previously, our connectivity infrastructure can be implemented by a variety of software components or products within our enterprise. In this section, we describe the IBM products that are likely to be deployed within your connectivity infrastructure and describe the relevant integration patterns and considerations.

8.4.1 WebSphere Message Broker solutions

WebSphere Message Broker is the solution for clients who seek an ESB that provides connectivity and transformation, spanning heterogeneous IT environments. With its support for a broad range of transports, protocols, and data formats, WebSphere Message Broker performs the necessary conversions between diversely instrumented systems, so that they are able to communicate with each other freely. WebSphere Message Broker enables intelligent applications and services to receive and generate data that is based on an interconnected infrastructure and makes smarter decisions possible.

WebSphere Message Broker offers a solution to transform and route business data between applications:

- It can function as a message and protocol switch, so you can connect disparate applications and business data across multiple platforms.
- It delivers transformation and intelligent routing capabilities for all your business information.
- It makes business data available exactly where you want it in the format you need it.

WebSphere Message Broker provides a flexible and dynamic infrastructure to simplify application integration. With its robust design, scalable architecture, high performance, and ease of use, you can implement an enterprise-wide SOA in stages.

With this flexible combination of functions, your organization is given the tools to solve your integration requirements from any starting point: a simple two-application solution or a multi-application, multiplatform design.

WebSphere Message Broker can accommodate many IT environments with support for an array of application and middleware technologies. In addition, it offers the following benefits:

- Exploits the industry-leading WebSphere MQ messaging infrastructure.
- Supports a broad range of transport protocols:
  - Integration with WebSphere MQ, JMS 1.1, HTTP(S)
  - Web services, such as SOAP and REST
  - Files
  - Service Component Architecture (SCA)
  - TCP/IP, and SAP software and data formats, for example, binary, text, XML, and industry formats
- Offers numerous ways to customize mediation, for example, route, filter, transform, and filter.
- Uses a simple programming model for connectivity and mediation, including a robust set of prebuilt patterns.
- Supports a wide range of transformation options with graphical mapping, Java, ESQL, XSL, PHP, and WebSphere Transformation Extender.
- Delivers extensive administration and systems management facilities for developed solutions.
- Delivers similar performance to traditional transaction processing systems.

Available integration mechanisms are as follows:

- SOAP nodes can invoke web services (hosted transparent decision services) or web services (custom), synchronously or asynchronously.
- SCA nodes can invoke SCA components, synchronously or asynchronously.
- JMS nodes can invoke a JRules MDB.
- A RES can be embedded by using Java code inside a JavaCompute node or user-defined node.

8.4.2 WebSphere ESB solutions

WebSphere ESB is a flexible connectivity infrastructure for integrating applications and services. It enables the development of an SOA. WebSphere ESB powers the SOA by reducing the complexity of integrating the applications and services. WebSphere ESB is ideal if you have other solutions on WebSphere Application Server, WebSphere Portal, or the BPM platform. You can achieve efficiencies in skills, cost, and time-to-value across middleware products by adding WebSphere ESB to your standards-based IT environment. WebSphere ESB provides the following benefits:

- Allows powerful development in WebSphere Integration Developer with pattern-based development, guided task flows, mapping enhancements, and simplification.
- Simplifies system installation and cluster configuration.
- Includes enhanced operational visibility with enhanced service monitoring and health and problem determination business space widgets.
- Improves operational flexibility with enhanced module administration business space widgets and endpoint-based mediation policy support.

WebSphere ESB offers a solution to transform and route business data between applications. Because it can function as a message and protocol switch, you can connect disparate applications and business data across multiple platforms. It also delivers transformation and intelligent routing capabilities for all your business information.

WebSphere ESB can accommodate many IT environments with support for an array of application and middleware technologies. In addition, it offers the following benefits:

- Exploits WebSphere Application Server infrastructure.
- Supports a broad range of transport protocols, including:
  - Integration with WebSphere MQ, JMS 1.1, HTTP(S)
  - Web services, such as SOAP and REST
  - Files
  - Service Component Architecture (SCA)
  - TCP/IP, and SAP Software and data formats, for example, binary, text, XML, and industry formats
- Offers numerous ways to customize mediation, for example, route, filter, transform, and filter.
- Uses a simple programming model for connectivity and mediation, including a robust set of prebuilt patterns.
- Supports a wide range of transformation options with graphical mapping, Java, XSL, and WebSphere Transformation Extender.
- Delivers administration and systems management facilities for developed solutions.

In this book, we do not explicitly explore the integration approaches with WebSphere ESB because the mediation and connectivity capabilities of this product are included within WebSphere Process Server. As such, the integration with WebSphere ILOG JRules is effectively covered with the previous BPM section.

Available integration mechanisms are as follows:

- SOAP bindings can invoke web services (hosted transparent decision services) or web services (custom), synchronously or asynchronously.
- SCA bindings can invoke SCA components, synchronously or asynchronously.
- JMS bindings can invoke a JRules MDB.
- EJB bindings can invoke J2EE EJB components, synchronously or asynchronously.
- Embed a RES using Java code inside a mediation primitive.

8.4.3 WebSphere DataPower Integration Appliance solutions

WebSphere DataPower Integration Appliance XI50 is IBM hardware ESB, delivering common message transformation, integration, and routing functions in a network device, cutting operational costs and improving performance. By making on-demand data integration part of the shared SOA infrastructure, the XI50 is one of the few nondisruptive technologies for application integration.

The WebSphere DataPower Integration Appliance offers a solution to transform and route business data between applications. Because it can function as a message and protocol switch, you can connect disparate applications and business data across multiple platforms. It also delivers transformation and intelligent routing capabilities for all your business information. It makes business data available exactly where you want it in the format you need it.

The WebSphere DataPower Integration Appliance offers the following benefits:

- Wire speed routing and transformation
- Simple deploy model that is associated with appliances
- Browser-based administration tooling that can be easy to use.
- Exploits the industry-leading WebSphere MQ messaging infrastructure.

- Supports a broad range of transport protocols, including the following protocols:
  - Integration with WebSphere MQ, JMS 1.1, HTTP(S)
  - Web services, such as SOAP and REST
  - Files
  - Service Component Architecture (SCA)
  - TCP/IP, and SAP Software and data formats, for example, binary, text, XML, and industry formats

In this book, we do not perform integration of JRules using a DataPower appliance. However, we expect that the types of integration approaches are less than are available on the other software products we explore:

- SCA component invocation is not natively supported in DataPower, therefore this option is not possible.
- DataPower is not a Java implementation (such as WebSphere ESB or WebSphere Process Server) and does not contain a JVM within its architecture (such as WebSphere Message Broker), so the option of an embedded Rule Execution Server is not available.

Available integration mechanisms are as follows:

- HTTP handlers can invoke web services (hosted transparent decision services) or web services (custom).
- WMQ and JMS handlers can invoke a JRules MDB.

Part 3

Realizing the scenarios

Part 3 contains the following chapters:

9

Integrating CICS applications

In this chapter, we cover a scenario that integrates CICS applications.

This chapter contains the following topics:

- Introduction to the scenario
- Running the CICS web services to JRules demonstration
- Integration with CICS web services call to ILOG custom web services
- ILOG Rules for COBOL

9.1 Introduction to the scenario

In this section, we look at an introduction to the scenario.

9.1.1 Overview, products used, and content described

In this scenario, we demonstrate how a simple CICS pricing application can be modified to take advantage of business rules. The application we use is a CICS application that is hosted in a CICS TS 4.1 system. By integrating business rules into a CICS application, we enable the CICS application to become much more flexible and agile. This flexibility means that business rules can be changed inside the application without having to go through a whole application development cycle. With this approach, we can decouple changes to the business rules from the application development life cycle.

An additional advantage of using business rules is that we can remove (sometimes complex) rules from the application code, then store and document them in natural language. This advantage enables us to understand all the business rules we have in our application and gives us confidence that we understand the implications of the changes we are making.

In this chapter, we demonstrate two methods of adding business rules to our CICS application:

- Integration of a CICS application and JRules Rule Execution Server using CICS web services and JRules hosted transparent decision services
- Adding business rules into a CICS application through a call to a COBOL subprogram, generated by Rules for COBOL

Products used in this scenario

The software we used in this scenario is as follows:

- CICS Transaction Server version 4.1
- WebSphere Application Server 7.0
- WebSphere ILOG JRules version 7.0.3
- WebSphere ILOG Rules for COBOL version 7.0.3

Chapter content

In this chapter we describe the following topics:

- The high-level structure of the existing CICS pricing application.
- The externals of the existing application in detail.
- The detailed internals of the CICS pricing application:
  - The application algorithms and code
  - The CICS resources required by the application
- The business rules that we have found in the existing CICS pricing application.
- The JRules (and Rules for COBOL) Rule Projects that we created from the business rules we found.
- How to quickly set up a demonstration of the CICS Pricing application calling JRules using CICS web services.
- How to set up the original CICS Pricing application to call a Hosted Transparent Decision Service using CICS web services.
- How to generate a Rules for COBOL subprogram to implement the pricing rules, and how to integrate this subprogram with the original CICS pricing application.

9.1.2 Structure of the CICS pricing application

The CICS application in this scenario provides a price for a motor insurance quote. The price of the quote is based on the value of the vehicle, the vehicle type, and the level of deduction or excess to be applied to the policy.

Our CICS application consists of three programs that perform the following tasks:

- Program one provides a basic mapping support (BMS) interface. The second program drives this interface and takes data from it.
- Program two then passes the data from the first program by using a channel and container on an EXEC CICS LINK command to the third program.
- Program three runs the pricing algorithm. If a price is successfully obtained, the application displays the price and a message that details any surcharges or discounts that have been applied.

Figure 9-1 shows the structure of the application and the names of the CICS programs that implement the function.

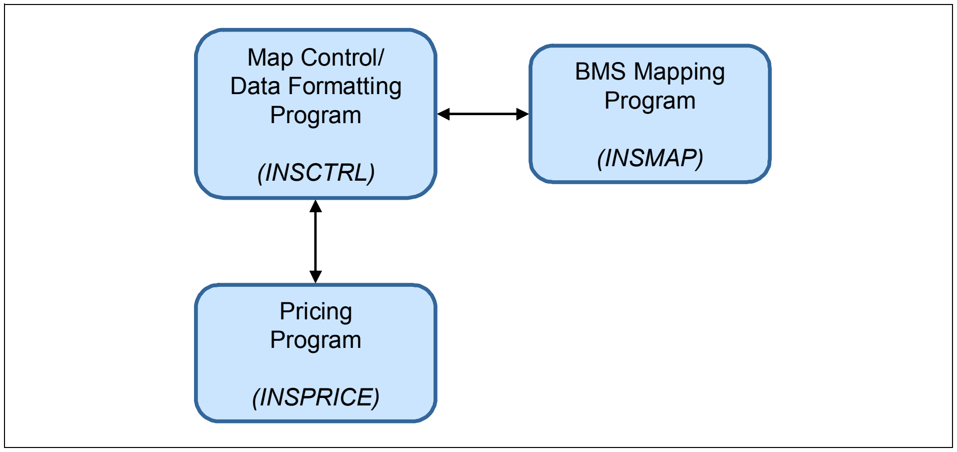


Figure 9-1 Structure of CICS pricing application

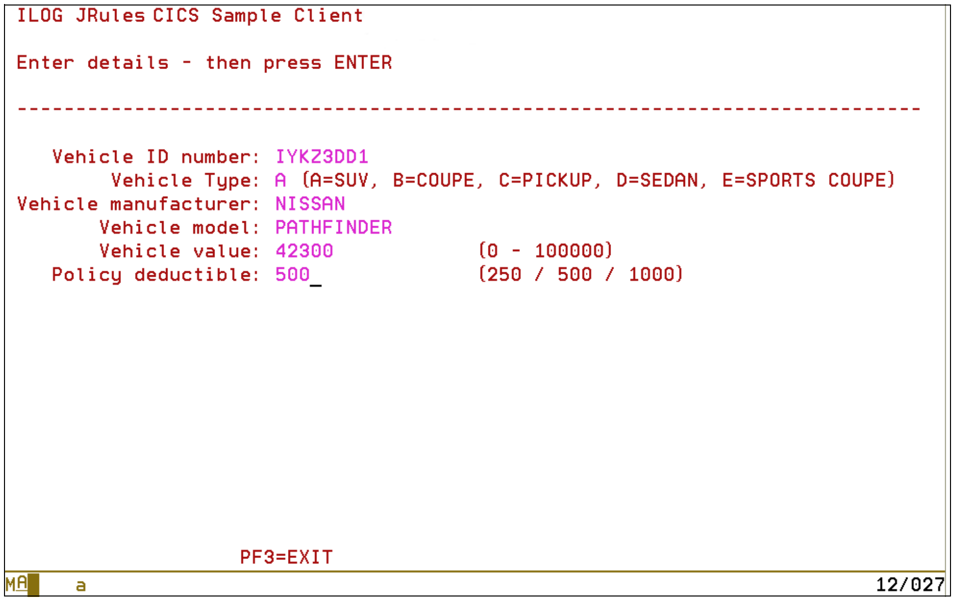
9.1.3 CICS pricing application externals

In this section, we look at the externals for the CICs pricing application.

Pricing application data input panel

On the first window of our pricing application, the user enters information about the car to be insured, and details about the deductible or excess to be applied to the policy.

Figure 9-2 shows an example of the data that is entered into this window.



```
ILOG JRules CICS Sample Client

Enter details - then press ENTER

----------------------------------------------------------------------------

   Vehicle ID number: IYKZ3DD1
        Vehicle Type: A (A=SUV, B=COUPE, C=PICKUP, D=SEDAN, E=SPORTS COUPE)
Vehicle manufacturer: NISSAN
       Vehicle model: PATHFINDER
       Vehicle value: 42300            (0 - 100000)
   Policy deductible: 500_             (250 / 500 / 1000)











                  PF3=EXIT
MA      a                                                               12/027
```

Figure 9-2 Entering data into the pricing application

The fields on the first panel of the pricing application are as follows:

Vehicle ID number	Alphanumeric value

Vehicle Type	Single character field:
	A = Sport utility vehicle
	B = Coupe
	C = Pickup truck
	D = Sedan
	E = Sports coupe

Vehicle manufacturer	Alphanumeric value
Vehicle model	Alphanumeric value
Vehicle value	Numeric value in the range of 0 - 100000
Policy deductible	Numeric value of either 250, 500, or 1000

The critical fields for making a decision in this application are the Vehicle Type, Vehicle value, and Policy deductible.

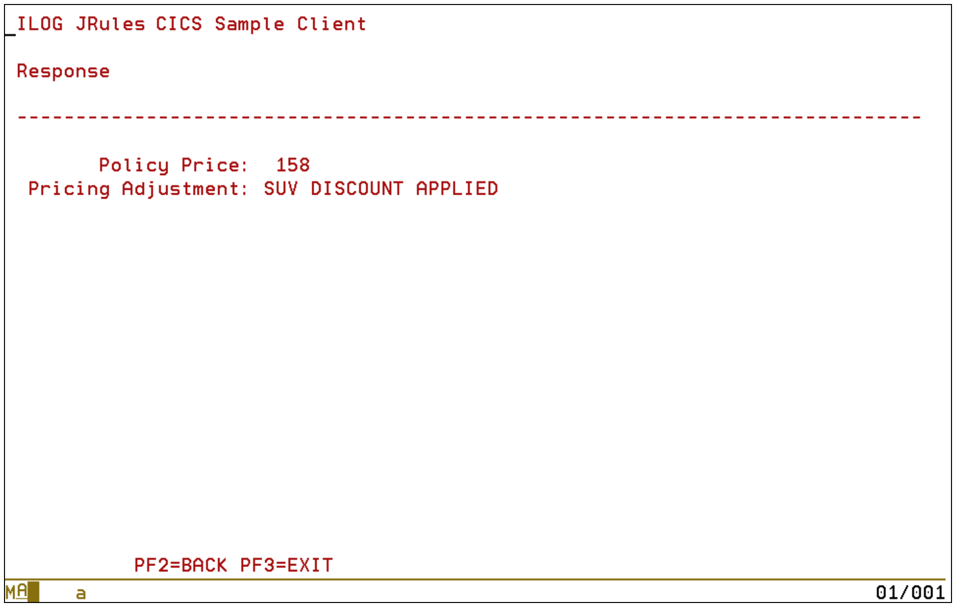
Figure 9-2 on page 191 shows that we set these fields to the following values:

- The vehicle type is A (for SUV).
- The vehicle value is 42300.
- The policy deductible is 500.

On the Pricing Application data input panel, we can either press Enter to process the pricing request, or press the F3 key to exit the application.

Pricing application output panel

The output panel of the pricing application opens after you press Enter on the application's input data panel. Figure 9-3 shows the output produced when the input described in the previous section is entered into the application.



```
_ILOG JRules CICS Sample Client

 Response

 -----------------------------------------------------------------------------


        Policy Price:  158
  Pricing Adjustment: SUV DISCOUNT APPLIED




















           PF2=BACK PF3=EXIT
MA      a                                                                 01/001
```

Figure 9-3 Output from the pricing application

The two fields on the pricing application output panel are as follows:

Policy Price	A numeric field that shows the final price of the policy
Pricing Adjustment	An alphanumeric field that displays a message informing the user about any discounts or surcharges that have been applied when producing the final price for the policy

Figure 9-3 shows that the final price of our policy is 158, and that a discount has been applied to the price as the vehicle to be insured is an SUV.

9.1.4 CICS pricing application internals

The following pricing application internals are in this section:

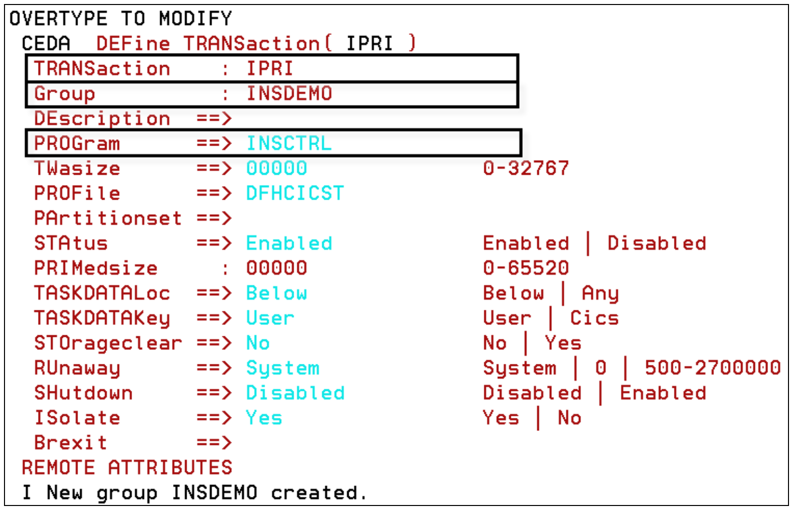
- Required CICS resources
- The INSCTRL program
- The INSPRICE program

Required CICS resources

To run the CICS pricing application, three CICS resources must be created:

- A CICS transaction named IPRI

  This transaction points to a program named INSCTRL (the mapping control and data formatting program). The transaction is added to a CSD group named INSDEMO. All other fields in the transaction definition are left as default. Figure 9-4 shows an example of the transaction definition with the relevant fields highlighted.

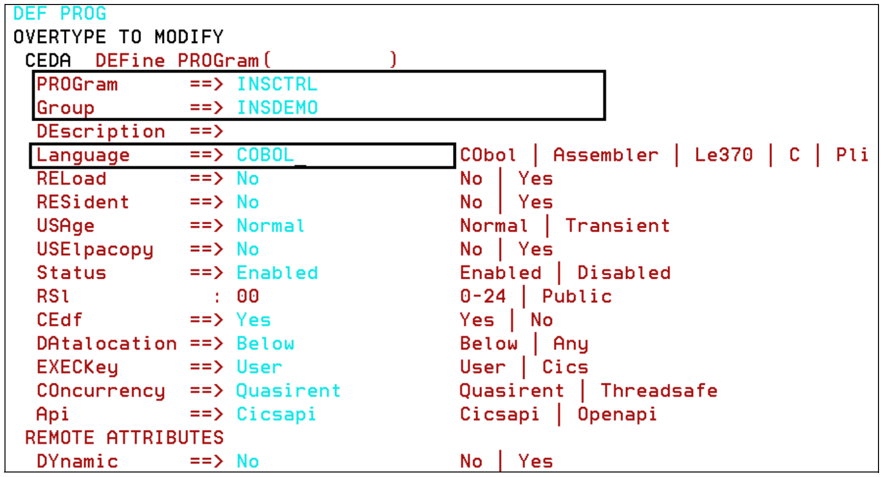


```
OVERTYPE TO MODIFY
 CEDA  DEFine TRANSaction( IPRI )
  TRANSaction     : IPRI
  Group           : INSDEMO
  DEscription  ==>
  PROGram      ==> INSCTRL
  TWasize      ==> 00000              0-32767
  PROFile      ==> DFHCICST
  PArtitionset ==>
  STAtus       ==> Enabled            Enabled | Disabled
  PRIMedsize      : 00000             0-65520
  TASKDATALoc  ==> Below              Below | Any
  TASKDATAKey  ==> User               User | Cics
  STOrageclear ==> No                 No | Yes
  RUnaway      ==> System             System | 0 | 500-2700000
  SHutdown     ==> Disabled           Disabled | Enabled
  ISolate      ==> Yes                Yes | No
  Brexit       ==>
 REMOTE ATTRIBUTES
 I New group INSDEMO created.
```

Figure 9-4 The IPRI transaction definition

- A program definition for the INSCTRL program. This COBOL program will be added to the INSDEMO CSD group. All other attributes in the program definition are left as default.

Figure 9-5 shows an example of the INSCTRL program definition with the modified parts highlighted.

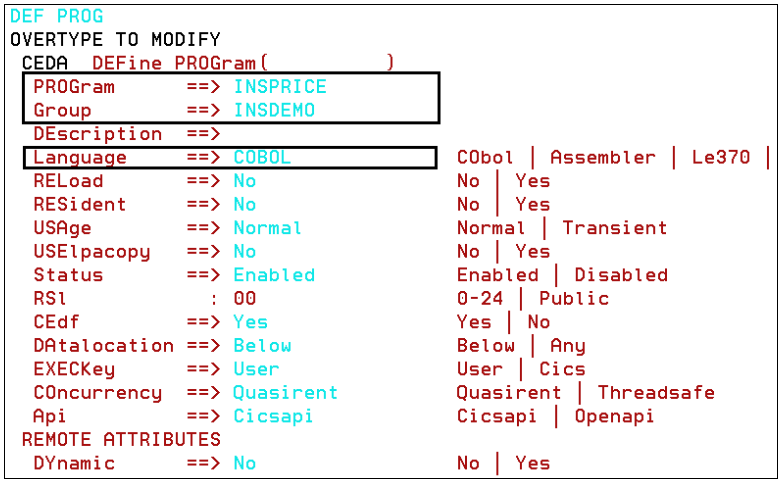


```
DEF PROG
OVERTYPE TO MODIFY
 CEDA  DEFine PROGram(          )
  PROGram      ==> INSCTRL
  Group        ==> INSDEMO
  DEscription  ==>
  Language     ==> COBOL_            COBol | Assembler | Le370 | C | Pli
  RELoad       ==> No                No | Yes
  RESident     ==> No                No | Yes
  USAge        ==> Normal            Normal | Transient
  USElpacopy   ==> No                No | Yes
  Status       ==> Enabled           Enabled | Disabled
  RSl            : 00                0-24 | Public
  CEdf         ==> Yes               Yes | No
  DAtalocation ==> Below             Below | Any
  EXECKey      ==> User              User | Cics
  COncurrency  ==> Quasirent         Quasirent | Threadsafe
  Api          ==> Cicsapi           Cicsapi | Openapi
 REMOTE ATTRIBUTES
  DYnamic      ==> No                No | Yes
```

Figure 9-5 The INSCTRL program definition

The INSCTRL program we defined performs an EXEC CICS LINK to a program named INSPRICE to perform the actual pricing of the insurance quote. We now need to define a program named INSPRICE. Like the INSCTRL program it is also a COBOL program that will be defined to the CSD group INSDEMO.

Figure 9-6 shows an example of the INSPRICE program definition with the modified parts highlighted.

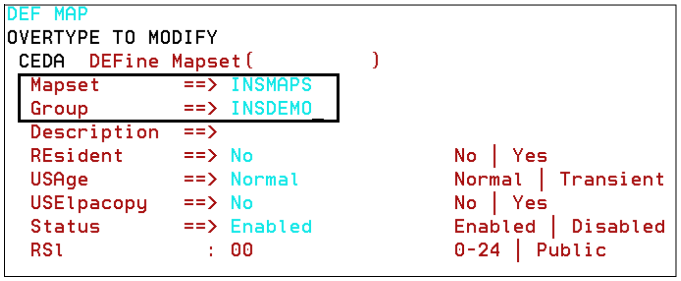


```
DEF PROG
OVERTYPE TO MODIFY
 CEDA  DEFine PROGram(          )
  PROGram      ==> INSPRICE
  Group        ==> INSDEMO
  DEscription  ==>
  Language     ==> COBOL             COBol | Assembler | Le370 |
  RELoad       ==> No                No | Yes
  RESident     ==> No                No | Yes
  USAge        ==> Normal            Normal | Transient
  USElpacopy   ==> No                No | Yes
  Status       ==> Enabled           Enabled | Disabled
  RSl            : 00                0-24 | Public
  CEdf         ==> Yes               Yes | No
  DAtalocation ==> Below             Below | Any
  EXECKey      ==> User              User | Cics
  COncurrency  ==> Quasirent         Quasirent | Threadsafe
  Api          ==> Cicsapi           Cicsapi | Openapi
 REMOTE ATTRIBUTES
  DYnamic      ==> No                No | Yes
```

Figure 9-6 The INSPRICE program definition

- The final CICS resource that we define to the CSD is the mapset for the BMS maps that are used in the application. The mapset is named INSMAPS, and it is added to the CSD group INSDEMO. All values for the mapset are left as default.

  Figure 9-7 shows an example of the INSMAPS mapset definition with the modified parts highlighted.



```
DEF MAP
OVERTYPE TO MODIFY
 CEDA  DEFine Mapset(          )
  Mapset       ==> INSMAPS
  Group        ==> INSDEMO_
  Description  ==>
  REsident     ==> No                  No | Yes
  USAge        ==> Normal              Normal | Transient
  USElpacopy   ==> No                  No | Yes
  Status       ==> Enabled             Enabled | Disabled
  RSl            : 00                  0-24 | Public
```

Figure 9-7   The INSMAPS mapset definition

The INSCTRL program

As mentioned previously, the INSCTRL program controls the user interaction and formats data before passing it to the INSPRICE program to perform the pricing function. In this section, we examine the internals of the INSCTRL program in more detail. Figure 9-8 on page 196 shows the program flow of INSCTRL.

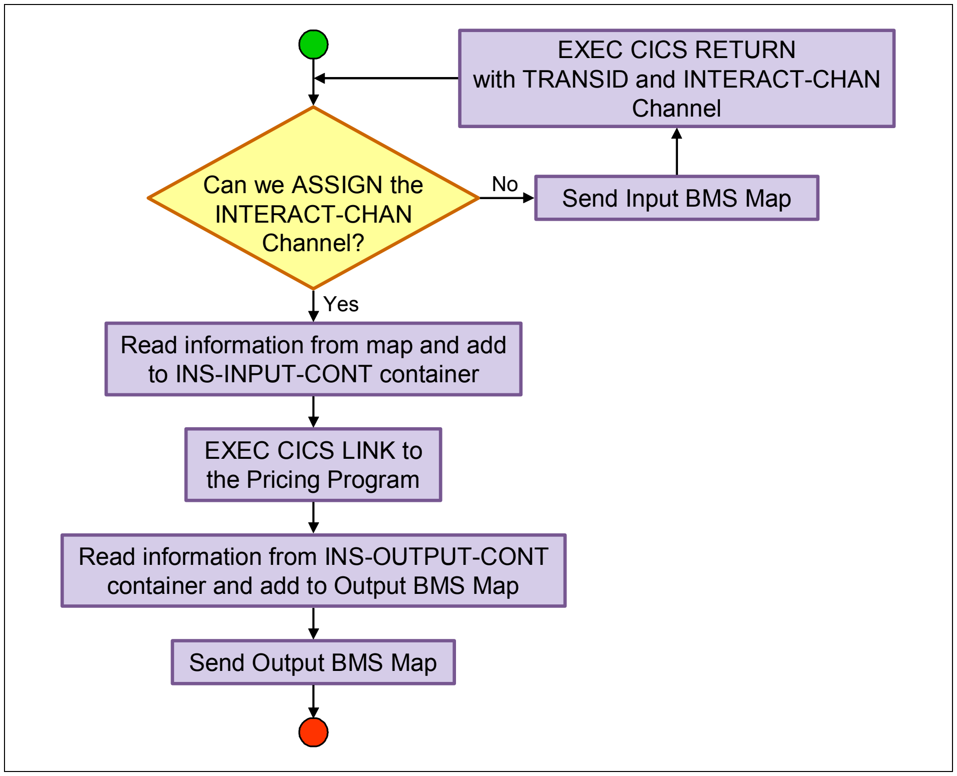

EXEC CICS RETURN with TRANSID and INTERACT-CHAN Channel
Can we ASSIGN the INTERACT-CHAN Channel?
No
Send Input BMS Map
Yes
Read information from map and add to INS-INPUT-CONT container
EXEC CICS LINK to the Pricing Program
Read information from INS-OUTPUT-CONT container and add to Output BMS Map
Send Output BMS Map

Figure 9-8 INSCTRL program flow

Note: We simplified the program flow by removing the flows that are controlled when the user presses the F3 (exit) or F2 (return to previous panel) key.

In the INSCTRL program, two CICS channels are used:

- The INTERACT-CHAN channel is used to control the program flow between invocations of INSCTRL. The program flow information is stored in this channel in a single container named INTERACT-CONT.
- The INSPRICE-CHAN channel is used to send information to and receive information from the INSPRICE pricing program. Data is sent on this channel in two containers:
  - The INS-IN-CONT container is used to send input to the INSPRICE program. It contains information about the vehicle to be insured, and also the amount of any deductible to be applied to the insurance policy.

– The INS-OUT-CONT container is used to receive output from the INSPRICE program. It contains the policy price and information about any discounts or surcharges that have been applied to the price.

The INSCTRL program starts by performing an EXEC CICS ASSIGN CHANNEL command. It does this to determine whether any program flow information has been left from a previous invocation of the INSCTRL program. If it cannot assign the INTERACT-CHAN channel it assumes that this invocation is the first time. In this situation, INSCTRL issues an EXEC CICS SEND MAP command to display the input BMS map and adds program flow control data to the INTERACT-CONT container on the INTERACT-CHAN channel.

When this operation has completed, it finishes the transaction by performing an EXEC CICS RETURN command. The ID of the transaction that started INSCTRL is supplied on the EXEC CICS RETURN command along with the INTERACT-CHAN channel. By passing this information about the RETURN command, we ensure that the next time the user presses the Enter key the INSCTRL program is re-invoked with the INTERACT-CONT container from its previous invocation.

On the second invocation of INSCTRL, the program reads and formats the information that is supplied on the input BMS map. This information is then added to a data structure, which is then added to the INS-IN-CONT container on the INSPRICE-CHAN channel. Example 9-1 shows the COBOL data structure that is added to the INS-IN-CONT container.

Example 9-1 The data structure added to the INS-IN-CONT container

```
* Quote input container structure
 01 CA-QUOTE-INPUT.
     03 CA-VEHICLE-DETAILS.
         05 CA-VEHICLE-ID-NUM        PIC X(50).
         05 CA-VEHICLE-TYPE          PIC X(1).
             88 SUV                  VALUE 'A'.
             88 COUPE                VALUE 'B'.
             88 PICKUP               VALUE 'C'.
             88 SEDAN                VALUE 'D'.
             88 SPORTS-COUPE         VALUE 'E'.
         05 CA-VEHICLE-MANUFACTURER  PIC X(30).
         05 CA-VEHICLE-MODEL         PIC X(30).
         05 CA-VEHICLE-VALUE         PIC 9(6).
     03 CA-DEDUCTIBLE              PIC 9(4).
```

After INSCTRL has stored all the quote information in the INS-IN-CONT container, it performs an EXEC CICS LINK command to the INSPRICE program.

It passes the INSPRICE-CHAN channel on this command to enable the INSPRICE program to receive the INS-IN-CONT container.

When the INSPRICE program returns control to the INSCTRL program, the INSCTRL program will get output information from INSPRICE. It achieves this by getting the INS-OUT-CONT container from the INSPRICE-CHAN channel. INSCTRL stores the information from the container in a COBOL data structure. Example 9-2 shows the COBOL data structure used to receive the output.

Example 9-2 The data structure that receives data from the INS-OUT-CONT container

```
* Quote output container structure
 01 CA-QUOTE-OUTPUT.
     03 CA-PREMIUM               PIC 9(4).
     03 CA-RETURN-CODE           PIC 9(2).
     03 CA-RESPONSE-MESSAGE      PIC X(80).
```

After formatting the data, INSCTRL adds it into the output BMS map and performs an EXEC CICS SEND MAP command to send the response to the user's terminal.

The INSPRICE program

The INSPRICE program starts by taking information from the INS-IN-CONT container, passed to it by INSCTRL. The program first performs a series of IF-THEN-ELSE commands to try to determine a base price for the insurance quote. After obtaining this base price it then runs through a series of IF statements to determine whether any discounts or surcharges must be applied. After completion, INSPRICE puts its output into the INS-OUT-CONT container that is returned to INSCTRL.

Example 9-3 shows the code that INSPRICE uses to price an insurance quote.

Example 9-3 Code example for pricing an insurance quote

```
GENERATE-BASE-PRICE.

* RULE:  Vehicle value is between 0 and 5500
     IF CA-VEHICLE-VALUE IS GREATER THAN ZERO AND
        CA-VEHICLE-VALUE IS NOT GREATER THAN 5500
         IF CA-DEDUCTIBLE IS EQUAL TO 250 THEN
           MOVE 120 TO WS-BASE-PREMIUM
         ELSE
           IF CA-DEDUCTIBLE IS EQUAL TO 500 THEN
             MOVE 110 TO WS-BASE-PREMIUM
           ELSE
             IF CA-DEDUCTIBLE IS EQUAL TO 1000 THEN
```

```
                MOVE 100 TO WS-BASE-PREMIUM
              ELSE
                MOVE 98 TO CA-RETURN-CODE
                MOVE 'INVALID DEDUCTIBLE AMOUNT'
                  TO CA-RESPONSE-MESSAGE
                EXEC CICS RETURN END-EXEC
              END-IF
            END-IF
          END-IF
      END-IF

* RULE:  Vehicle value is between 5501 and 11000
      IF CA-VEHICLE-VALUE IS GREATER THAN 5500 AND
         CA-VEHICLE-VALUE IS NOT GREATER THAN 11000
          IF CA-DEDUCTIBLE IS EQUAL TO 250 THEN
            MOVE 130 TO WS-BASE-PREMIUM
          ELSE
            IF CA-DEDUCTIBLE IS EQUAL TO 500 THEN
              MOVE 120 TO WS-BASE-PREMIUM
            ELSE
              IF CA-DEDUCTIBLE IS EQUAL TO 1000 THEN
                MOVE 110 TO WS-BASE-PREMIUM
              ELSE
                MOVE 98 TO CA-RETURN-CODE
                MOVE 'INVALID DEDUCTIBLE AMOUNT'
                  TO CA-RESPONSE-MESSAGE
                EXEC CICS RETURN END-EXEC
              END-IF
            END-IF
          END-IF
      END-IF

...
...
...

* RULE:  Vehicle value is between 11001 and 20000
      IF CA-VEHICLE-VALUE IS GREATER THAN 11001 AND
         CA-VEHICLE-VALUE IS NOT GREATER THAN 20000
          IF CA-DEDUCTIBLE IS EQUAL TO 250 THEN
            MOVE 150 TO WS-BASE-PREMIUM
          ELSE
            IF CA-DEDUCTIBLE IS EQUAL TO 500 THEN
              MOVE 145 TO WS-BASE-PREMIUM
            ELSE
```

```
                IF CA-DEDUCTIBLE IS EQUAL TO 1000 THEN
                  MOVE 140 TO WS-BASE-PREMIUM
                ELSE
                  MOVE 98 TO CA-RETURN-CODE
                  MOVE 'INVALID DEDUCTIBLE AMOUNT'
                    TO CA-RESPONSE-MESSAGE
                  EXEC CICS RETURN END-EXEC
                END-IF
              END-IF
            END-IF
     END-IF

      EXIT.

EVALUATE CA-VEHICLE-TYPE
       WHEN 'A'
* RULE:  SUV discount (1%)
         COMPUTE WS-BASE-PREMIUM = WS-BASE-PREMIUM * 0.99
         MOVE 'SUV DISCOUNT APPLIED'
           TO CA-RESPONSE-MESSAGE
       WHEN 'C'
* RULE:  Pickup truck discount (2%)
         COMPUTE WS-BASE-PREMIUM = WS-BASE-PREMIUM * 0.98
         MOVE 'PICKUP TRUCK DISCOUNT APPLIED'
           TO CA-RESPONSE-MESSAGE
       WHEN 'E'
* RULE:  Sports coupe surcharge (4%)
         COMPUTE WS-BASE-PREMIUM = WS-BASE-PREMIUM * 1.04
         MOVE 'SPORTS COUPE SURCHARDE APPLIED'
           TO CA-RESPONSE-MESSAGE
       WHEN OTHER
* RULE:  No pricing adjustment applies
         MOVE 'NO PRICING ADJUSTMENT APPLIED'
           TO CA-RESPONSE-MESSAGE
     END-EVALUATE

     EXIT.
```

9.1.5 Business rules in the CICS pricing application

This section outlines the rules that exist in the CICS pricing application.

The business rules that are present in the INSPRICE program can be classified into two main types:

- Rules for determining the baseline price of an insurance quote

  These rules use the value and type of a vehicle along with any deductible applied to the insurance policy to produce a baseline price.
- Surplus and discounting rules

  These rules apply either a discount or a surcharge to the baseline price. The level of discount or surcharge is based on the type of vehicle being insured.

The rules that determine the baseline price for a quote are best described in Table 9-1, which shows the rules for baseline pricing in the INSPRICE program.

Table 9-1 Rules for determining the base price

Vehicle value		**Deductible**	**Base Price**
Minimum	**Maximum**		
0	5500	250	120
		500	110
		1000	100
5501	11000	250	130
		500	120
		1000	110
11001	20000	250	150
		500	145
		1000	140
20001	35000	250	160
		500	150
		1000	140

Vehicle value		Deductible	Base Price
Minimum	Maximum		
35001	55000	250	170
		500	160
		1000	150
55001	100000	250	190
		500	180
		1000	165

The second category of rules, the surplus and discounting rules, can be further decomposed into three distinct rules:

- When the vehicle to be insured is an SUV, apply a 1% discount to the base price.
- When the vehicle to be insured is a pickup truck, apply a 2% discount to the base price.
- When the vehicle to be insured is a sports coupe, apply a 4% surcharge to the base price.

9.1.6 Pricing application rule project

Later in this scenario, we use a rule project to create a rule application that can be deployed to a RES. We also use a rule project to generate COBOL code by using WebSphere ILOG Rules for COBOL. The rules and rule flows in both projects are the same, although the underlying implementation might differ. For example, the rule project to be deployed to a RES is based on an XSD schema; the Rules for COBOL project is based on a COBOL copybook.

This section describes the rule flows and rules that are common to both rule projects. Figure 9-9 on page 203 shows the overall structure of the rules used in the rule projects.

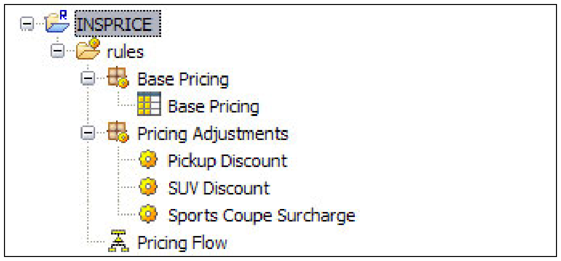


Figure 9-9 Structure of the INSPRICE rule project

The rule project contains two packages for our two types of rules:

- The Base Pricing package contains a decision table that implement the basic pricing rules.
- The Pricing Adjustments package contains the rules that either apply a discount or a surcharge to the basic price of a policy.

Our project also contains a rule flow used to define the order in which our rule packages are executed. Figure 9-10 shows the Pricing Flow rule flow in our project.

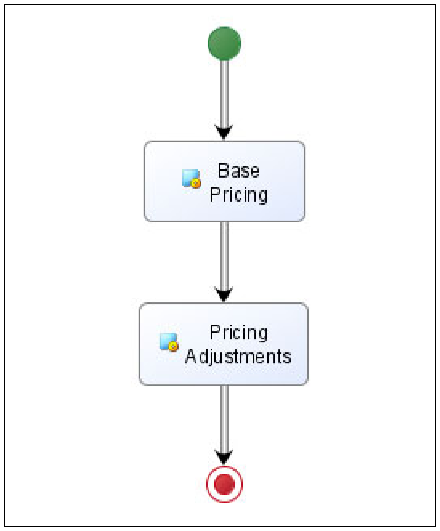
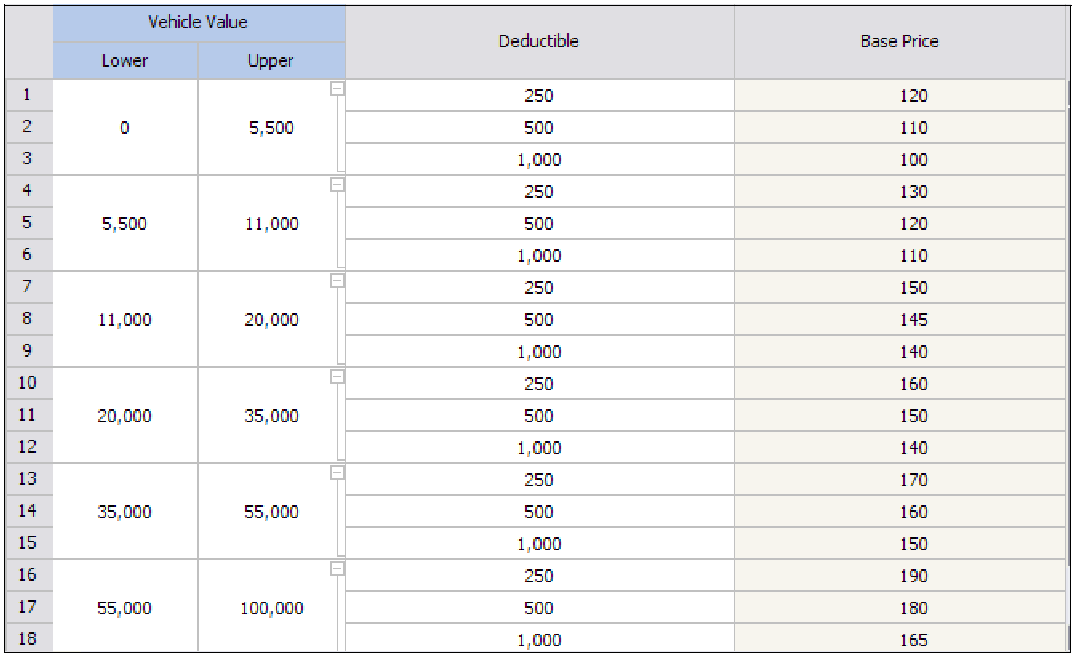


Figure 9-10 The Pricing Flow rule flow

The first package we execute in the rule flow is Base Pricing, which gives us our initial quote price before any adjustments are applied. The next package to be executed is Pricing Adjustments, which adjusts the initial quote price based on the type of vehicle being insured.

The Base Pricing package contains a decision table called Base Pricing. Figure 9-11 shows the structure of this decision table.



	Vehicle Value		Deductible	Base Price
	Lower	Upper		
1	0	5,500	250	120
2			500	110
3			1,000	100
4	5,500	11,000	250	130
5			500	120
6			1,000	110
7	11,000	20,000	250	150
8			500	145
9			1,000	140
10	20,000	35,000	250	160
11			500	150
12			1,000	140
13	35,000	55,000	250	170
14			500	160
15			1,000	150
16	55,000	100,000	250	190
17			500	180
18			1,000	165

Figure 9-11 The Base Pricing decision table

The Pricing Adjustments package has three business rules defined:

- Pickup Discount
- SUV Discount
- Sports Coupe Surcharge

Example 9-4 shows the definition of the Pickup Discount rule.

Example 9-4 The Pickup Discount rule

```
if
the vehicle of 'a request for quotation' is a Pickup Truck

then
apply a 2 % discount to the premium ;
set the response message of 'the quote' to "Pickup Truck Discount
Applied" ;
```

Example 9-5 shows the definition of the SUV Discount rule.

Example 9-5 The SUV Discount rule

```
if
the vehicle of 'a request for quotation' is a Sports Utility Vehicle

then
apply a 1 % discount to the premium ;
set the response message of 'the quote' to "SUV Discount Applied";
```

Example 9-6 shows the definition of the Sports Coupe Surcharge rule.

Example 9-6 The Sports Coupe Surcharge rule

```
if
the vehicle of 'a request for quotation' is a Sports Coupe

then
apply a 4 % surcharge to the premium ;
set the response message of 'the quote' to "Sports Coupe Surcharge
Applied" ;
```

We examine how the rule project described in this section can be integrated with our COBOL CICS pricing application. We examine two methods of integration:

- Integration through a CICS web service call to a rule application that is hosted on a Rule Execution Server.
- Integration with a WebSphere ILOG Rules for COBOL-generated program by using the Rules for COBOL CICS Wrapper program support.

9.2 Running the CICS web services to JRules demonstration

In the Additional Materials section of the Redbooks website (described in Appendix A, "Additional material" on page 359), we have included a set of resources that can help you to quickly set up a demonstration of CICS calling a Rule Execution Server using a web service.

To run this demonstration the following assumptions must be true:

- The demonstration is run on a CICS system that is capable of, and has been configured to, make an outbound web service call.
- The WebSphere ILOG Rule Execution server resides on a WebSphere Application Server to which CICS can make a web service call.
- No security setup can prevent a basic web service call from failing.

The following parts that are listed in Appendix A, "Additional material" on page 359 are required:

- The following RuleApp Archive file that we deploy into Rule Team Server:

  `ruleApp_InspriceWSproj_1.0.jar`

- The following CICS WSBind file that is used to call the Hosted Transparent Decision Service:

  `InspriceWS.wsbind`

- The following load library that contains the CICS pricing application executable:

  `ILOGRED.LOAD.XMIT`

- The PDS that contains various JCL jobs, which relate to setting up and compiling the CICS pricing application

  `ILOGRED.SOURCE.XMIT`

Several other parts are also listed in Appendix A, "Additional material" on page 359, but they are not required to run this demonstration:

- The following file contains the Rule Project described in the Rules for COBOL scenario that is discussed later in this chapter:

  `InspriceCbl.zip`

- The following file contains the Rule Project that is described in the web services scenario, which is discussed later in this chapter:

  `InspriceWS.zip`

- The following schema is used to create the Rule Project described in the web services scenario, which is discussed later in this chapter:

  `InspriceWS.xsd`
- The following WSDL file is for the hosted transparent decision service that we call from CICS:

  `InspriceWS.wsdl`
- The original source code of the CICS pricing application is as follows:

  `ILOGRED.SOURCE.XMIT`
- The following file contains source code that is specific to the Rules for COBOL scenario, which is described later in the chapter:

  `ILOGRED.SOURCE.R4C.XMIT`
- The following file contains source code that is specific to the web services scenario, which is described later in this chapter:

  `ILOGRED.SOURCE.WS.XMIT`
- The following file contains JCL to assist in the compilation of the application source code:

  `ILOGRED.PROC.XMIT`

9.2.1 Deploying the RuleApp archive to the Rule Execution Server

The first step in running the demonstration is to deploy the RuleApp archive to the Rule Execution Server:

1. Extract `ruleApp_InspriceWSproj_1.0.jar` file to a directory on your workstation.
2. Log on to the Rule Execution Server web console.
3. Click the **Explorer** tab.
4. In the Navigator section, click **RuleApps**. The RuleApps View opens.
5. Click **Deploy RuleApp Archive**.

Figure 9-12 shows the Rule Execution Server with all of the selected hyperlinks.

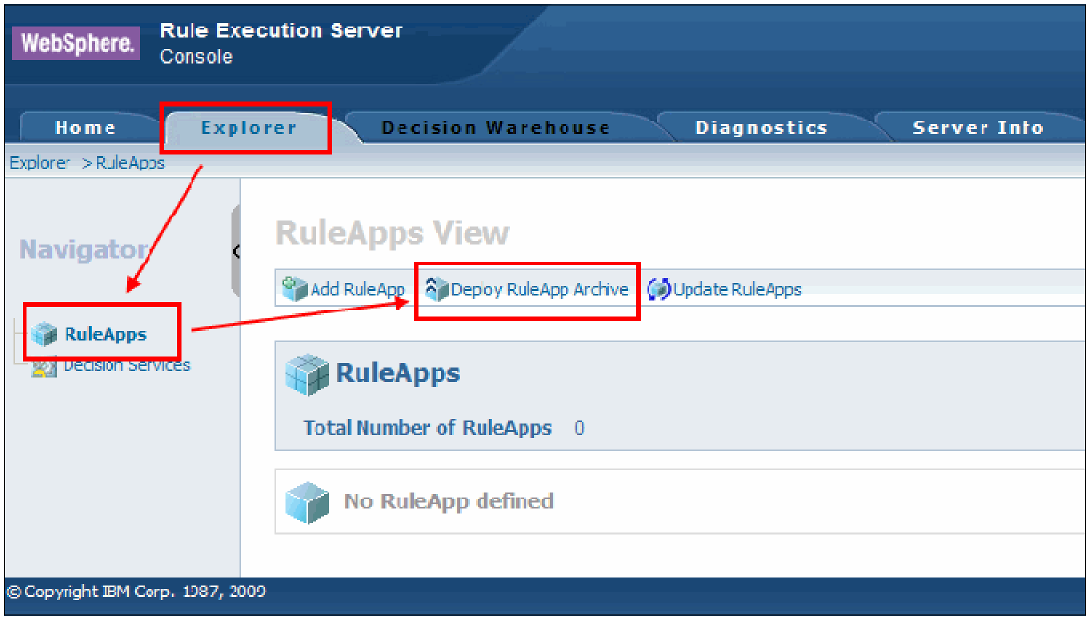
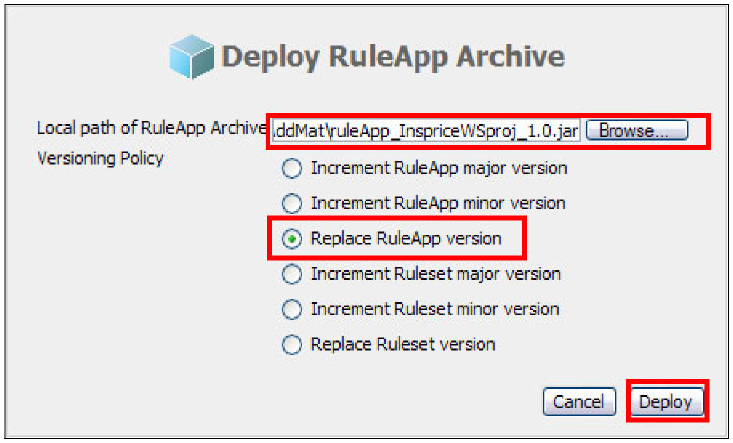


Figure 9-12 Finding the Deploy RuleApp Archive hyperlink

6. In the Deploy RuleApp Archive window, click **Browse** and select the `ruleApp_InspriceWSproj_1.0.jar` file from the path on your local workstation.
7. Click **Replace RuleApp version** and then click **Deploy**. Figure 9-13 shows an example of how to complete the Deploy RuleApp Archive window.


Deploy RuleApp Archive

Local path of RuleApp Archive: \ddMat\ruleApp_InspriceWSproj_1.0.jar [Browse...]

Versioning Policy:
- Increment RuleApp major version
- Increment RuleApp minor version
- [✓ Replace RuleApp version]
- Increment Ruleset major version
- Increment Ruleset minor version
- Replace Ruleset version

Figure 9-13 The Deploy RuleApp Archive panel

A RuleApp deployment report is produced. It indicates that the archive has been successfully deployed.

8. View the report and then click **OK** to return to the standard RuleApps View.

9.2.2 Creating the required CICS and z/FS resources

This section describes how to create the resources required on CICS and z/FS to run the demonstration.

Uploading the WSBind file

We use FTP to send the WSBind (`InspriceWS.wsbind`) file to the mainframe; we use the standard Windows® FTP client. The file must be installed in a directory on z/FS that can be accessed by CICS. In our example of setting up this demonstration, we upload the WSBind file to the following directory:

```
/u/cicsrs5/inspricews/wsbind
```

Tips:

- On the FTP of the WSBind file the representation type must be binary. If the file is uploaded in ASCII it will become corrupted and the web service invocation will fail.
- Make a note of the path that is used to upload the WSBind file; this path is required when updating the JCL that is used to define the CICS resources.

Uploading the JCL and load library

To upload the JCL and Load library data sets to the mainframe, we again use the Windows FTP client:

1. On the client, change the directory to the user ID’s high-level qualifier:

   ```
   cd ‘CICSRS5’
   ```

2. Change the FTP representation type to binary. Type in the following command:

   ```
   quote site recfm=fb lrecl=80 blksize=3120
   ```

3. Upload of the JCL data set by using the following command:

   ```
   put ILOGRED.JCL.XMIT
   ```

4. Log on to TSO and use the following TSO command:

   ```
   RECEIVE INTO INDA(‘CICSRS5.ILOGRED.JCL.XMIT’)
   ```

5. A data set named `CICSRS5.ILOGRED.JCL` is created on the system. Upload of the Load Library by using the following command:

   ```
   put ILOGRED.LOAD.XMIT
   ```

6. On TSO, enter the following command:

   ```
   RECEIVE INTO INDA('CICSRS5.ILOGRED.LOAD.XMIT')
   ```

   A data set called `CICSRS5.ILOGRED.LOAD` is created on the system.

Modifying, creating, and installing the CICS resources

The `CICSRS5.ILOGRED.JCL` file contains a CSD upgrade job named `CSD`. We modify this job so that we can create the required CICS resources on our CICS system:

1. In the CSD job, modify the CICSHLQ and CSD variables so that they point to the correct CICS data sets and CSD. Example 9-7 shows our CICSHLQ and CSD parameters after we have modified them.

   Example 9-7 The updated CICSHLQ and CSD parameters

   ```
   SET CICSHLQ='CICSTS320.CICS'
   SET CSD='CICSWRK1.TEAM6.CICS320.DFHCSD'
   ```

2. Update several of the resource definitions. The job contains a DEFINE PIPELINE command. In this command, update the `@CONFIGPATH@` section of the CONFIGFILE parameter to point to the location of the following configuration file:

   ```
   basicsoap11requester.xml
   ```

   In the same resource, update the `@WSDIRPATH@` part of the WSDIR parameter to point to the directory on z/FS that was uploaded to the WSBind file. Example 9-8 shows our updated pipeline definition. The parts that are updated are in bold.

   Example 9-8 The updated pipeline definition

   ```
   DEFINE PIPELINE(INSPIPE) GROUP(INSDEMO)
   CONFIGFILE(/usr/lpp/cicsts/cicsts32/samples/pipelines/basicsoap11req
   uester.xml)
   WSDIR(/u/cicsrs5/inspricews/wsbind)
   ```

3. The next parameter to be updated is the HOST parameter on the DEFINE URIMAP command. This parameter must be changed to the host name and port number of the Rule Execution Server.

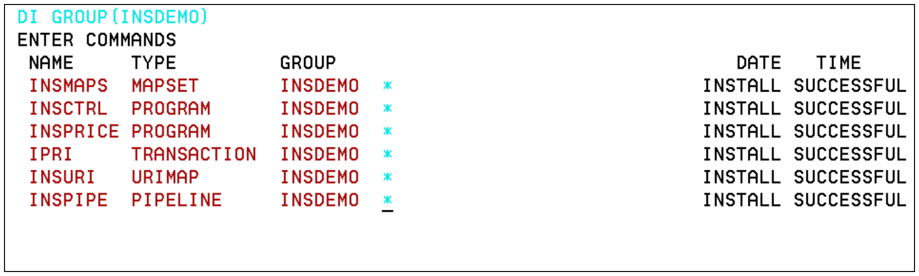
Example 9-9 shows the updated definition with the host parameter.

Example 9-9 The updated URI map definition

```
DEFINE URIMAP(INSURI) GROUP(INSDEMO)
       USAGE(CLIENT)
       HOST(wtsc.itso.ibm.com:9127)
       PATH(DecisionService/ws/InspriceWSproj/1.0/InspriceWS/1.0)
```

4. After saving these updates, submit the job. After the job completes, you can see that the resources have been created on CSD. Use the CEDA transaction to install the resources on the CICS system, as shown in Figure 9-14.

> **Note:** The CICS region must have access to the `CICSRS5.ILOGRED.LOAD` load library. To provide access, either add the library into the RPL concatenation in the JCL for our CICS region and restart the job, or create a CICS Library resource to get access to the load library while your CICS region is still running.



```
DI GROUP(INSDEMO)
ENTER COMMANDS
 NAME     TYPE         GROUP                                   DATE   TIME
 INSMAPS  MAPSET       INSDEMO  *                           INSTALL SUCCESSFUL
 INSCTRL  PROGRAM      INSDEMO  *                           INSTALL SUCCESSFUL
 INSPRICE PROGRAM      INSDEMO  *                           INSTALL SUCCESSFUL
 IPRI     TRANSACTION  INSDEMO  *                           INSTALL SUCCESSFUL
 INSURI   URIMAP       INSDEMO  *                           INSTALL SUCCESSFUL
 INSPIPE  PIPELINE     INSDEMO  *                           INSTALL SUCCESSFUL
```

Figure 9-14 Installing the resources in the INSDEMO CSD group

5. You can now run the application by starting transaction IPRI. Every time you press Enter to get a price for a vehicle, a web service call is being made to the Rule Execution Server to determine the price of an insurance quote.

Summary

This section has described how to quickly set up and run the sample CICS application provided in the Additional Materials section of the Redbooks website. In 9.3.4, “Modifying the CICS pricing application” on page 224, we explain how we modified the original CICS pricing application to use the Rule Execution Server web service.

9.3 Integration with CICS web services call to ILOG custom web services

In this scenario, we import a rule project into Rule Studio. The project has already been created for us. This project is a JRules implementation of the project that was described in 9.1.6, “Pricing application rule project” on page 202.

The business object model (BOM) for the rule project is derived from an XSD schema. The schema design tries to remain as close as possible to the COBOL CA-QUOTE-INPUT and CA-QUOTE-OUTPUT data structures that are defined in Example 9-1 on page 197 and Example 9-2 on page 198.

The schema that is used to create the project BOM is in Example 9-10.

Example 9-10 The schema used to create the web service project BOM

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://www.ilog.com/insdemo"
xmlns="http://www.ilog.com/insdemo"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:complexType name="QuoteRequest">
      <xs:sequence>
         <xs:element name="VehicleDetails" type="Vehicle"/>
         <xs:element name="Deductible" type="xs:float"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Quote">
      <xs:sequence>
         <xs:element name="Premium" type="xs:float" minOccurs="0"/>
         <xs:element name="Reason" type="xs:string" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
   <xs:complexType name="Vehicle">
      <xs:sequence>
         <xs:element name="VehicleIdentificationNumber"
type="xs:string"/>
         <xs:element name="VehicleType">
            <xs:simpleType>
               <xs:restriction base="xs:string">
                  <xs:enumeration value="A"/>
                  <xs:enumeration value="B"/>
                  <xs:enumeration value="C"/>
                  <xs:enumeration value="D"/>
                  <xs:enumeration value="E"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:element>
         <xs:element name="Manufacturer" type="xs:string"/>
         <xs:element name="Model" type="xs:string"/>
         <xs:element name="Value" type="xs:float"/>
      </xs:sequence>
   </xs:complexType>
</xs:schema>
```

After we generate a rule application (and deploy this application to a Rule Execution Server that runs on WebSphere Application Server for z/OS), we get the WSDL for the HTDS web service for the rule application. We use this WSDL to create an outbound web service call from the INSCTRL program in the CICS pricing application, described previously. The output that the HTDS returns to INSCTRL will match the output returned by the current COBOL implementation of the INSPRICE program, described previously.

In this scenario, we use the following software:

- WebSphere ILOG JRules Rule Execution Server version 7.0.3 deployed on WebSphere Application Server for z/OS version 7
- CICS Transaction Server version 3.2
- WebSphere ILOG JRules Rule Studio version 7.0.3

9.3.1 Importing rule project, deploying to RES, and generating WSDL

In this section, we look at importing the rule project, deploying to RES, and generating a WSDL file.

Importing the rule project

Note: We start this task with an archive file of the rule project on our workstation drive. You may download this archive file (`InspriceWS.zip`) from the Additional Materials section of the Redbooks website, which is explained in Appendix A, "Additional material" on page 359.

Perform the following steps to import the rule project:

1. Start Rule Studio to import the rule project.
2. In the Rule Studio, click **File** → **Import**. In the Import wizard, open the General folder, select **Existing Projects Into Workspace,** and click **Next**, as shown in Figure 9-15.

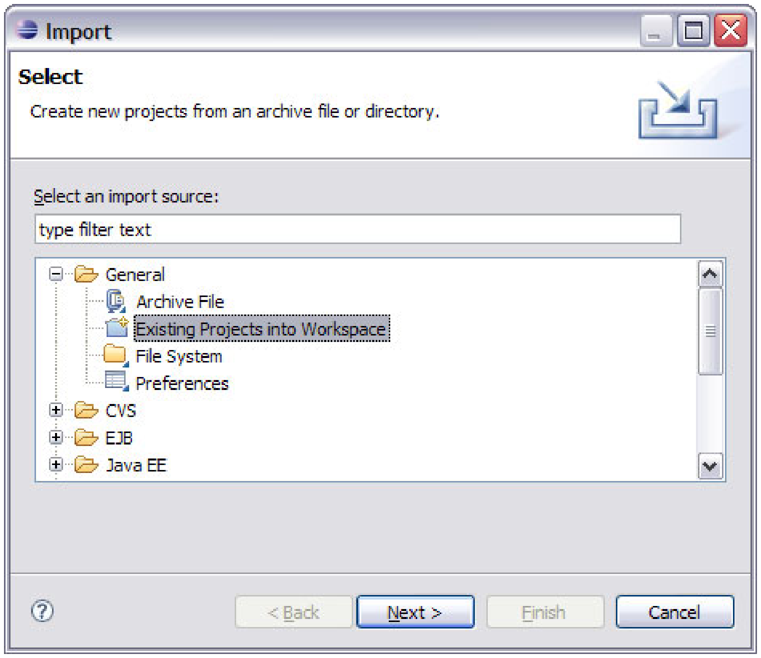
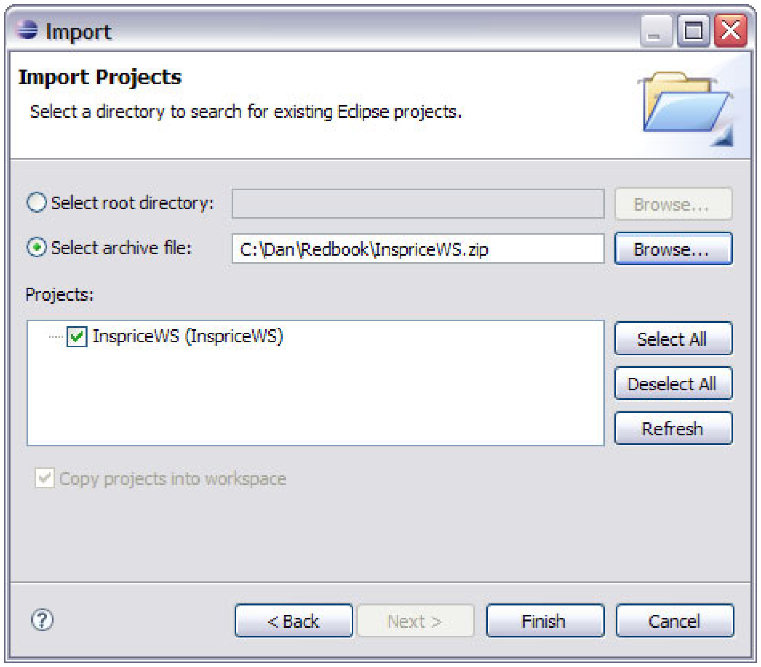


Figure 9-15 The Import wizard source selection window

3. Click **Select archive file** (Figure 9-16 on page 215) and enter or browse to the path to your version of the `InspriceWS.zip` file. Click **Finish**.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory:

[✓ Select archive file:] C:\Dan\Redbook\InspriceWS.zip

Projects:

✓ InspriceWS (InspriceWS)

✓ Copy projects into workspace

Figure 9-16 Importing the InspriceWS project

After the import completes, the `InspriceWS` rule project appears in the Rule Explorer window. At this point, you can explore the BOM, rules, and rule flows that make up the rule project.

However, instead of exploring the project, we generate a rule application and deploy it to the Rule Execution Server.

Creating a rule application

In this section, we create a rule application from the rule project that we just imported. After the rule application has been created, we deploy it to the RES running on our mainframe.

Perform the following steps:

1. To create the rule application, select the **InspriceWS** project from the Rule Explorer in Rule Studio. Look at the bottom of the Rule Studio window in the Rule Project Map view. Scroll to the right until you see the Deploy and Integrate box on the Rule Project Map, as shown in Figure 9-17.

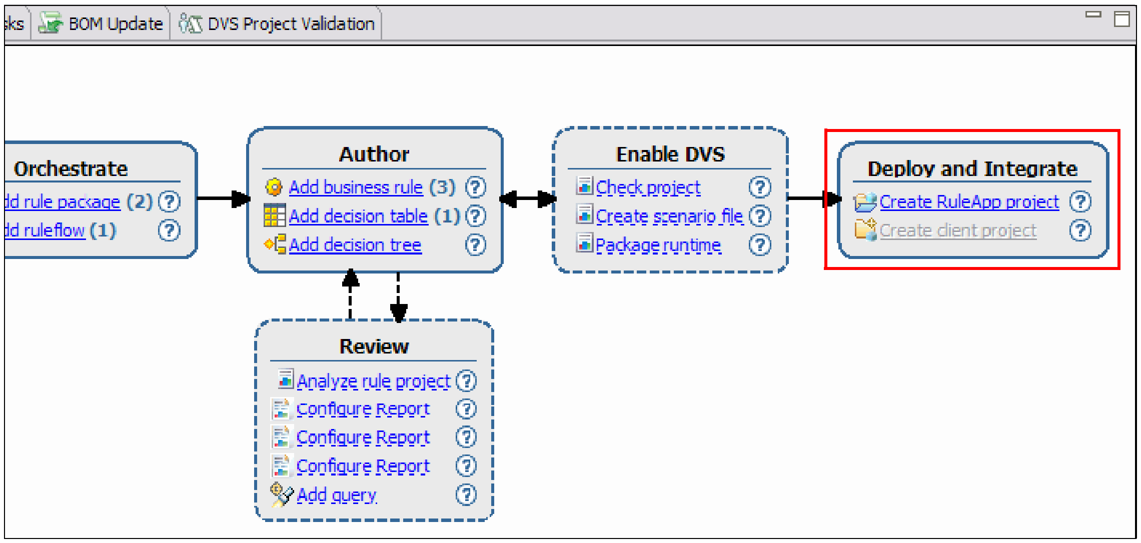


Figure 9-17 The Deploy and Integrate section of the Rule Project Map

Tip: If you cannot see the Rule Project Map, use the menu bar to click **Window** → **Show View** → **Rule Project Map**.

2. In the Deploy and Integrate box of the project map, select the **Create RuleApp project** link.
3. In the New RuleApp Project dialog window, enter `InspriceWSproj` as the project name and click **Finish**. A rule application called InspriceWSproj appears in the Rule Explorer.

Deploying the rule application

Perform the following steps:

1. To deploy the rule application, right-click the **InspriceWSproj** rule application in the Rule Explorer.
2. From the menu, click **RuleApp** → **Deploy**, as shown in Figure 9-18.

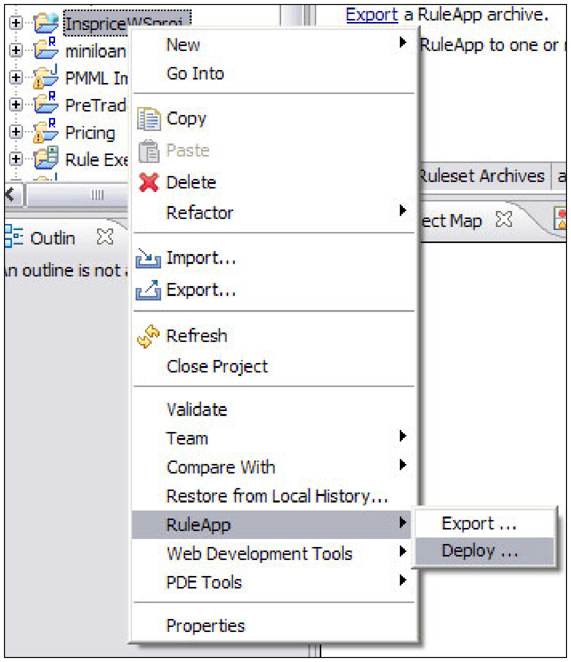
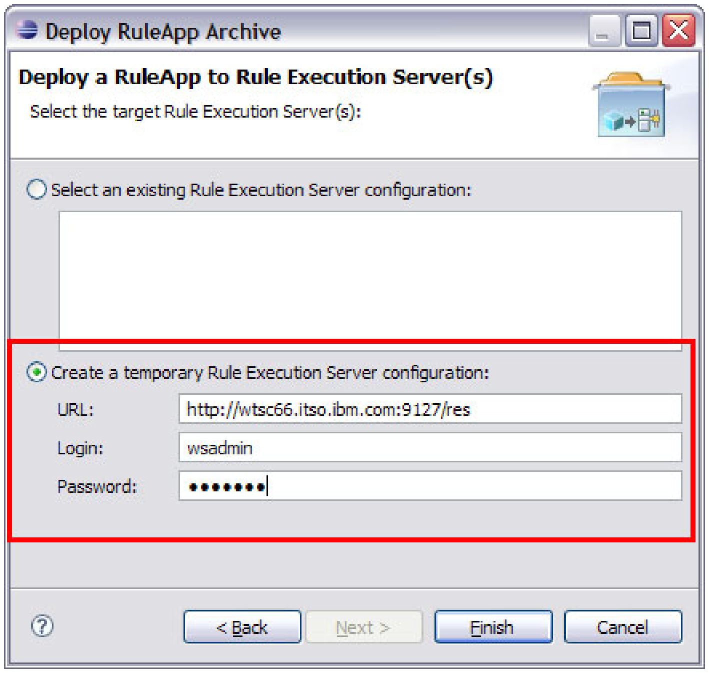


Figure 9-18 Selecting the Deploy function for the rule application

3. The Deploy RuleApp Archive window opens. On the first window, click **Replace RuleApp version** and then click **Next**.

4. Click **Create a temporary Rule Execution Server configuration** and enter the URL of your RES, and your user ID and password for that RES, as shown in Figure 9-19.


Deploy RuleApp Archive
Deploy a RuleApp to Rule Execution Server(s)
Select the target Rule Execution Server(s):
Select an existing Rule Execution Server configuration:
< Back
Next >
Finish
Cancel

Create a temporary Rule Execution Server configuration:

URL: http://wtsc66.itso.ibm.com:9127/res

Login: wsadmin

Password: •••••••

Figure 9-19 Entering the login details for the RES

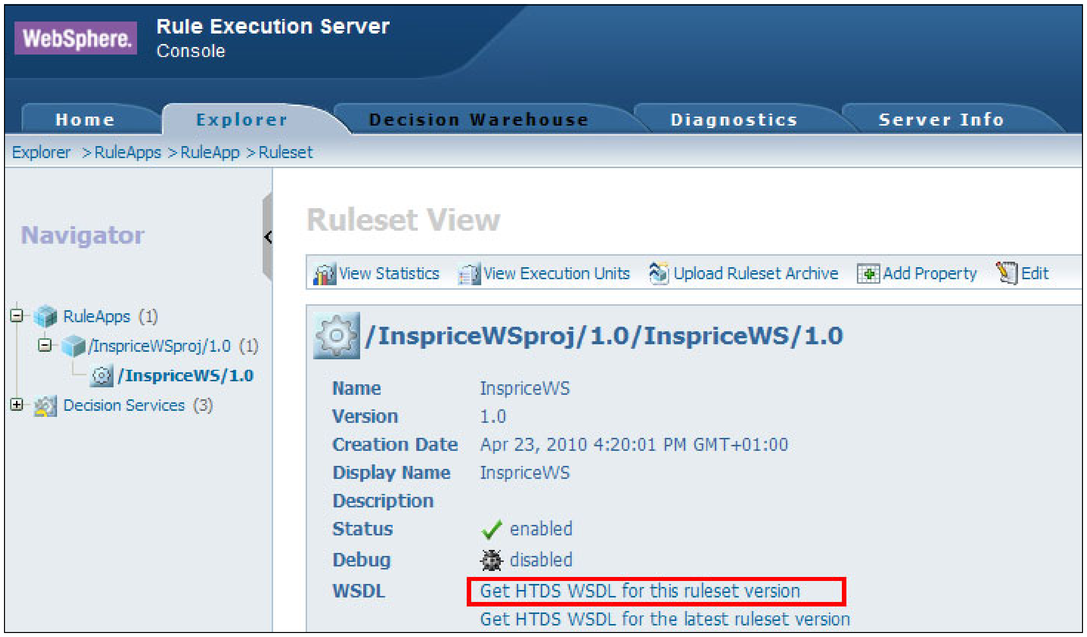
5. Click **Finish.** The deployment of the rules worked and success messages are written to the Rule Studio Console.

Generating the rule application WSDL

Perform the following steps:

1. To generate the WSDL that allows you to access the rules through web services, log on to our Rule Execution Server.
2. Select the **Explorer** tab. In the Navigator section of the console, expand **RuleApps** → **/InspriceWSproj/1.0**, → **/InspriceWS/1.0**. The main Ruleset View frame displays the `InspriceWS` application.

3. To generate the WSDL for the web service, click **Get HTDS WSDL for the ruleset version** in the Ruleset View, as shown in Figure 9-20.



/InspriceWSproj/1.0/InspriceWS/1.0

Name	InspriceWS
Version	1.0
Creation Date	Apr 23, 2010 4:20:01 PM GMT+01:00
Display Name	InspriceWS
Description	
Status	enabled
Debug	disabled
WSDL	Get HTDS WSDL for this ruleset version
	Get HTDS WSDL for the latest ruleset version

Figure 9-20 Position of the Get HTDS WSDL hyperlink

4. A new browser window opens that contains the WSDL for the rule application. Save this WSDL to your workstation by clicking **File** → **Save as** on the web browser menu. Save the file as `InspriceWS.wsdl`.

Tip: The first time we uploaded our WSDL to the mainframe, we found that it did not maintain its formatting. We found that by opening the WSDL file in WordPad on Windows and clicking **Save** preserved the file formatting during the upload to the mainframe.

9.3.2 Creating the CICS WSBind file

In this section, we create the CICS WSBind file.

Uploading the WSDL to the mainframe

Tip: In this scenario, the FTP client must convert the WSDL file from ASCII to EBCDIC when the file is put on the mainframe. From our experience, we have found that not all FTP clients can handle this data conversion correctly. If an application on the mainframe fails to read a file it can sometimes be because the FTP client has failed to correctly convert to EBCDIC.

The two FTP clients that we have used successfully are the standard FTP client that is included with the Windows operating system and the FTP client that is included as part of IBM Rational® Developer for z.

To transfer the COBOL programs to the mainframe, we used the standard FTP client that is included with Windows. We use FTP to transfer the WSDL to zFS (using the `/u/cicsrs5/inspricews/wsdl` path). We uploaded the file using the FTP session shown in Example 9-11.

Example 9-11 FTP session to upload the WSDL to the mainframe

```
ftp> open
To wtsc66oe.itso.ibm.com
Connected to wtsc66oe.itso.ibm.com.
220-FTP Server (user ‘donnelly@uk.ibm.com’)
220
User (wtsc66oe.itso.ibm.com:(none)): cicsrs5
331-Password:
331
Password:
230-220-FTPOE1 IBM FTP CS V1R10 at wtsc66oe.itso.ibm.com, 16.56:17 on
2010-04-23.
230-CICSRS5 is logged on. Working directory is “/u/cicsrs5”.
230
ftp> cd inspricews/wsdl
250 HFS directory /u/cicsrs5/inspricews/wsdl is the current working directory
ftp> put InspriceWS.wsdl
200 Port request OK.
125 Storing data set /u/cicsrs5/inspricews/wsdl/InspriceWS.wsdl
250 Transfer completed successfully.
ftp: 4972 bytes sent in 0.00Seconds 4972000.00Kbytes/sec.
ftp> close
221 Quit command received. Goodbye.
```

Modifying and running the CICS web services assistant

Before we run the DFHWS2LS procedure to generate the COBOL data structure to call our rules, we update the `InspriceWS` WSDL file. We do this because the WSDL file that is generated by Rules Execution Server specifies that the XML is encoded in a UTF-8 codepage:

```
<?xml version="1.0" encoding="UTF-8"?>
```

After the file is uploaded to the mainframe, it is converted to an EBCDIC codepage. To correct the file, we remove the encoding section from the tag as follows:

```
<?xml version="1.0"?>
```

To run the CICS web services assistant, we submit JCL similar to that shown in Example 9-12.

Example 9-12 The CICS web services assistant JCL

```
//WS2LS1  JOB (276702),MSGCLASS=H,CLASS=A,NOTIFY=CICSRS5,REGION=0M
//*
//JOBPROC JCLLIB ORDER=CICSTS32.CICS.SDFHINST
//*
//WS2LS    EXEC DFHWS2LS,
//     JAVADIR='java/J1.5',
//     USSDIR='cicsts32',
//     PATHPREF=''
//INPUT.SYSUT1 DD *
 PDSLIB=CICSRS5.ILOG.COBOL
 LANG=COBOL
 WSDL=/u/cicsrs5/inspricews/wsdl/InspriceWS.wsdl
 WSBIND=/u/cicsrs5/inspricews/wsbind/InspriceWS.wsbind
 LOGFILE=/u/cicsrs5/inspricews/InspriceWS.log
 REQMEM=WEBIN
 RESPMEM=WEBOUT
 CHAR-VARYING=YES
 MAPPING-LEVEL=1.2
/*
//*
```

Upon completion of the job, two warnings are generated (Example 9-13).

Example 9-13 JCL warnings

```
DFHPI9586W A reserved word "Value" has been detected in the input document, it
has been changed to "XValue".
DFHPI9586W A reserved word "Quote" has been detected in the input document, it
has been changed to "XQuote".
```

The warnings are generated because the variable that holds the vehicle's value has been defined as `Value` and the quote has been defined as `Quote`. This definition causes a clash in COBOL because the terms `Value` and `Quote` are reserved words in COBOL. We accept the terms `XValue` and `XQuote` as a substitution.

We can now examine the data structures that the CICS web services assistant generated. Example 9-14 and Example 9-15 show the input and output data structures we use when calling the web service from our COBOL program.

Example 9-14 The COBOL input data structure for the web service

```
05 DecisionServiceRequest.
  10 quoteRequest1.
    15 quoteRequest.
      20 VehicleDetails.
        25 VehicleIdentificatio-length   PIC S9999 COMP-5
 SYNC.
        25 VehicleIdentificationNumber   PIC X(255).
        25 VehicleType-length            PIC S9999 COMP-5
 SYNC.
        25 VehicleType                   PIC X(1).
        25 Manufacturer-length           PIC S9999 COMP-5
 SYNC.
        25 Manufacturer                  PIC X(255).
        25 Model-length                  PIC S9999 COMP-5
 SYNC.
        25 Model                         PIC X(255).
        25 XValue                        COMP-1 SYNC.
      20 Deductible                    COMP-1 SYNC.
```

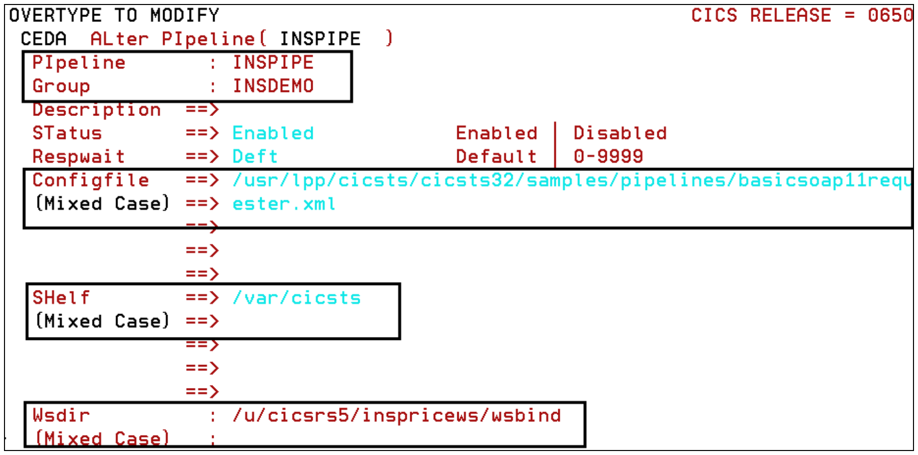
Example 9-15 The COBOL output data structure for the web service

```
05 DecisionServiceResponse.
  10 ilogXrulesXoutputStr-length   PIC S9999 COMP-5 SYNC.
  10 ilogXrulesXoutputString       PIC X(255).
  10 ilogXrulesXfiredRulesCount    PIC S9(9) COMP-5 SYNC.
  10 theQuote.
    15 XQuote.
      20 Premium                       COMP-1 SYNC.
      20 Reason-length                 PIC S9999 COMP-5
 SYNC.
      20 Reason                        PIC X(255).
```

Now that we have generated the CICS WSBind file and the COBOL data structures that are used to call the web service, we can create the CICS resources that are required to issue an outbound web service call.

9.3.3 Creating the required CICS resources

We now must define and install a CICS pipeline to enable us to call the JRules web service. With the pipeline, we use the `basicsoap11requester.xml` configuration file that is provided as a CICS sample. We set the default of the pipeline shelf to `/var/cicsts`, and set the web services binding directory to the `/u/cicsrs5/inspricews/wsbind` directory that we defined to hold our WSBind files. We let all other attributes in the pipeline resource default. Figure 9-21 shows an example of our pipeline resource with the relevant parts highlighted.



```
OVERTYPE TO MODIFY                                            CICS RELEASE = 0650
 CEDA  ALter PIpeline( INSPIPE  )
  PIpeline         : INSPIPE
  Group            : INSDEMO
  Description  ==>
  STatus       ==> Enabled            Enabled | Disabled
  Respwait     ==> Deft               Default | 0-9999
  Configfile   ==> /usr/lpp/cicsts/cicsts32/samples/pipelines/basicsoap11requ
  (Mixed Case) ==> ester.xml
               ==>
               ==>
               ==>
  SHelf        ==> /var/cicsts
  (Mixed Case) ==>
               ==>
               ==>
               ==>
  Wsdir            : /u/cicsrs5/inspricews/wsbind
  (Mixed Case)     :
```

Figure 9-21 The pipeline definition for our web service call

We can now install the pipeline and then check the CICS MSGUSR log to see whether the pipeline installation and scan happened successfully. The messages from our installation can be seen in Example 9-16.

Example 9-16 The pipeline installation messages

```
TC10     CEDA CICSUSER 04/26/10 09:27:10 INSTALL PIPELINE(INSPIPE)
GROUP(INSDEMO)

DFHPI0703 I 04/26/2010 09:27:10 SCSCMAS6 CICSRS5 PIPELINE INSPIPE is
about to scan the WSDIR directory.

DFHPI0901 I 04/26/2010 09:27:10 SCSCMAS6 CICSRS5 New WEBSERVICE
InspriceWS is being created during a scan against PIPELINE INSPIPE.
```

```
DFHPI0910 I 04/26/2010 09:27:10 SCSCMAS6 CICSRS5 WEBSERVICE InspriceWS
within PIPELINE INSPIPE has been created.

DFHPI0915 I 04/26/2010 09:27:10 SCSCMAS6 CICSRS5 WEBSERVICE InspriceWS
is now INSERVICE and is ready for use.

DFHPI0704 I 04/26/2010 09:27:10 SCSCMAS6 CICSRS5 PIPELINE INSPIPE
Implicit scan has completed. Number of wsbind files found in the
          WSDIR directory: 000001. Number of successful WEBSERVICE
creates: 000001. Number of failed WEBSERVICE creates: 000000.
```

We see from the messages in the log that the `InspriceWS` web service has been created and is ready to use. We now modify our pricing application to call the new web service.

9.3.4 Modifying the CICS pricing application

To minimize the changes that we must make to our calling program (INSCTRL), we create a web service calling program that has the same name and uses the same channel and containers as the original INSPRICE program:

1. In this program, we take the quote request data out of the INS-IN-CONT container.
2. We then copy the information to the `DecisionServiceRequest` COBOL input structure.
3. This input structure is then put into the DFHWS-DATA container and the web service `executeDecisionService` is invoked.
4. On the return from the web service, we get the response from the DFHWS-DATA container and put it into the `DecisionServiceResponse` COBOL output structure.
5. We copy the information from the `DecisionServiceResponse` structure and add it to the CA-QUOTE-OUTPUT structure.
6. We add this structure to the INS-OUT-CONT container and return control to the INSCTRL program. Figure 9-22 on page 225 shows the flow of control in our new INSPRICE program.

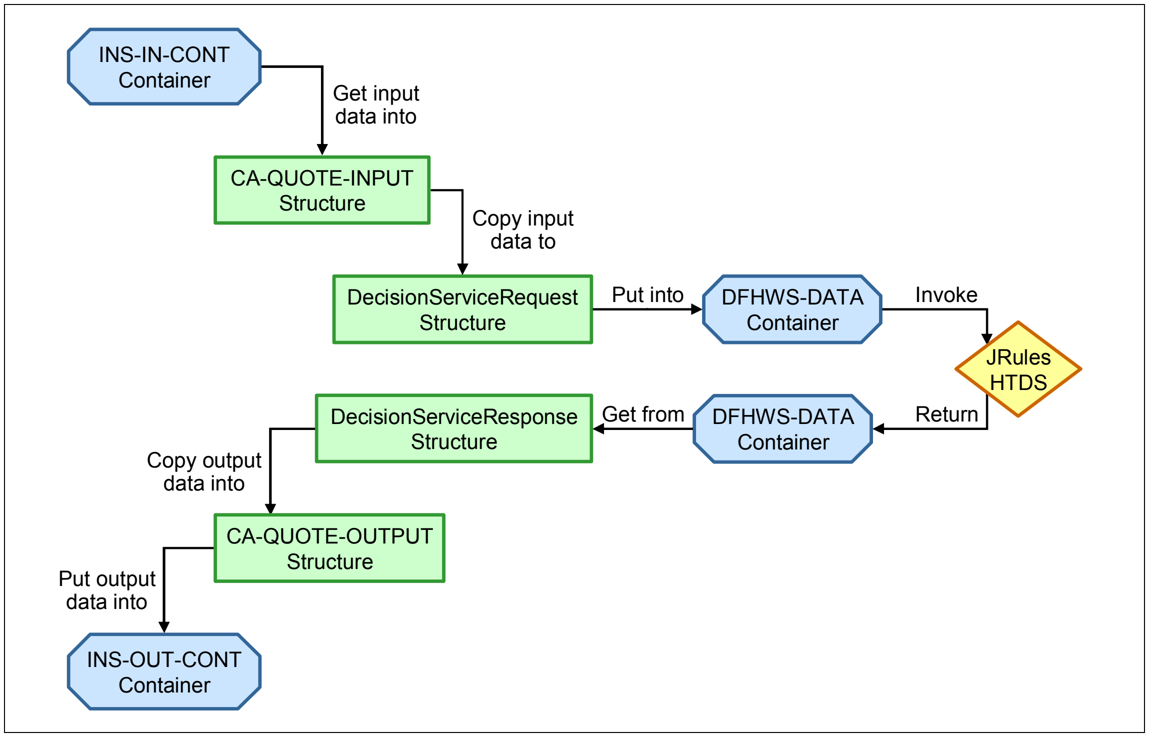


Figure 9-22 Flow of control in the new INSPRICE program

Example 9-17 shows the COBOL code that implements this flow of control.

Example 9-17 The updated web service version of INSPRICE

```
CBL CICS('COBOL3') APOST
IDENTIFICATION DIVISION.
PROGRAM-ID. INSPRICE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 COUNTER                      PIC 9(5).
01 TEMP-STR                     PIC X(255).
01 TEMP-STR2                    PIC X(255).
01 DATA-LENGTH                  PIC S9999 COMP-5.

 Container structure copybook
COPY QUOTECPY.

01 INPUT-STRUCTURE.
```

```
  05 DecisionServiceRequest.
    10 quoteRequest1.
      15 quoteRequest.
        20 VehicleDetails.
     25 VehicleIdentificatio-length PIC S9999 COMP-5 SYNC.
          25 VehicleIdentificationNumber   PIC X(255).
          25 VehicleType-length   PIC S9999 COMP-5 SYNC.
          25 VehicleType                   PIC X(1).
      25 Manufacturer-length  PIC S9999 COMP-5 SYNC.
           25 Manufacturer                  PIC X(255).
           25 Model-length         PIC S9999 COMP-5 SYNC.
           25 Model                         PIC X(255).
           25 XValue                        COMP-1 SYNC.
         20 Deductible                    COMP-1 SYNC.

 01 OUTPUT-STRUCTURE.
   05 DecisionServiceResponse.
     10 ilogXrulesXoutputStr-length   PIC S9999 COMP-5 SYNC.
     10 ilogXrulesXoutputString       PIC X(255).
     10 ilogXrulesXfiredRulesCount    PIC S9(9) COMP-5 SYNC.
     10 theQuote.
       15 XQuote.
         20 Premium                       COMP-1 SYNC.
         20 Reason-length       PIC S9999 COMP-5 SYNC.
         20 Reason                        PIC X(255).

 01 COMMAND-RESP                PIC S9(8) COMP.
 01 COMMAND-RESP2               PIC S9(8) COMP.

 01 WEB-SERVICE-NAME PIC X(32) value 'InspriceWS'.
 01 WEB-CHANNEL      PIC X(16) value 'SERVICE-CHANNEL'.
 01 WEB-CONTAINER    PIC X(16) value 'DFHWS-DATA'.
 01 WEB-OPERATION    PIC X(255) value 'executeDecisionService'.
*------------------------------------------------------------------*
LINKAGE SECTION.

 PROCEDURE DIVISION.

 MAINLINE SECTION.

* Get the input container
     EXEC CICS GET CONTAINER(INS-IN-CONT)
                   INTO(CA-QUOTE-INPUT)
                   FLENGTH(LENGTH OF CA-QUOTE-INPUT)
                   CHANNEL(INSPRICE-CHAN)
                   END-EXEC

* Move data from our input structure to our web service
* input structure
```

```
     move CA-VEHICLE-ID-NUM to VehicleIdentificationNumber
     move CA-VEHICLE-ID-NUM to TEMP-STR
     perform CALCULATE-LENGTH
     move DATA-LENGTH to VehicleIdentificatio-length

     move CA-VEHICLE-TYPE to VehicleType
     move CA-VEHICLE-TYPE to TEMP-STR
      perform CALCULATE-LENGTH
     move DATA-LENGTH to VehicleType-length

     move CA-VEHICLE-MANUFACTURER to Manufacturer
     move CA-VEHICLE-MANUFACTURER to TEMP-STR
     perform CALCULATE-LENGTH
     move DATA-LENGTH to Manufacturer-length

     move CA-VEHICLE-MODEL to Model
     move CA-VEHICLE-MODEL to TEMP-STR
     perform CALCULATE-LENGTH
     move DATA-LENGTH to Model-length

     move CA-VEHICLE-VALUE to XValue

     move CA-DEDUCTIBLE to Deductible

* Put the input into the Web Service container
     EXEC CICS PUT CONTAINER(WEB-CONTAINER)
                   CHANNEL(WEB-CHANNEL)
                   FROM(INPUT-STRUCTURE)
                   RESP(COMMAND-RESP)
                   RESP2(COMMAND-RESP2)
                   END-EXEC

* Now call the pricing Web Service
     EXEC CICS INVOKE WEBSERVICE(WEB-SERVICE-NAME)
               CHANNEL(WEB-CHANNEL)
OPERATION(WEB-OPERATION)
               RESP(COMMAND-RESP)
               RESP2(COMMAND-RESP2)
               END-EXEC

* Get the result of the Web Service call
     EXEC CICS GET CONTAINER(WEB-CONTAINER)
                   CHANNEL(WEB-CHANNEL)
                   INTO(OUTPUT-STRUCTURE)
                   RESP(COMMAND-RESP)
                   RESP2(COMMAND-RESP2)
                   END-EXEC

* Move the response into the response data structure
```

```
     move Premium to CA-PREMIUM
     move Reason to CA-RESPONSE-MESSAGE

* Put the output into its container
     EXEC CICS PUT CONTAINER(INS-OUT-CONT)
                   FROM(CA-QUOTE-OUTPUT)
                   CHANNEL(INSPRICE-CHAN)
                   END-EXEC

     EXEC CICS RETURN END-EXEC

     EXIT.

* Calculate variable lengths
      move 0 to COUNTER
     move 0 to DATA-LENGTH
     move function reverse(TEMP-STR) to TEMP-STR2

     inspect temp-str2 tallying COUNTER for leading spaces

     move length of temp-str to DATA-LENGTH

     subtract COUNTER from DATA-LENGTH

     EXIT.
```

7. Finally, we perform a **NEWCOPY** command on the INSPRICE program to ensure we have the latest versions of the programs loaded. Example 9-18 shows the **CEMT** command used to achieve this.

Example 9-18 The NEWCOPY command

```
CEMT SET PROGRAM(INSPRICE) NEW
```

8. We start the application by starting the IPRI transaction.

The functionality of the application is the same as before; however, when we want to change our pricing rules, we can now modify them by deploying updated rules to the Rule Execution Server.

9.3.5 Summary

In this section, we demonstrated how business rules written in Rule Studio and deployed to a Rules Execution Server can be integrated into an existing application using JRules hosted transparent decision services and CICS web services support. By extracting our business rules from our COBOL application, we gain the following benefits:

- The basic pricing rules are now stored in a decision table. This table is easier to understand and update than the COBOL code.
- The rules can be published to Rule Team Server and shared with the business side of the organization. This means that people in the business side of the organization can directly change the rules rather than raising a code change request with IT.
- By deploying the rules to a Rule Execution Server, we are able to make changes to the business rules in our application without having to recompile or relink any application code. This way means our business rules can be changed when the business requires it, rather than having to wait for a new version of the application to be deployed.

9.4 ILOG Rules for COBOL

In this scenario, we import a rule project into Rule Studio. The project, which has already been created for us, is a Rules for COBOL implementation of the project that was described in section 9.1.6, “Pricing application rule project” on page 202. The BOM for the rule project was derived from the COBOL copybook that contained the definitions for the following COBOL data structures for the CICS pricing application:

- The CA-QUOTE-INPUT data structure is described in Example 9-1 on page 197.

  The input parameter to the rule project is based on this data structure
- The CA-QUOTE-OUTPUT data structure is described in Example 9-2 on page 198.

  The output parameter is based on this data structure.

After we import the rule project, we generate COBOL programs that implement the rule flow described in the project. We then use FTP to transfer this code up to the mainframe where our CICS pricing application is hosted. We make several modifications to our INSCTRL program to allow us to call the generated COBOL programs and then recompile the application. Finally, we modify the resources in our CICS system to ensure the application can execute correctly.

We use the following software in this scenario:

- CICS Transaction Server version 3.2
- WebSphere ILOG Rules for COBOL version 7.0.3

9.4.1 Importing rule project, generating COBOL code, using FTP to the mainframe

In this section, we look at importing the rule project, generating COBOL code, and using FTP.

Importing the rule project

Note: We start this task with an archive file of the Rules for COBOL rule project on our workstation drive. The archive file is called `InspriceCbl.zip` and can be downloaded from the Additional Materials section of the Redbooks website. See Appendix A, “Additional material” on page 359 for instructions.

Perform the following steps:

1. To import the Rules for COBOL project, start version of Rule Studio. Our version of Rule Studio has the Rules for COBOL plug-in already installed.
2. To start the import, click **File** → **Import** on the Rule Studio menu. The Import wizard opens.

3. Open the **General** folder, select **Existing Projects Into Workspace,** and click **Next**, as shown in Figure 9-23.

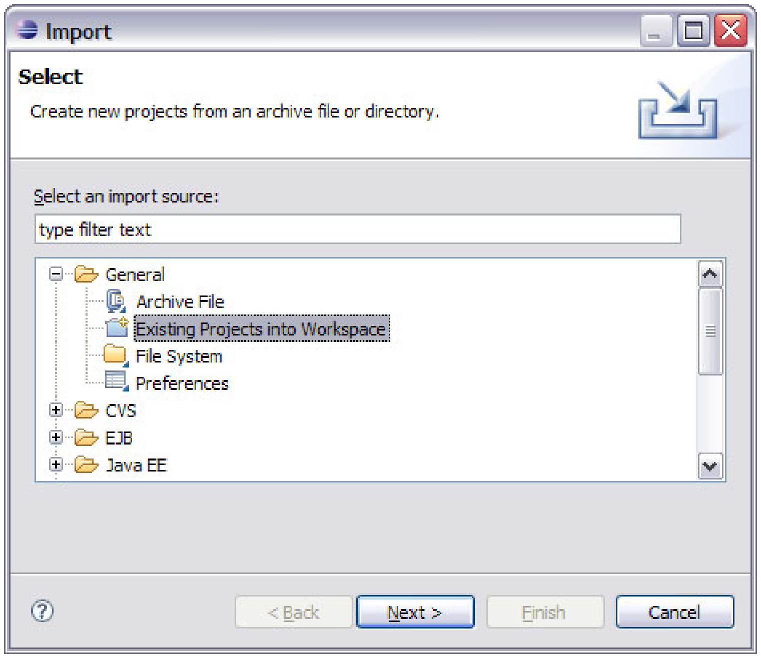


Figure 9-23 Select Existing Projects into workspace from the wizard

4. We select the **Select archive file** button, enter the path to our version of the `InspriceCbl.zip` file, and then click **Finish**, as shown in Figure 9-24.

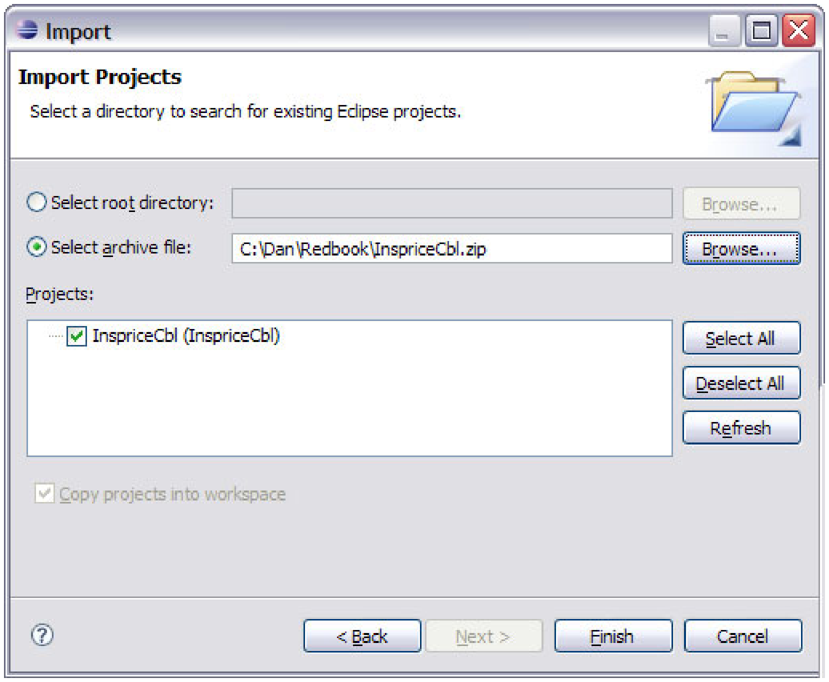


Figure 9-24 Enter the archive file path and click the finish button

After the import completes, the `InspriceCbl` rule project appears in the **Rule Explorer** window. At this point we could explore the BOM, rules, and rule flows that make up the rule project. Instead of exploring the project, we are going to generate our COBOL code.

Generating the COBOL code

Perform the following steps:

1. To generate COBOL code from the rule project, right-click the **InspriceCbl** project.
2. From the menu, select **Rules for COBOL** → **Generate COBOL code**, as shown in Figure 9-25 on page 233.

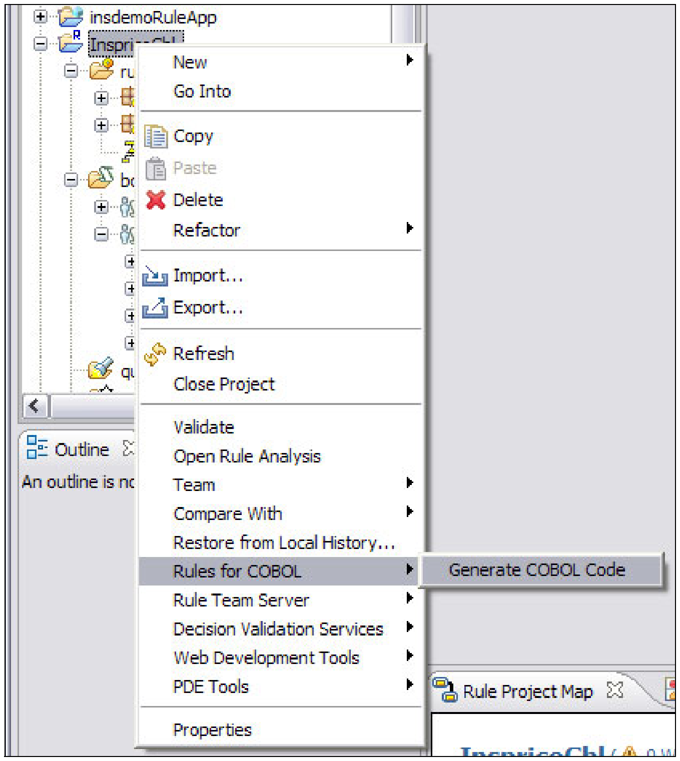
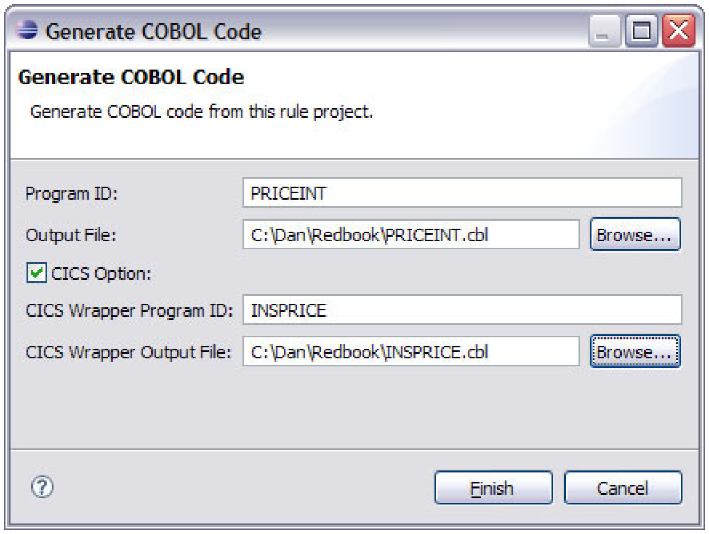


Figure 9-25 Selecting the Generate COBOL code menu option

3. In the Generate COBOL Code window, enter `PRICEINT` as the program identifier for the generated code, and enter a path to the folder on the workstation where you want the code to be stored, as shown in Figure 9-26 on page 234.
4. Select the **CICS Option** check box. The check box allows you to create a CICS wrapper program. With the CICS wrapper program, you can pass information to Rules for COBOL code through channels and containers.

   Normal Rules for COBOL code is either statically or dynamically linked to from the calling program. However, because we are passing information to and from our INSPRICE program through containers, we want to be able to do the same with our Rules for COBOL code. This way minimizes the changes that must be made to INSCTRL. To further minimize the changes to the INSCTRL code, we call the CICS wrapper program INSPRICE. This means that we do not have to change the EXEC CICS LINK statement in INSCTRL.

To define the name of the CICS wrapper program, enter `INSPRICE` into the CICS Wrapper Program ID field (see Figure 9-26) and enter the path to store the generated code into the CICS Wrapper Output File field. Click **Finish** to generate the COBOL programs.


Generate COBOL Code

Generate COBOL Code

Generate COBOL code from this rule project.

Program ID: PRICEINT

Output File: C:\Dan\Redbook\PRICEINT.cbl

[✓] CICS Option:

CICS Wrapper Program ID: INSPRICE

CICS Wrapper Output File: C:\Dan\Redbook\INSPRICE.cbl

Figure 9-26 The Generate COBOL Code dialog

Note: Certain Rule Project properties must be set to successfully create a CICS wrapper program. In the project that we have imported, these properties are already set. You can see these properties by right-clicking the Rule Project and selecting **Properties**. In the window that is displayed select **Rules for COBOL**, then select the **CICS** tab. In this window, you see that a channel name of INSPRICE-CHAN has been set for the project. You also see that the input rule parameter has been mapped to a container named INS-IN-CONT, and the output parameter has been mapped to a container named INS-OUT-CONT.

Uploading the COBOL to the mainframe

To use FTP to transfer the COBOL programs to the mainframe, we used the standard FTP client that is included with Windows. We used FTP for sending both the INSPRICE and PRICEINT COBOL programs to a partitioned data set on the mainframe.

Tip: In this scenario, the FTP client must convert the COBOL file from ASCII to EBCDIC when it is put on the mainframe. From our experience we have found that not all FTP clients can handle this data conversion correctly. If an application on the mainframe fails to read a file it can sometimes be because the FTP client has failed to correctly convert to EBCDIC.

The two FTP clients that we have used successfully are the standard FTP client that comes with the Windows operating system and the FTP client that comes as part of Rational Developer for z.

9.4.2 Modifying and compiling the application code

We now must perform minor modifications to our INSCTRL program so that it correctly calls the Rules for COBOL generated code. Example 9-19 shows how INSCTRL currently calls our existing INSPRICE program.

Example 9-19 INSCTRL call to the INSPRICE program

```
CALL-PRICE-PROG.

*    Put the pricing program input into its container
     EXEC CICS PUT CONTAINER(INS-IN-CONT)
                   FROM(CA-QUOTE-INPUT)
                   CHANNEL(INSPRICE-CHAN)
                   END-EXEC

     EXEC CICS LINK PROGRAM ('INSPRICE')
                    CHANNEL (INSPRICE-CHAN)
                    END-EXEC
```

We must add a CICS command, PUT CONTAINER, for our INS-OUT-CONT container. We do this because the generated CICS wrapper program expects all Rule Project containers to be passed to it. The CICS wrapper program also passes back both containers when it returns to the calling program.
Example 9-20 on page 236 shows our call to `INSPRICE` after we have added the new PUT CONTAINER command.

Example 9-20 PUT CONTAINER command in an INSPRICE call

```
CALL-PRICE-PROG.

*    Put the pricing program input into its container
     EXEC CICS PUT CONTAINER(INS-IN-CONT)
                   FROM(CA-QUOTE-INPUT)
                   CHANNEL(INSPRICE-CHAN)
                   END-EXEC

*    Add the container that the R4C program expects
     EXEC CICS PUT CONTAINER(INS-OUT-CONT)
                   FROM(CA-QUOTE-OUTPUT)
                   CHANNEL(INSPRICE-CHAN)
                   END-EXEC

     EXEC CICS LINK PROGRAM ('INSPRICE')
                    CHANNEL (INSPRICE-CHAN)
                    END-EXEC
```

Attention: Because of a known bug in the generated CICS wrapper code, we perform minor modifications to the INSPRICE program, which we uploaded to the mainframe. The main issue with the CICS wrapper program is that it uses a constant in the CALL statement that is never defined. In our example, the CALL statement is as follows:

```
CALL PRICEINT USING CA-QUOTE-INPUT CA-QUOTE-OUTPUT.
```

We had to define the `PRICEINT` constant in the working storage section of the INSPRICE program as follows:

```
01 PRICEINT PIC X(8) VALUE 'PRICEINT'.
```

We also had to perform another modification to the CICS wrapper program because the CALL statement and the EXEC CICS statements that follow it are created in the wrong column. We found these statements were generated in column 8. We had to edit the code to move these statements to column 12.

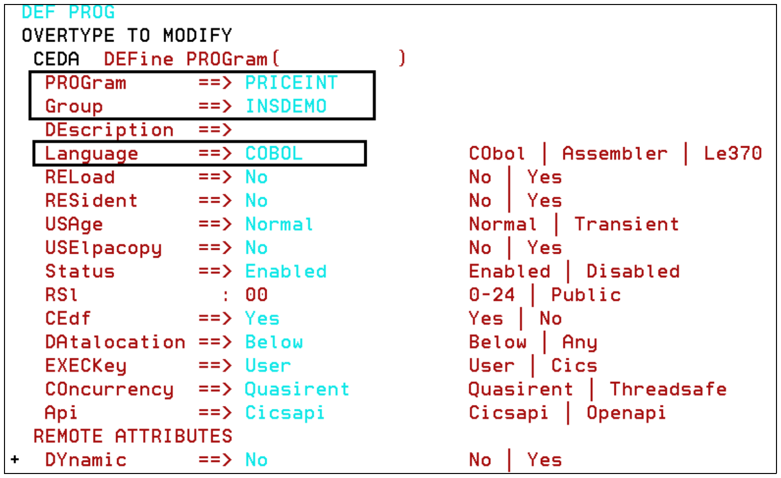
After performing the required code modifications, we can recompile the INSCTRL and INSPRICE program by using our standard COBOL compile JCL. Because the Rules for COBOL generated program (PRICEINT) is dynamically linked to from the INSPRICE program, we must also compile it as though it were a stand-alone COBOL program.

Important: The job that is used to compile the Rules for COBOL generated program (in our case PRICEINT) *must not* have the CICS translator option specified. If this option is specified, the compile program will expect a DFHCOMMAREA to be passed to it as the first parameter. Because the CICS wrapper program does not pass a dummy DFHCOMMAREA in this situation, the PRICEINT program will fail with an ASRA abend.

After recompilation of the three programs (INSCTRL, INSPRICE, and PRICEINT), we log on to the CICS system and modify it to run our updated application.

9.4.3 Modifying the CICS systems resources

In the previous sections, we generate and compile a new program called PRICEINT. We now define this program to the CICS system. Using CEDA, we define the program to our INSDEMO group in which our remaining application definitions are stored. We specify that it is a COBOL program and then let the rest of the program attributes default. Figure 9-27 shows the PRICEINT program definition with the relevant parts highlighted.



```
DEF PROG
OVERTYPE TO MODIFY
 CEDA  DEFine PROGram(         )
  PROGram      ==> PRICEINT
  Group        ==> INSDEMO
  DEscription  ==>
  Language     ==> COBOL               COBol | Assembler | Le370
  RELoad       ==> No                  No | Yes
  RESident     ==> No                  No | Yes
  USAge        ==> Normal              Normal | Transient
  USElpacopy   ==> No                  No | Yes
  Status       ==> Enabled             Enabled | Disabled
  RSl            : 00                  0-24 | Public
  CEdf         ==> Yes                 Yes | No
  DAtalocation ==> Below               Below | Any
  EXECKey      ==> User                User | Cics
  COncurrency  ==> Quasirent           Quasirent | Threadsafe
  Api          ==> Cicsapi             Cicsapi | Openapi
 REMOTE ATTRIBUTES
+ DYnamic      ==> No                  No | Yes
```

Figure 9-27 The PRICEINT program definition

We then install this program definition by using the CEDA INSTALL command. Finally, we perform a NEWCOPY command on the INSCTRL program and the INSPRICE program to ensure we have the latest versions of the programs loaded.

Example 9-21 shows the CEMT commands we use.

Example 9-21 The INSCTRL and INSPRICE NEWCOPY commands

```
CEMT SET PROGRAM(INSCTRL) NEW
CEMT SET PROGRAM(INSPRICE) NEW
```

We now start the application by starting the IPRI transaction. The functionality of the application is the same as before; however, when we want to change our pricing rules we can now modify them in Rule Studio or Rule Team Server rather than directly modifying the COBOL.

9.4.4 Summary

In this section, we demonstrated how COBOL code that is generated in Rules for COBOL can be integrated into an existing application by using the CICS Wrapper program functionality. By extracting our business rules from our COBOL application we gain the following benefits:

- The basic pricing rules are now stored in a decision table. This table is easier to understand and update than the COBOL code.
- The rules can be published to the Rule Team Server and shared with the business side of the organization. This means that people in the business side of the organization can directly change the rules rather than raising a code change request with IT.
- A Rules for COBOL solution slots in to the existing application architecture without requiring any major restructuring.
- As the program generated by Rules for COBOL closely integrates with our existing application, the performance of our application is faster than a web services-based solution.

The disadvantage of a solution that is based on Rules for COBOL over a JRules web service-based solution is that we lose some agility. With Rules for COBOL, the COBOL code must be generated, compiled, and deployed each time a change is made to the rules. With JRules hot-deploying the rules without having to perform any of those tasks is possible.

When deciding which solution to use, factors such as performance, flexibility, and the cost of refactoring the application all must be considered. In a situation where performance is crucial, trading the flexibility of JRules for the performance gains of Rules for COBOL might be worthwhile. In a situation where flexibility is important, the cost of making a call through web services may be worth paying.

10

Integrating WebSphere Process Server with JRules

In this chapter, we provide a scenario that uses WebSphere Process Server with JRules, and WebSphere Integration Developer.

This chapter contains the following topics:

- Introduction to the scenario
- Overview of the solution
- Verifying the environment
- Running the solution environment
- Developing the scenario

10.1 Introduction to the scenario

A team of architects has decided to automate the quotation generation activities, shown as a BPM stack in Figure 10-1.

The team has also decided to use JRules to externalize the business rules. The team plans to use WebSphere Business Modeler for doing the process modeling. WebSphere Integration Developer, WebSphere ILOG Rule Studio and Rule Team Server will be used for development and assembly of the artifacts. WebSphere Process Server and WebSphere ILOG Rule Execution Server are to be used in the runtime environment.

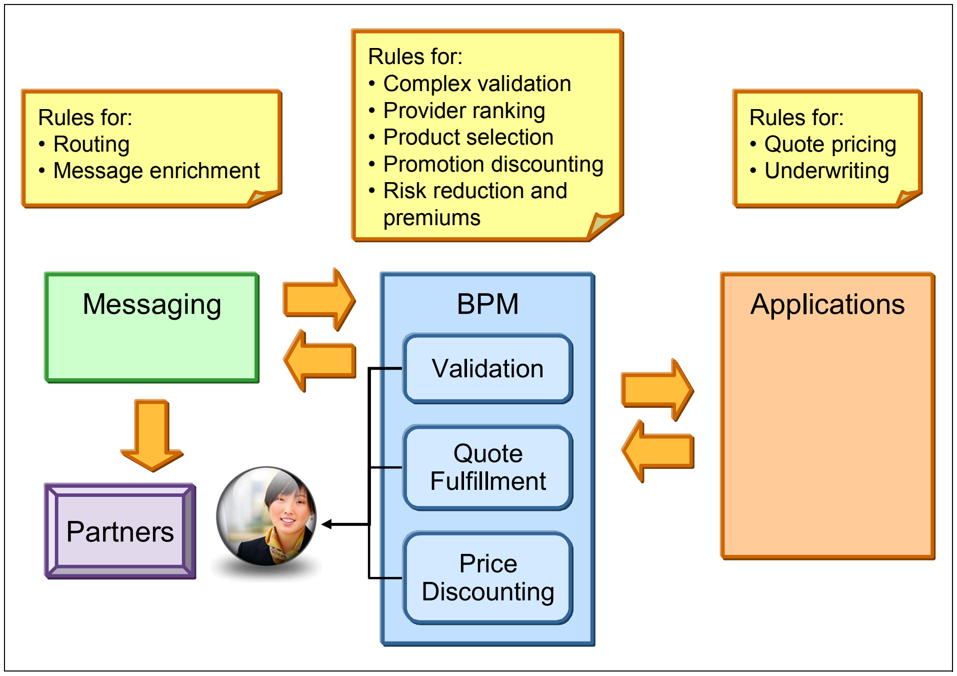


Figure 10-1 Scenario for underwriting process

10.2 Overview of the solution

A team of analysts has created a simple business model by using WebSphere Business Modeler. The model will handle the quote request processing. See Figure 10-2.

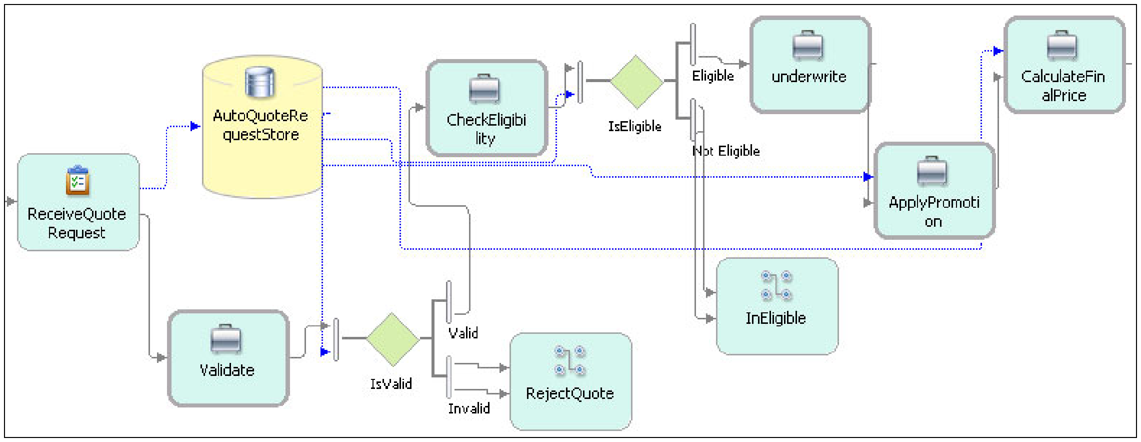


Figure 10-2 Modeled process flow in WebSphere Business Modeler

As you can see, the Validate and CheckEligibility tasks are good candidates for rule-based implementation because the process is using the output of those steps to make a decision. The Underwrite task relies on underwriting rules (business policies) that can be shared throughout the enterprise so it can also be implemented as a business rule. The dynamic nature of promotions (frequent changes that are based on market conditions) dictate that promotions also be implemented as a Business Rule. In a real world scenario, such an exercise should be conducted to discover rules from existing business processes.

After the discovery of business rules in the process model is done, they must be implemented (validity checks, eligibility checks, product underwriting, and promotions) as business rules in ILOG JRules. The remaining tasks are implemented as simple BPEL microflows in the solution. The BPEL flow orchestrates the solution integration decision points in the process flow.

Note: Multiple integration patterns with ILOG JRules are described later in this chapter.

Design-time considerations are as follows:

- A common predefined schema for the business objects in WebSphere Integration Developer and Business Modeler is used. We also use the same schema to generate the XOM in WebSphere ILOG JRules.
- We follow the general best practice of creating the service and schema definitions using WebSphere Integration Developer because it has extra capabilities and is a better tool to create schemas. We import those as business object or services inside WebSphere Business Modeler before creating the process model.
- Because we have various integration options (POJO, remote EJB, web service), we use the Selector feature of WebSphere Integration Developer which enables us to select the integration pattern by using the WebSphere Process Server administrative console. This configuration provides us with agility in the runtime environment.

After the model from the WebSphere Business Modeler is imported in the WebSphere Integration Developer, the BPEL model is similar to Figure 10-3 on page 243.

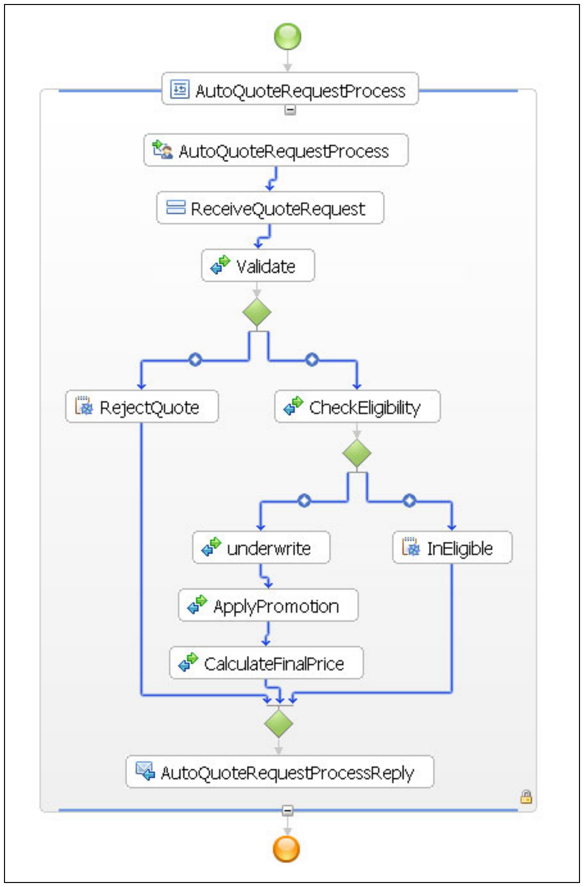


Figure 10-3 Assembled BPEL process flow

10.3 Verifying the environment

Ensure that the following software is installed and configured before you develop or run the sample:

- WebSphere Business Modeler 7.0.0
- WebSphere Integration Developer 7.0.0
- WebSphere ILOG JRules 7.0.3
- WebSphere Process Server 7.0.0 on Windows XP SP3
- WebSphere ILOG RES 7.0.3 on the same WebSphere Process Server installation
- WebSphere bundle for JRules 7.0.3
- SupportPac LA71 (1.0)

Getting help: If you need any help during installation, use the product installation guides for the individual products. For our RES database, we used Derby, which is the default. You may use any of the databases that are supported by RES.

For SupportPac LA71, see the installation instructions, accessible at the following address:

ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/la71_readme.html

In addition, ensure that WebSphere Process Server and RES are running.

10.4 Running the solution environment

In this section, we provide instructions for setting up and testing the solution environment.

10.4.1 Setting up the environment

Note: Use the modules supplied in the `\finish` directory of the source code. Extract the source projects in your file system to start this activity.

If you want to develop your modules before testing it, skip this section for now and come back after you have finished developing all modules.

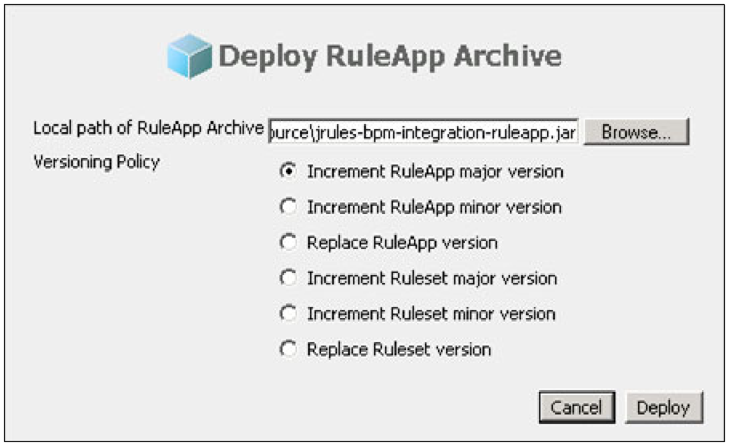
Perform the following steps:

1. Be sure that WebSphere Process Server and RES are running
2. Deploy the insdemoRuleApp to the RES console by launching the RES console and clicking **Deploy RuleApp Archive**.

> **Tip:** WebSphere Process Server and RES must be are running. To open RES console, open a browser and enter the RES URL. For our test we used the URL in the following format:
>
> `http://localhost:9086/res`

3. Select the `jrules-bpm-integration-ruleapp.jar` file, which is available in the following file system directory, and then click **Deploy** (Figure 10-4):

   `\finish\rule directory`



Deploy RuleApp Archive

Local path of RuleApp Archive: ource\jrules-bpm-integration-ruleapp.jar

Versioning Policy:
- [✓ Increment RuleApp major version]
- Increment RuleApp minor version
- Replace RuleApp version
- Increment Ruleset major version
- Increment Ruleset minor version
- Replace Ruleset version

Figure 10-4 Deploying RuleApp with chosen version control

4. Ensure that the following modules in the directory are deployed and started in WebSphere Process Server:
   – jrules-bpm-integration-model
   – jrules-bpm-integration-model_impl
   – jrules-bpm-integration-bpel_impl
   – jrules-bpm-integration-pojo-rule_impl
   – jrules-bpm-integration-localejb-rule_impl
   – jrules-bpm-integration-remoteejb-rule_impl
   – jrules-bpm-integration-validation-wsdl-rule_impl
   – jrules-bpm-integration-eligibility-wsdl-rule_impl
   – jrules-bpm-integration-pricing-wsdl-rule_impl
   – jrules-bpm-integration-promotion-wsdl-rule_impl

To ensure the modules are there, perform the following steps:

a. Open your workspace in WebSphere Integration Developer

b. Import all of the modules listed in this step.

c. Build all of the projects, if not already built.

d. Go to **Server view** and start the WebSphere Process Server instance, if it is not already started.

e. Right-click the server instance.

f. Select **Add or Remove Projects** to add the modules you want to add.

5. Select the integration type for each decision service. We have options selecting POJO, remote EJB, local EJB, or WSDL. You may select a separate integration for separate decision points, as follows:

   a. Launch the WebSphere Administrative console and select **Servers** → **Server Types** → **WebSphere application servers** → **server1**.

   b. Click **Selectors-Selectors** in the Business Integration section of the panel, as shown in Figure 10-5.

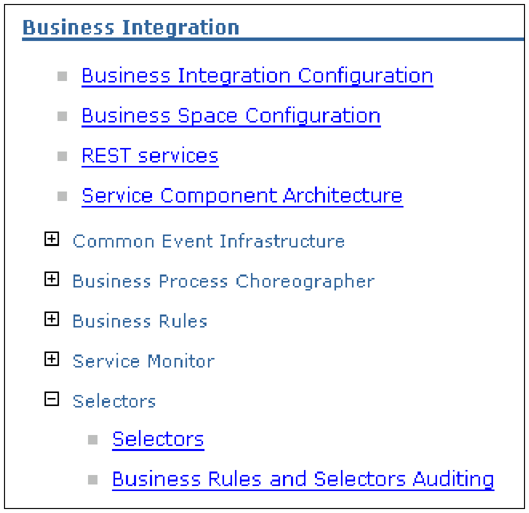

Business Integration
Business Integration Configuration
Business Space Configuration
REST services
Service Component Architecture
Common Event Infrastructure
Business Process Choreographer
Business Rules
Service Monitor
Selectors
Selectors
Business Rules and Selectors Auditing

Figure 10-5 WebSphere Administration console options

   c. The following selectors are available:

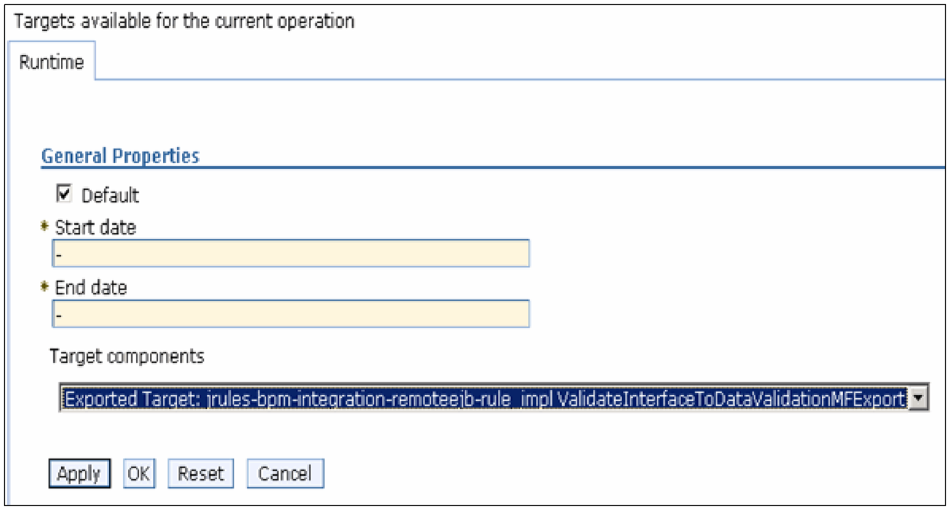
      - ValidateImplSelector
      - EligibilityImplSelector
      - UnderwriteImplSelector
      - PromotionImplSelector

We will be changing the ValidateImplSelector to use a separate implementation.

Tip: You can change each of the selectors to use any of the available implementations.

d. Select **ValidateImplSelector** and then click **Validate** and default link on the subsequent windows.

e. For the target components, select an integration type and click **OK**, as shown in Figure 10-6.


Targets available for the current operation

Runtime

General Properties

☑ Default

* Start date: -

* End date: -

Target components: Exported Target: jrules-bpm-integration-remoteejb-rule_impl ValidateInterfaceToDataValidationMFExport

Figure 10-6   Change selectors

f. Be sure that you select the **Default** and click **Commit** to enable the new implementation.

g. Set a selector for all the decision points and you are ready to test.

10.4.2 Testing the solution

The QuoteRequestProcess can be tested by using either the WebSphere Integration Developer Integration Test Client or any web service client. We have tested the solution using the test client in WebSphere Integration Developer.

Tips:

- Four test data files are included in AutoInsuranceQuotingXSDBOM project that you can use for testing. These are available in the `data\xml` folder.
- The Webservice WSDL can be found in the following location (use the appropriate port number):

  ```
  http://localhost:<portnumber>/jrules-bpm-integration-modelWeb/s
  ca/AutoQuoteRequestProcessExport1/WEB-INF/wsdl/processes/autoqu
  oterequestprocess/jrules-bpm-integration-model_AutoQuoteRequest
  ProcessExport1.wsdl
  ```

To test by using the WebSphere Integration Developer client, perform the following steps:

1. In WebSphere Integration Developer, open jrules-bpm-integration-model project and open the assembly diagram.
2. Right-click the process named **AutoQuoteRequestProcess** and select **Test Component** from the menu option. Populate the test data and click **Invoke**.

10.5 Developing the scenario

In this section, we look at development of the scenario.

10.5.1 Setting up the solution environment

Before starting to build the solution, ensure that the quote request process model application is imported into WebSphere Integration Developer and the RuleApp is imported into Rule Studio.

Importing projects in WebSphere Integration Developer

Launch WebSphere Integration Developer in a new workspace and import the following projects (**File** → **Import** → **General** → **Existing Projects** into the workspace):

- jrules-bpm-integration-model

  This module contains the process flow for the quote request process as defined in the business modeler. We do not modify this module as part of our solution.

- jrules-bpm-integration-model_lib

  This library contains the business object definitions and also the service interface definitions.

- jrules-bpm-integration-model_impl

  This module contains the implementation components for all the microflow implementations as selectors that helps pick a different integration option with JRules components during run time.

- jrules-bpm-integration-bpel_impl

  This module has BPEL implementation of the final price calculation activity. It also has BPEL implementation of promotion rules because they are simple and can be implemented in either WebSphere Integration Developer or Rule Studio.

Importing projects in Rule Studio

Launch JRules Rule Studio in a new workspace and import the following projects (by using **File** → **Import** → **General** → **Existing Projects** into the workspace):

- AutoInsuranceQuotingXSDBOM

  This project contains the XOM and BOM definitions for our solution.

- DataValidation

  This project contains the rules for validity check.

- Eligibility

  This project contains the rules for eligibility checks.

- insdemoRESConfig

  This project contains the RES connection settings.

- insdemoRuleApp

  This project contains the RuleApp and all the rules that must be deployed for the scenarios.

- Pricing

  This project contains pricing rules.
- Promotion

  This project contains promotion rules.

Optional Eclipse shell sharing: We install Rule Studio in the same Eclipse shell as WebSphere Integration Developer. Go to the following address for further details.

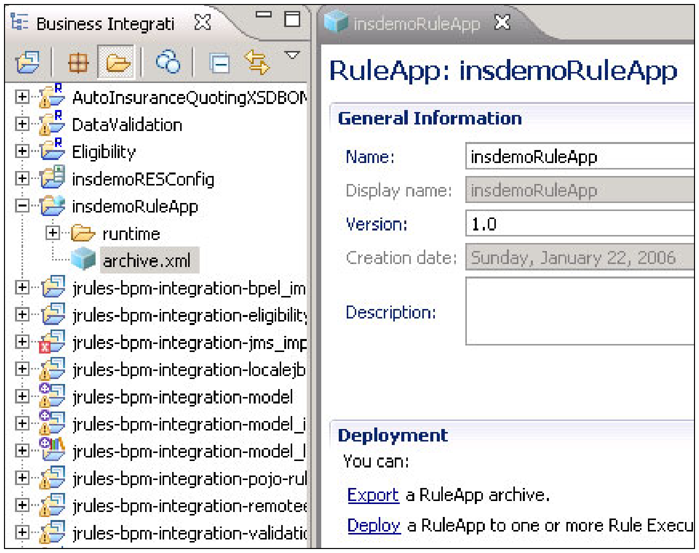
`http://www.ibm.com/developerworks/data/library/techarticle/dm-0811khatri/index.html?ca=drs-`

With a shared shell, we can import all the projects, which we listed, into the same workspace.

Extracting RuleApp archive

Ensure that the RuleApp archive for the insdemoRuleApp project is generated and stored in a folder on your hard drive:

1. Switch to Rule perspective and expand **insdemoRuleApp** in Rule Explorer.
2. Open the `archive.xml` file and click the **Export** link in the Deployment panel, as shown in Figure 10-7.


Business Integrati
AutoInsuranceQuotingXSDBOM
DataValidation
Eligibility
insdemoRESConfig
insdemoRuleApp
runtime
archive.xml
jrules-bpm-integration-bpel_im
jrules-bpm-integration-eligibility
jrules-bpm-integration-jms_imp
jrules-bpm-integration-localejb
jrules-bpm-integration-model
jrules-bpm-integration-model_i
jrules-bpm-integration-model_l
jrules-bpm-integration-pojo-rul
jrules-bpm-integration-remotee
jrules-bpm-integration-validatio
insdemoRuleApp

RuleApp: insdemoRuleApp

General Information

Name: insdemoRuleApp

Display name: insdemoRuleApp

Version: 1.0

Creation date: Sunday, January 22, 2006

Description:

Deployment

You can:

Export a RuleApp archive.

Deploy a RuleApp to one or more Rule Execu

Figure 10-7 Importing RuleApp archive in ILOG Rule Studio

3. Ensure all the rule sets are selected for the JAR file location by specifying the following file, and then click **Finish**:

   ```
   foldername\jrules-bpm-integration-ruleapp.jar
   ```

   Note: You can specify any name for the RuleApp. For our development, we used the following name:

   ```
   -ruleapp.jar
   ```

4. Verify that the console displays the following message:

   ```
   The "insdemoRuleApp" RuleApp project is exported to:
   foldername\jrules-bpm-integration-ruleapp.jar
   ```

   Tip: If the console is not visible, enable it by clicking **Window** → **Show View** → **Other** → **General** → **Console**.

10.5.2 Developing the integration components

In the process flow, our scenario has four decision points that must be integrated:

- Validation
- Eligibility
- Underwriting
- Promotion

In this section, we show how to create integration components for the validation decision point. You need to follow similar steps to develop an integration component for the remaining decision points.

Up to now, we have explored possible integration patterns. We have created SCA bindings for POJO, Remote EJB, and Local EJB integration. We explored web service binding to use HTDS feature available in the RES. We have also found several issues during local EJB integration, which is described in “Local EJB integration using the LA71 plug-in” on page 263.

Note: If you want to see the scenario running without developing the component, use the projects in the `finish` directory.

POJO integration using the LA71 plug-in

In this section, we implement the integration component for executing the Data Validation rule set from the *Validate* process, shown in the BPEL diagram in Figure 10-3 on page 243.

Figure 10-8 shows the implementation steps.

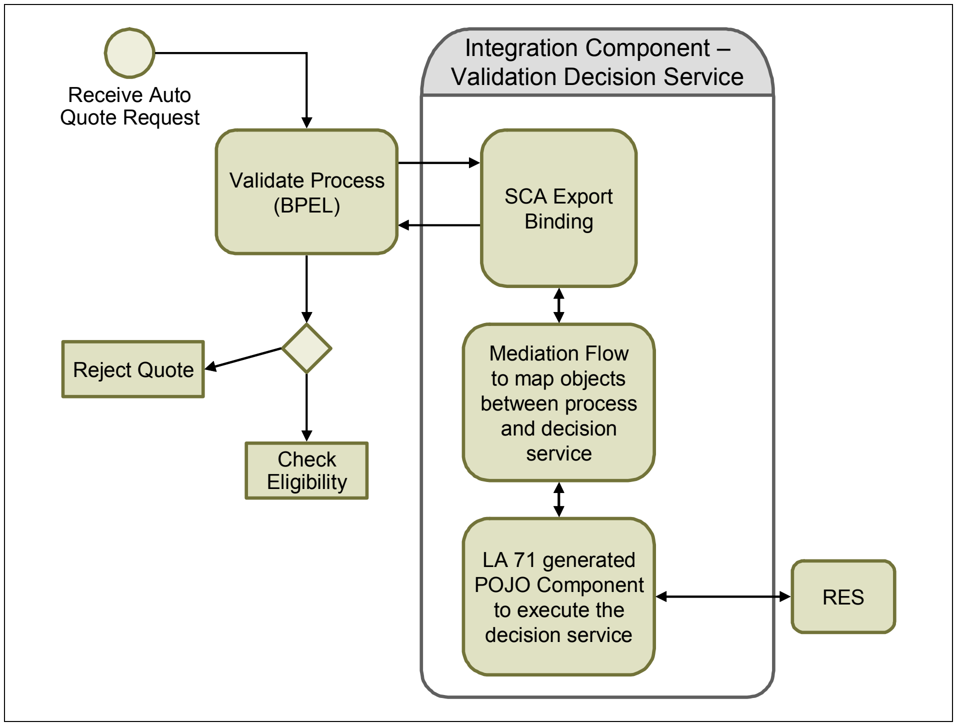

Receive Auto Quote Request
Validate Process (BPEL)
Reject Quote
Check Eligibility
Integration Component – Validation Decision Service
SCA Export Binding
Mediation Flow to map objects between process and decision service
LA 71 generated POJO Component to execute the decision service
RES

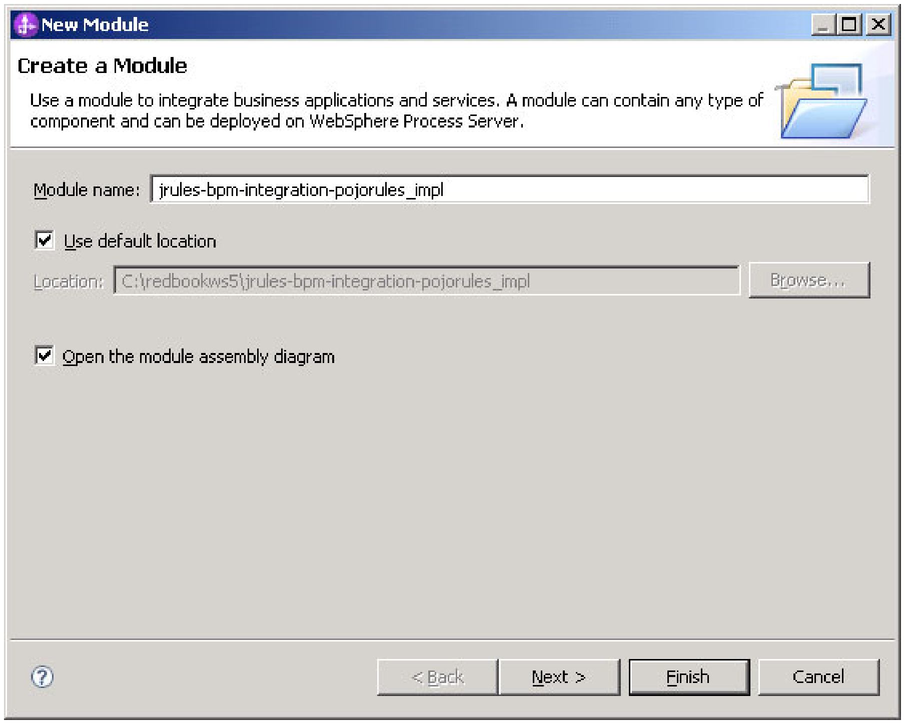
Figure 10-8 Implementation steps to integrate decision into process using POJO

Developing the integration component

Use the following procedure to develop the integration component.

1. Open WebSphere Integration Developer Workspace and select **Business Integration Perspective**.
2. Create a new module (Figure 10-9 on page 253) by clicking **File** → **New** → **Module**. Provide a name for the module and click **Next**.

   We used `jrules-bpm-integration-pojo-rules_impl` as the module name.


New Module

Create a Module

Use a module to integrate business applications and services. A module can contain any type of component and can be deployed on WebSphere Process Server.

Module name: jrules-bpm-integration-pojorules_impl

☑ Use default location

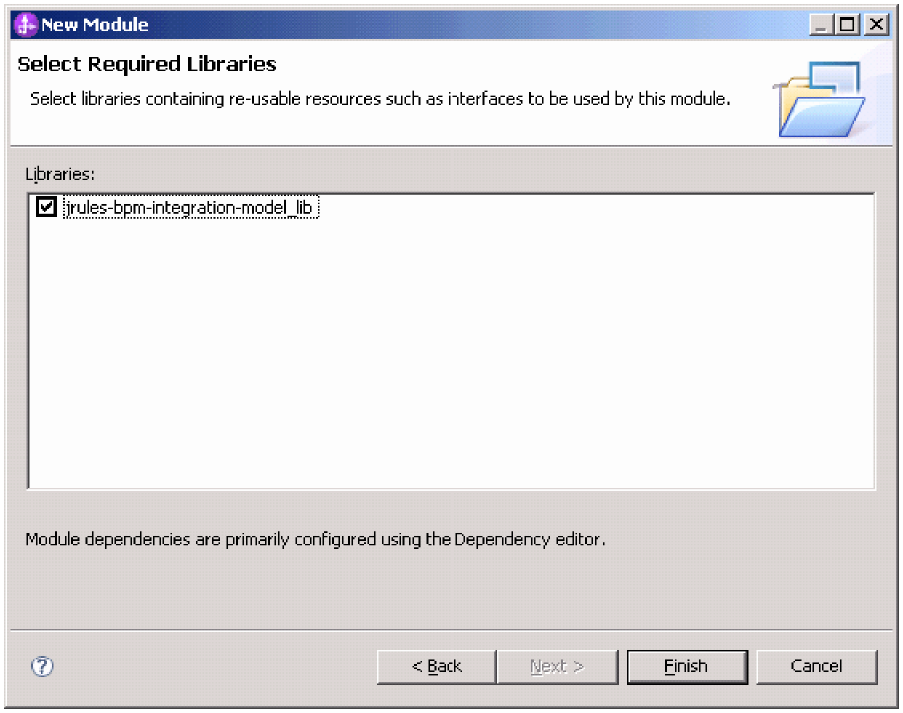
Location: C:\redbookws5\jrules-bpm-integration-pojorules_impl

☑ Open the module assembly diagram

Figure 10-9 Create a new module

Note: We used the default location because we encountered issues when we selected a different location for the module when using SupportPac LA71. Based on our experience, a good practice is to use the default location.

3. Select the check box next to the `jrules-bpm-integration-model_lib` library, as shown in Figure 10-10. Click **Finish**.


New Module

Select Required Libraries

Select libraries containing re-usable resources such as interfaces to be used by this module.

Libraries:

☑ jrules-bpm-integration-model_lib

Module dependencies are primarily configured using the Dependency editor.

Figure 10-10 Select libraries required

4. Expand the newly created module in the Business Integration Explorer. Select **Integration Logic,** right-click and then select **New** → **Other** from the menu. A dialog box opens, as shown in Figure 10-11.

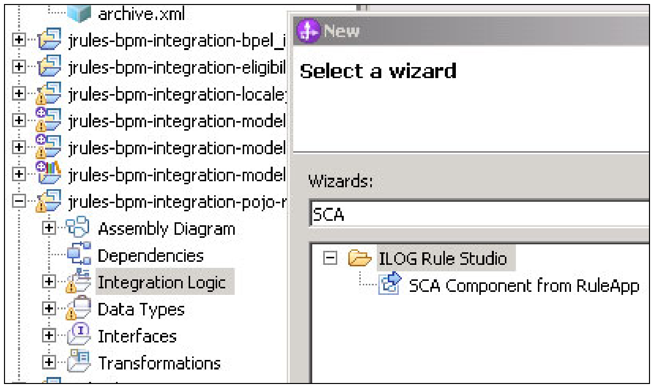
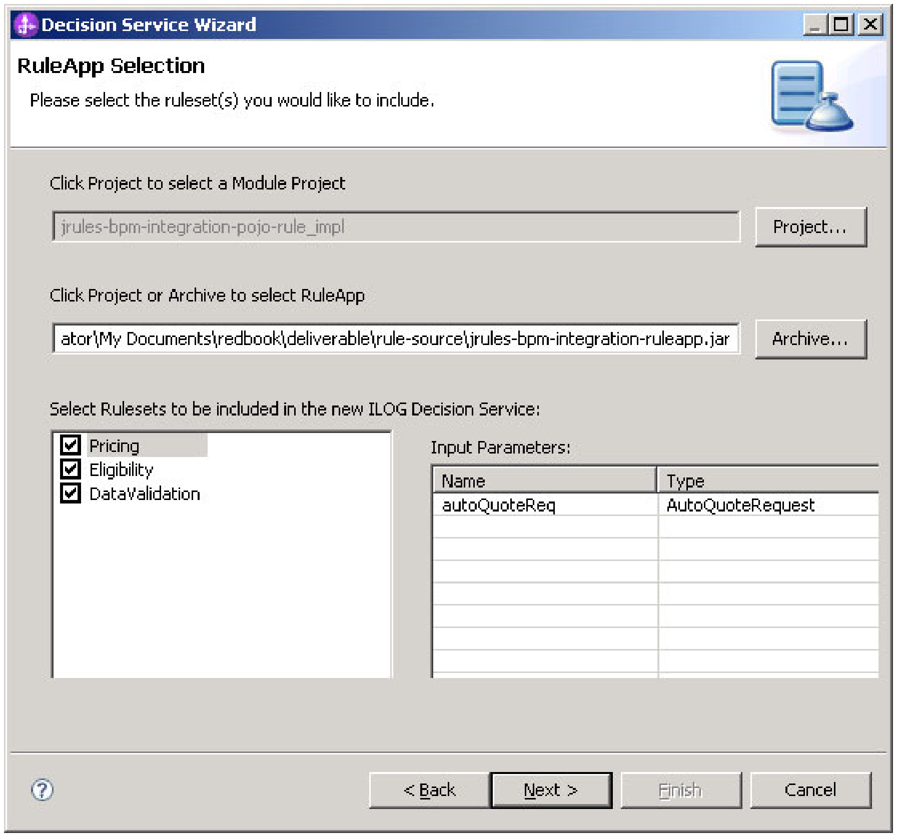

archive.xml
jrules-bpm-integration-bpel_i
jrules-bpm-integration-eligibil
jrules-bpm-integration-locale
jrules-bpm-integration-model
jrules-bpm-integration-model
jrules-bpm-integration-model
jrules-bpm-integration-pojo-r
Assembly Diagram
Dependencies
Integration Logic
Data Types
Interfaces
Transformations
New
Select a wizard
Wizards:
SCA
ILOG Rule Studio
SCA Component from RuleApp

Figure 10-11 Invoke LA71 wizard to simplify integration steps

5. Expand **ILOG Rule Studio** and select **SCA Component** from RuleApp. (You may also search for it by typing `SCA component` in the text box below the Wizards label.) Click **Next.**

   Note: Right-clicking the module and creating a new SCA component is another option of doing this step.

6. Click **Project**. A list of open projects in the workspace is displayed. Select `jrules-bpm-integration-pojo-rules_impl` and click **OK**.
7. Click **Archive** and select the `jrules-bpm-integration-ruleapp.jar` file that was exported (explained in 10.5.1, “Setting up the solution environment” on page 248).
8. The Wizard now lists all rule sets that are available in the RuleApp archive and corresponding input output parameters. Ensure that all the rule sets are selected and click **Next**, as shown in Figure 10-12.



Click Project to select a Module Project

jrules-bpm-integration-pojo-rule_impl

Click Project or Archive to select RuleApp

ator\My Documents\redbook\deliverable\rule-source\jrules-bpm-integration-ruleapp.jar

Select Rulesets to be included in the new ILOG Decision Service:

- [x] Pricing
- [x] Eligibility
- [x] DataValidation

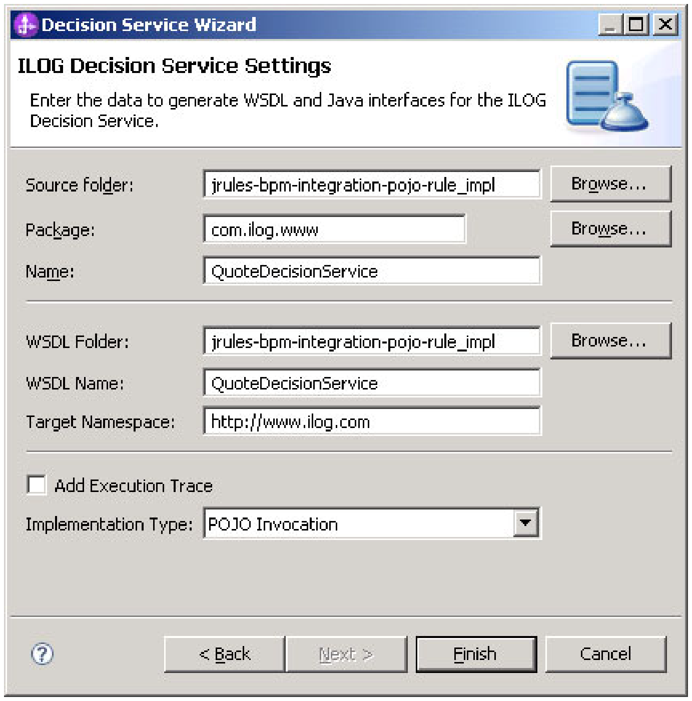
Input Parameters:

Name	Type
autoQuoteReq	AutoQuoteRequest

Figure 10-12 Select callable rule sets that can be invoked from process

9. Click **Add XSDs**, expand `jrules-bpm-integration-model_lib` and select the `insdemoObjectModel.xsd` file. Click **OK** to close the dialog and then click **Next**.

10. Input the package name you want the integration code to generate (we used `com.ilog.www`) and name of the component (we used `QuoteDecisionService`). The wizard automatically fills in the WSDL Name and Target Namespace based on the package name, as shown in Figure 10-13.


Decision Service Wizard
ILOG Decision Service Settings
Enter the data to generate WSDL and Java interfaces for the ILOG Decision Service.

Source folder: jrules-bpm-integration-pojo-rule_impl
Package: com.ilog.www
Name: QuoteDecisionService
WSDL Folder: jrules-bpm-integration-pojo-rule_impl
WSDL Name: QuoteDecisionService
Target Namespace: http://www.ilog.com
☐ Add Execution Trace
Implementation Type: POJO Invocation

Browse...
Browse...
Browse...
< Back
Next >
Finish
Cancel

Figure 10-13 Configure binding options and set POJO invocation method

11. Ensure that the Implementation Type is set to POJO Invocation and click **Finish**.

    The wizard generates the required artifacts and places the Java component in the assembly diagram of the module.

> **Note:** You may encounter a build error at this time. To eliminate the error, right-click the actual error, select **QuickFix**, and select **Change Java Compiler Level to 1.6**.

12. You must set the appropriate transaction scope for the Java Component that was added in the previous step. Because POJO implementation does not

support transaction, it must mirror the transactional behavior of the overall module. For our scenario, we selected the **Global** setting.

Select the component in the assembly diagram, go to the Properties view and select **Implementation**. Then go to the Qualifiers tab and add **Transaction**. This step will add set global transaction for the component.

Testing the component

To ensure that the JRules component is working correctly, test the component by right-clicking it in the assembly diagram. See Section 10.4.2, "Testing the solution" on page 248 for test data and how to test.

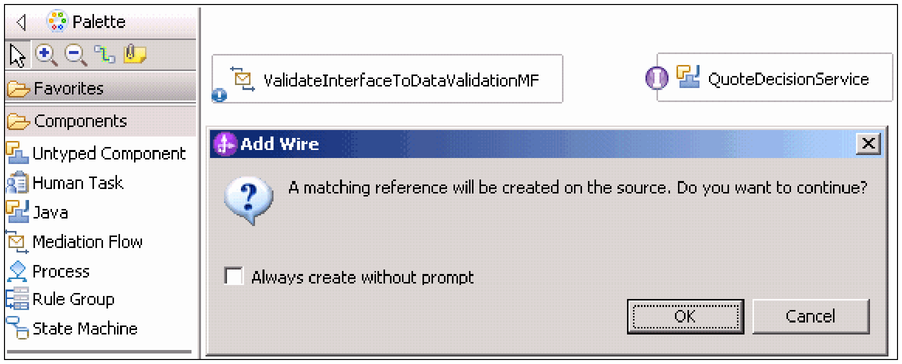
Optional: Creating a mediation flow

Because the object model in the BPEL does not exactly match with the one that the LA71 plug-in interfaces created before, we need a mediation flow to map the input and output objects.

Optional: Mediation flow is not required if the input and output object of the process calling JRules are the same. Therefore, the following steps are optional.

Perform the following steps

1. In the assembly diagram, drag the Mediation Flow from the Components Section of the palette onto the canvas. Rename the component to `ValidateInterfaceToDataValidationMF`.
2. Create a link from the `ValidateInterfaceToDataValidationMF` component to the QuoteDecisionService. When the Add Wire dialog box opens, click **OK** (Figure 10-14).

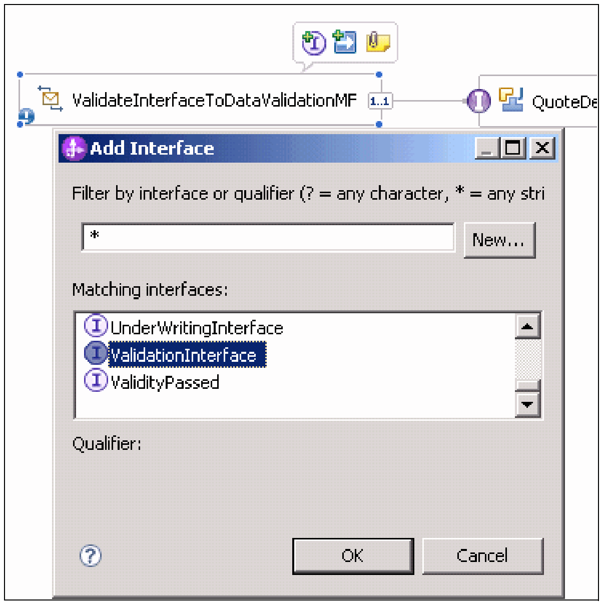


A matching reference will be created on the source. Do you want to continue?

Always create without prompt

Figure 10-14 Map the process interface to the decision interface

3. Click **Mediation Flow**, click the **Add Interface** option, and then select **ValidationInterface**. See Figure 10-15.



Filter by interface or qualifier (? = any character, * = any stri

*

New...

Matching interfaces:

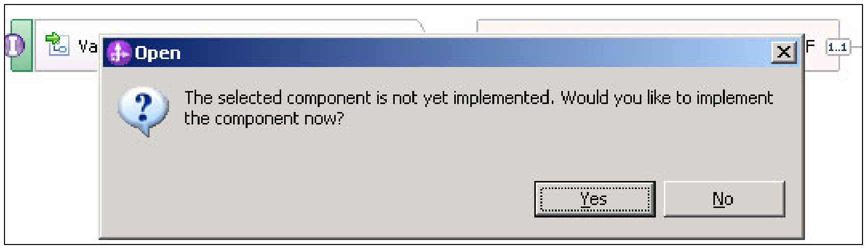
UnderWritingInterface
ValidationInterface
ValidityPassed

Qualifier:

OK
Cancel

Figure 10-15 Select the decision interface

4. Right-click the Mediation Flow and select **Generate Export - SCA Binding** to create an SCA binding for the component.
5. Double-click the mediation flow and click **Yes** when prompted to create it now (Figure 10-16).



The selected component is not yet implemented. Would you like to implement the component now?

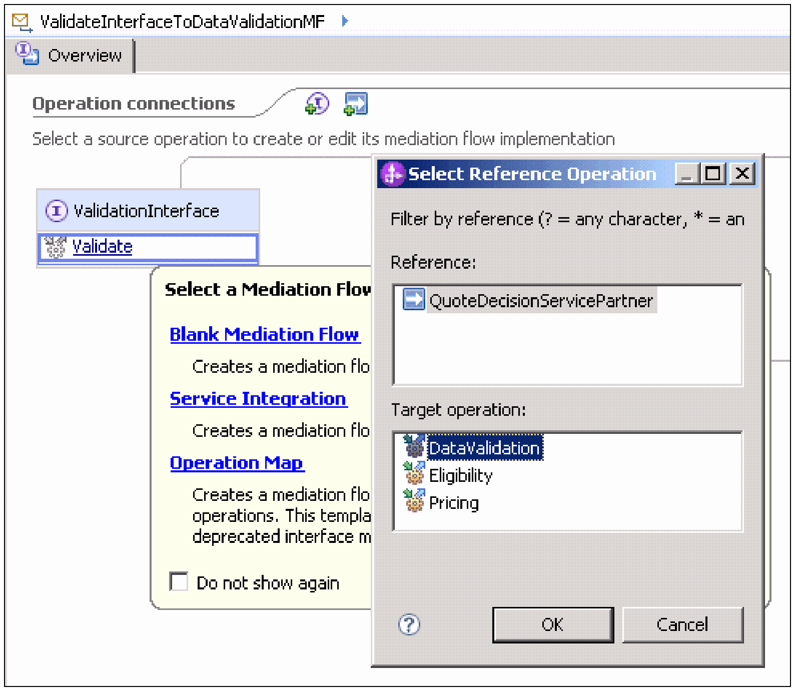
Yes
No

Figure 10-16 Creating an SCA Binding

6. Click **OK** to confirm the default root folder for the implementation.
7. Click the **Validate** operation of ValidationInterface and select **Operation Map**.

8. In the Select Reference Operation dialog, make the following selections and then click **OK** (Figure 10-17):

   – **QuoteDecisionServicePartner** under Reference
   – **DataValidation** as the Target operation


ValidateInterfaceToDataValidationMF

Overview

Operation connections

Select a source operation to create or edit its mediation flow implementation

ValidationInterface

Validate

Select a Mediation Flow

Blank Mediation Flow

Creates a mediation flo

Service Integration

Creates a mediation flo

Operation Map

Creates a mediation flo
operations. This templa
deprecated interface m

Do not show again

Select Reference Operation

Filter by reference (? = any character, * = an

Reference:

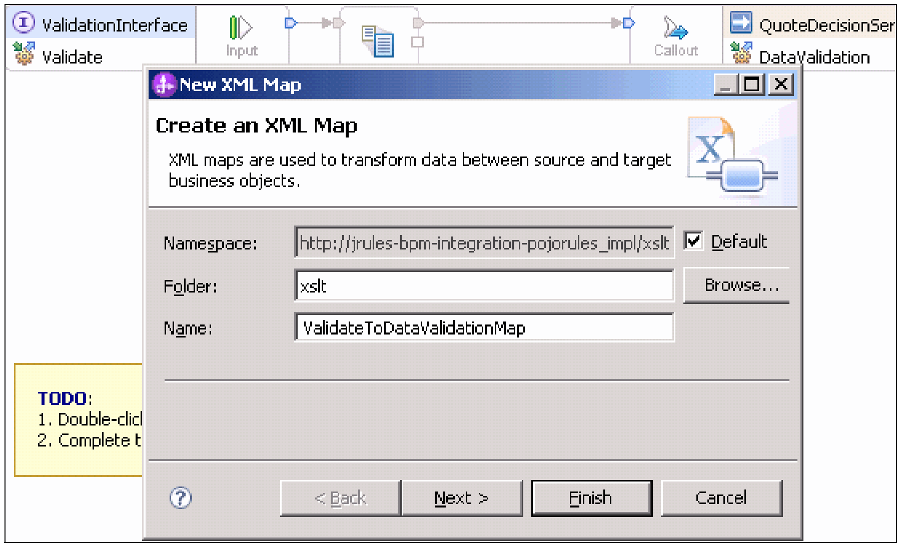
QuoteDecisionServicePartner

Target operation:

DataValidation
Eligibility
Pricing

Figure 10-17 Select the interface for a new SCA module

9. Double-click the Input Map that was generated in a previous step. Select **ValidateToDataValidationMap** as the name, as shown in Figure 10-18 on page 260. Click **Finish**.



Create an XML Map

XML maps are used to transform data between source and target business objects.

Namespace: http://jrules-bpm-integration-pojorules_impl/xslt [✓ Default]

Folder: xslt

Name: ValidateToDataValidationMap

TODO:
1. Double-clicl
2. Complete t

Figure 10-18 Create a new XML map

10. Expand the body section of both input and output objects and drag the AutoQuoteRequest to autoQuoteReq to create a Move mapping, as shown in Figure 10-19.

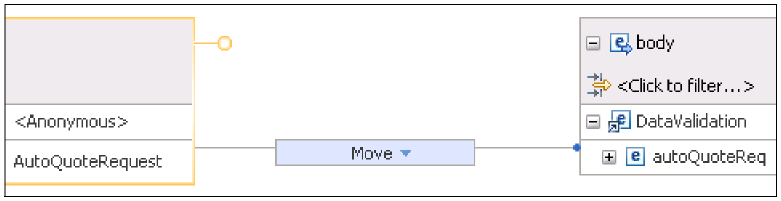
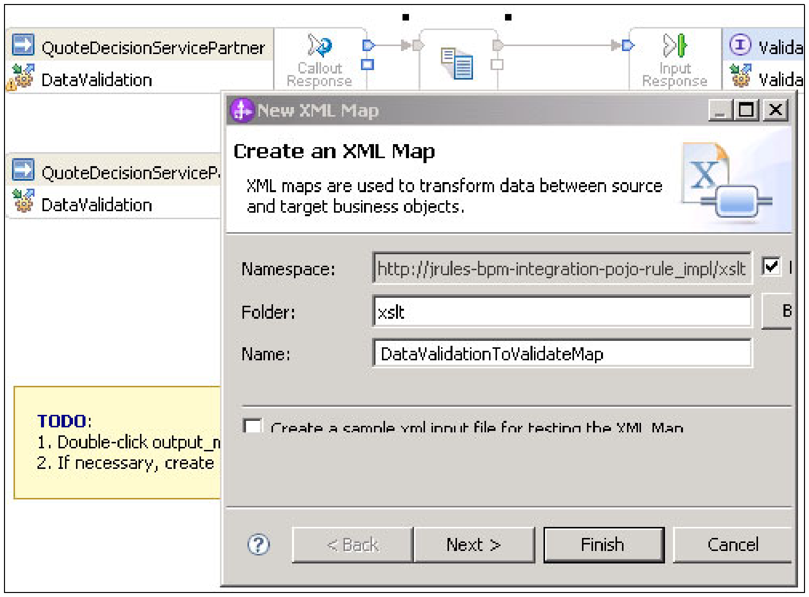


Figure 10-19 Mapping outgoing data objects

11. Save and close the map.

12. Click the Response tab of the mediation flow and double-click **OutputMap** to implement it. Select **DataValidationToValidateMap** as the name, as shown in Figure 10-20 on page 261. Click **Finish**.



New XML Map

Create an XML Map

XML maps are used to transform data between source and target business objects.

Namespace: http://jrules-bpm-integration-pojo-rule_impl/xslt

Folder: xslt

Name: DataValidationToValidateMap

Create a sample xml input file for testing the XML Map

Figure 10-20 Create another XML map

13.Expand the DataValidation and DataValidationExecutionResult objects in the input object and expand the validateResponse in output object. Drag the ValidationResp to ValidationResponse to create a Move map, as shown in Figure 10-21.

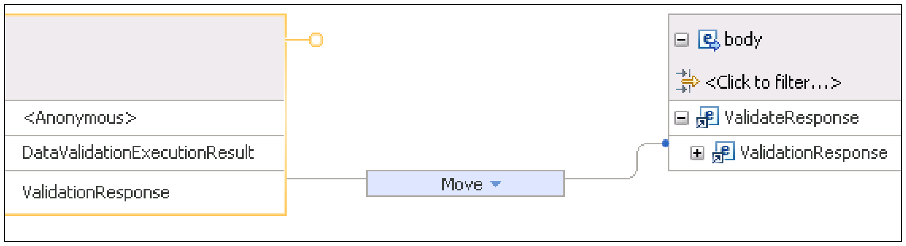
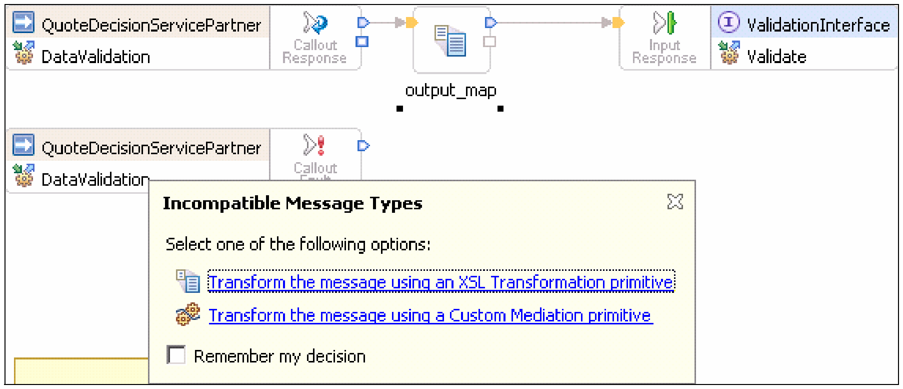


Figure 10-21 Map incoming result

14.Save and close the output map.

15.Create a link between the CalloutFault and InputResponse and select **Transform message using an XSL Transformation primitive** in the dialog box, as shown in Figure 10-22 on page 262.



Incompatible Message Types

Select one of the following options:

Transform the message using an XSL Transformation primitive

Transform the message using a Custom Mediation primitive

Remember my decision

Figure 10-22 Handle fault case

16. Double-click the XSL Transformation. Use the following name and then click **Finish**:

    `DataValidationFaultToValidateMap`

17. Create a move link between the detailMessage and MainMessage. Create an Assign object with false as the value for Validated. Create a move link between detailMessage and ErrorMessages, as shown in Figure 10-23.

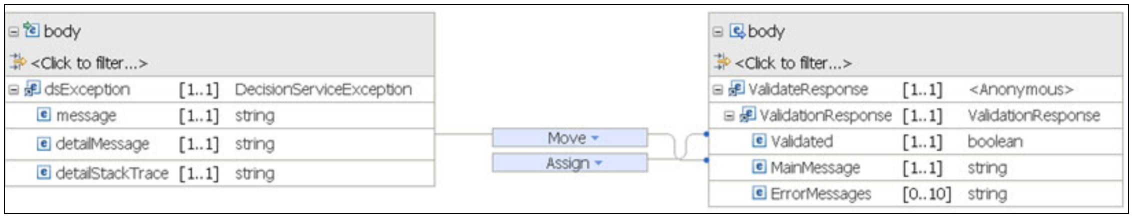

body
<Click to filter...>
dsException [1..1] DecisionServiceException
message [1..1] string
detailMessage [1..1] string
detailStackTrace [1..1] string
Move
Assign
body
<Click to filter...>
ValidateResponse [1..1] <Anonymous>
ValidationResponse [1..1] ValidationResponse
Validated [1..1] boolean
MainMessage [1..1] string
ErrorMessages [0..10] string

Figure 10-23 Create move link

18. Save and close the FaultMap.
19. Save and close the ValidateInterfaceToDataValidationMF mediation flow and also the assembly diagram.
20. Build the module and deploy it on the WebSphere Process Server. Follow standard procedures in WebSphere Integration Developer to build and deploy the modules to WebSphere Process Server.

Local EJB integration using the LA71 plug-in

In this section, we implement the integration component for executing the Data Validation rule set from the *Validate* process, shown in the BPEL diagram in Figure 10-3 on page 243, using Local EJB integration pattern. We found a problem in using this pattern, therefore, use remote EJB pattern instead.

We assume the rule set is already deployed in the RES and the BPEL process, which calls the rule set that is available in WebSphere Integration Developer. We implement the Integration Component shown in Figure 10-24.

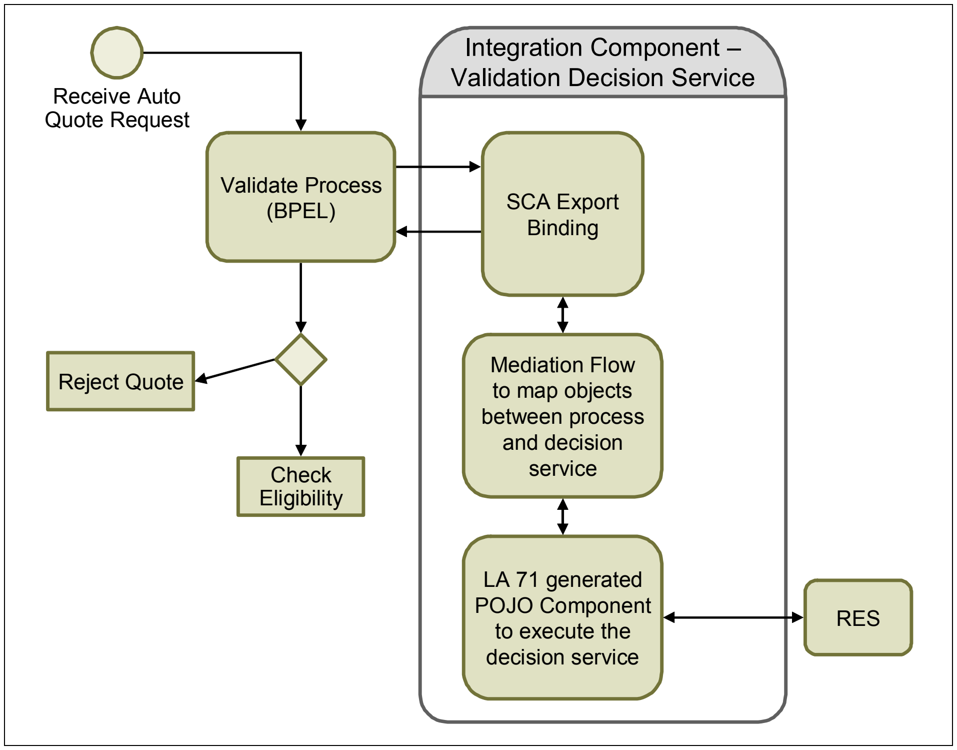

Receive Auto Quote Request
Validate Process (BPEL)
Reject Quote
Check Eligibility
Integration Component – Validation Decision Service
SCA Export Binding
Mediation Flow to map objects between process and decision service
LA 71 generated POJO Component to execute the decision service
RES

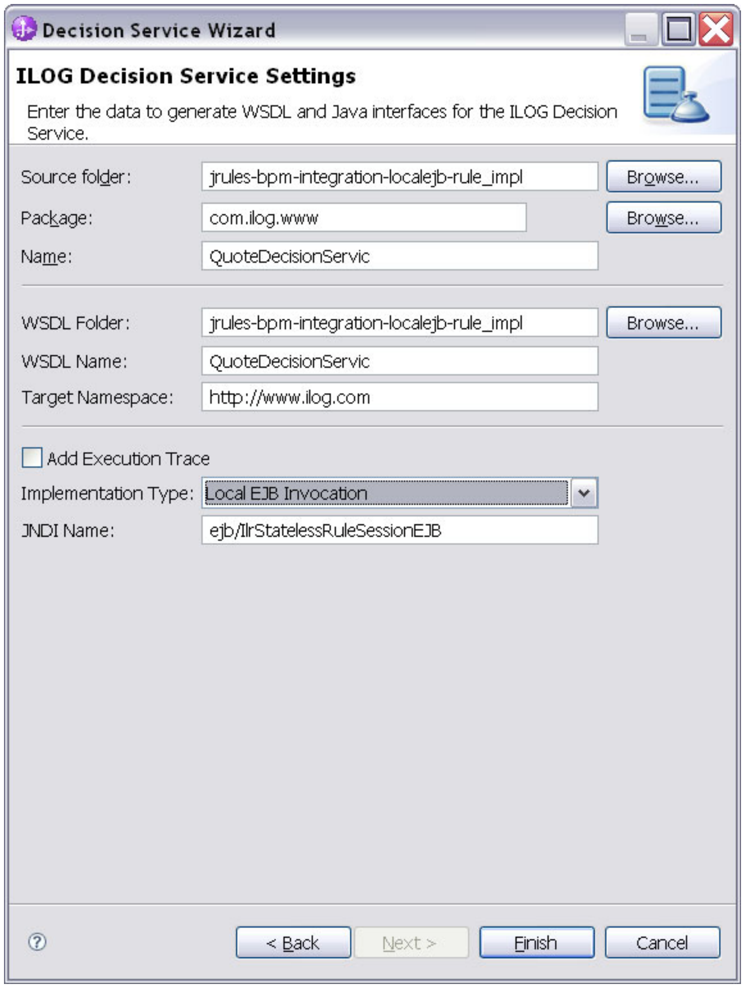
Figure 10-24 Implementation steps to integrate decision into process using local EJB

Use the following procedure to build the local EJB component:

1. Follow steps 1 on page 252 through 10 on page 256, integration using POJO calls. Choose a different module name. We used the following name:

   `jrules-bpm-integration-localejb-rule_impl`

2. Select **Local EJB Invocation** as the Implementation Type. See Figure 10-25. Use `ejb/IlrStatelessRuleSessionLocalEJB` as the JNDI Name and click **Finish**.



ILOG Decision Service Settings

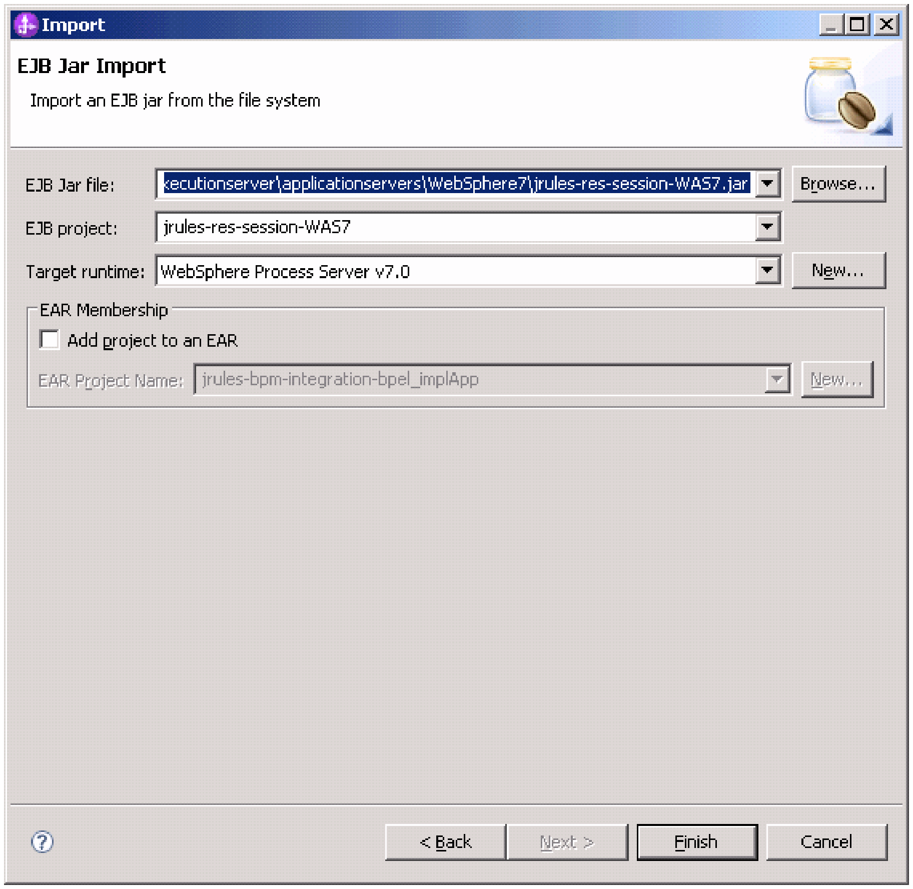
Enter the data to generate WSDL and Java interfaces for the ILOG Decision Service.

Source folder: jrules-bpm-integration-localejb-rule_impl
Package: com.ilog.www
Name: QuoteDecisionServic
WSDL Folder: jrules-bpm-integration-localejb-rule_impl
WSDL Name: QuoteDecisionServic
Target Namespace: http://www.ilog.com
Add Execution Trace: ☐
Implementation Type: Local EJB Invocation
JNDI Name: ejb/IlrStatelessRuleSessionEJB

Figure 10-25 Configure binding options and set local EJB invocation method

3. Switch to Java EE Perspective, Import the EJB JAR file by selecting **File** → **Import** → **EJB** → **EJB Jar File** and select the `jrules-res-session-WAS7.jar` file, as shown in Figure 10-26. This file is in the following standardILog JRule directory:

   ```
   C:\Program Files\IBM\WebSphereILOGJRules703\executionserver\applicationserv
   ers\WebSphere7
   ```



Import

EJB Jar Import

Import an EJB jar from the file system

EJB Jar file: xecutionserver\applicationservers\WebSphere7\jrules-res-session-WAS7.jar

EJB project: jrules-res-session-WAS7

Target runtime: WebSphere Process Server v7.0

EAR Membership

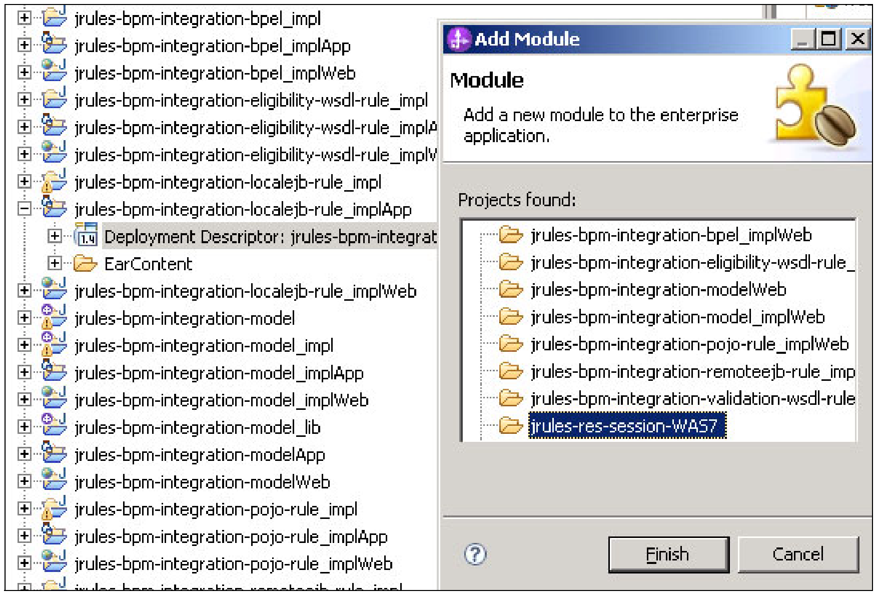
Add project to an EAR

EAR Project Name: jrules-bpm-integration-bpel_implApp

Figure 10-26 Import EJB jar file

4. Expand the `jrules-bpm-integration-localejb-rule_implApp` project and Edit the Deployment Descriptor. Navigate to the Module section, and click **Add** in the Modules section.

5. Select the `jrules-res-session-WAS7` folder from the list and click **Finish**. See Figure 10-27.


jrules-bpm-integration-bpel_impl
jrules-bpm-integration-bpel_implApp
jrules-bpm-integration-bpel_implWeb
jrules-bpm-integration-eligibility-wsdl-rule_impl
jrules-bpm-integration-eligibility-wsdl-rule_implA
jrules-bpm-integration-eligibility-wsdl-rule_implV
jrules-bpm-integration-localejb-rule_impl
jrules-bpm-integration-localejb-rule_implApp
Deployment Descriptor: jrules-bpm-integrat
EarContent
jrules-bpm-integration-localejb-rule_implWeb
jrules-bpm-integration-model
jrules-bpm-integration-model_impl
jrules-bpm-integration-model_implApp
jrules-bpm-integration-model_implWeb
jrules-bpm-integration-model_lib
jrules-bpm-integration-modelApp
jrules-bpm-integration-modelWeb
jrules-bpm-integration-pojo-rule_impl
jrules-bpm-integration-pojo-rule_implApp
jrules-bpm-integration-pojo-rule_implWeb

Add Module

Module

Add a new module to the enterprise application.

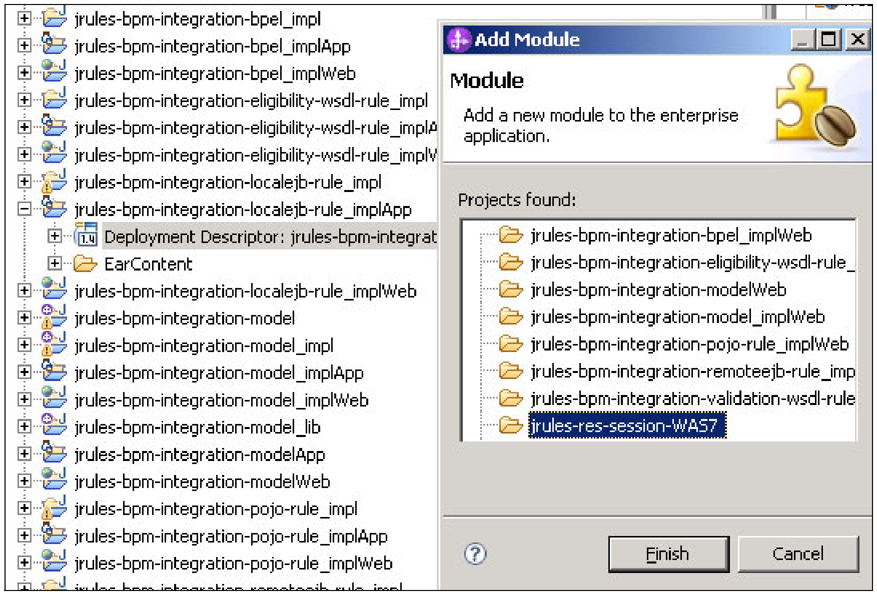
Projects found:

jrules-bpm-integration-bpel_implWeb
jrules-bpm-integration-eligibility-wsdl-rule_
jrules-bpm-integration-modelWeb
jrules-bpm-integration-model_implWeb
jrules-bpm-integration-pojo-rule_implWeb
jrules-bpm-integration-remoteejb-rule_imp
jrules-bpm-integration-validation-wsdl-rule
jrules-res-session-WAS7

Figure 10-27 Add a new SCA module

6. Save the deployment descriptor.
7. Expand the `jrules-bpm-integration-localejb-rule_implWeb` project and Edit the Deployment Descriptor. Navigate to the References tab and the reference to `eis/XUConnectionFactory` resource.
8. Click **Add** to add a reference, and select the **EJB Reference** button.
9. Click next and enter `ejb/IlrStatelessRuleSessionLocalEJB` as the name. Expand the `jrules-bpm-integration-localejb-rule_implApp` and select **IlrStatelessRuleSessionEJB** from the tree. Ensure that Create an EJB Client is selected and Ref Type is set to Local.
10. Click **Finish** and save the deployment descriptor, as shown in Figure 10-28 on page 267.

Note: While working on this solution, we observed that these changes do not remain; the changes revert to the previous settings when we clean all projects. The only way to fix this problem was to redo steps 5 on page 266 through 10 on page 266 in a clean workspace.


jrules-bpm-integration-bpel_impl
jrules-bpm-integration-bpel_implApp
jrules-bpm-integration-bpel_implWeb
jrules-bpm-integration-eligibility-wsdl-rule_impl
jrules-bpm-integration-eligibility-wsdl-rule_implA
jrules-bpm-integration-eligibility-wsdl-rule_implV
jrules-bpm-integration-localejb-rule_impl
jrules-bpm-integration-localejb-rule_implApp
Deployment Descriptor: jrules-bpm-integrat
EarContent
jrules-bpm-integration-localejb-rule_implWeb
jrules-bpm-integration-model
jrules-bpm-integration-model_impl
jrules-bpm-integration-model_implApp
jrules-bpm-integration-model_implWeb
jrules-bpm-integration-model_lib
jrules-bpm-integration-modelApp
jrules-bpm-integration-modelWeb
jrules-bpm-integration-pojo-rule_impl
jrules-bpm-integration-pojo-rule_implApp
jrules-bpm-integration-pojo-rule_implWeb

Module

Add a new module to the enterprise application.

Projects found:

jrules-bpm-integration-bpel_implWeb
jrules-bpm-integration-eligibility-wsdl-rule_
jrules-bpm-integration-modelWeb
jrules-bpm-integration-model_implWeb
jrules-bpm-integration-pojo-rule_implWeb
jrules-bpm-integration-remoteejb-rule_imp
jrules-bpm-integration-validation-wsdl-rule
jrules-res-session-WAS7

Figure 10-28 Add EJB reference

11. The mediation flow and related transformations created for the POJO interface can be reused for this step also. Follow step 1 on page 257 through step 20 on page 262 to create the required transforms and deploy the modules.

   Tip: You can also copy the artifacts from the following project to this project and recreate the binding in the assembly diagram:

   ```
   jrules-bpm-integration-pojo-rule_impl
   ```

Remote EJB integration using LA71 plug-in

In this section, we implement the integration component for executing the Data Validation rule set from the *Validate* process, shown in the BPEL diagram in Figure 10-3 on page 243, using the Remote EJB integration pattern.

We assume the rule set is already deployed in the RES and the BPEL process, which will call the rule set, is available in WebSphere Integration Developer.

We implement the Integration Component shown in Figure 10-29.

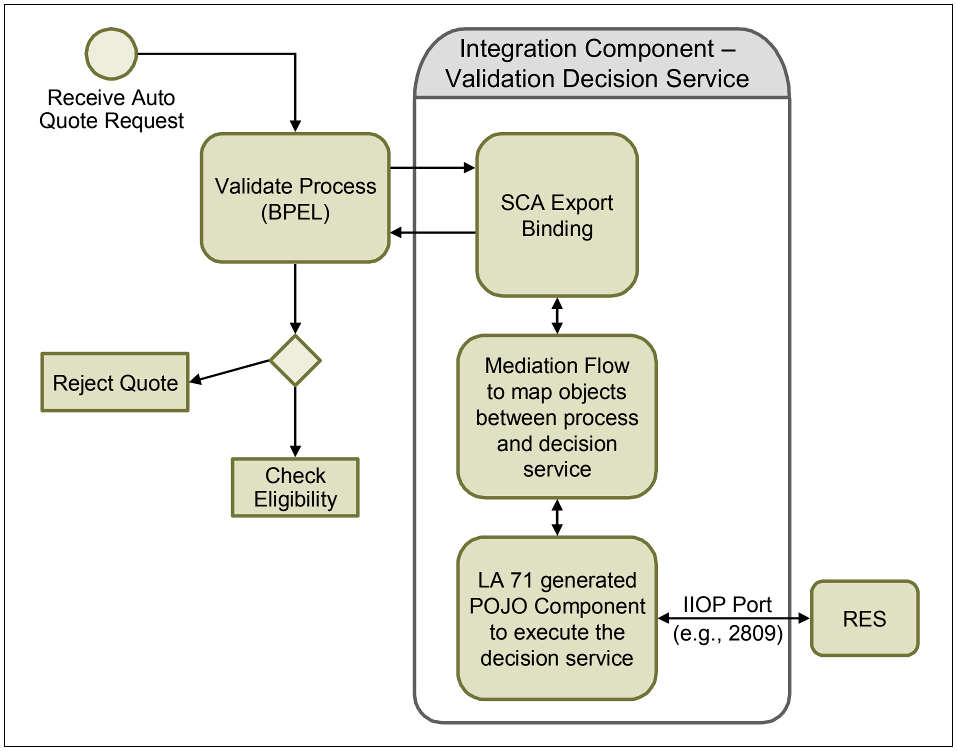

Receive Auto Quote Request
Validate Process (BPEL)
Reject Quote
Check Eligibility
Integration Component – Validation Decision Service
SCA Export Binding
Mediation Flow to map objects between process and decision service
LA 71 generated POJO Component to execute the decision service
IIOP Port (e.g., 2809)
RES

Figure 10-29 Implementation steps to integrate decision into process using remote EJB

Use the following procedure to develop a remote EJB component:

1. Follow steps 1 on page 252 through 10 on page 256. Choose a different module name. We used the following name:

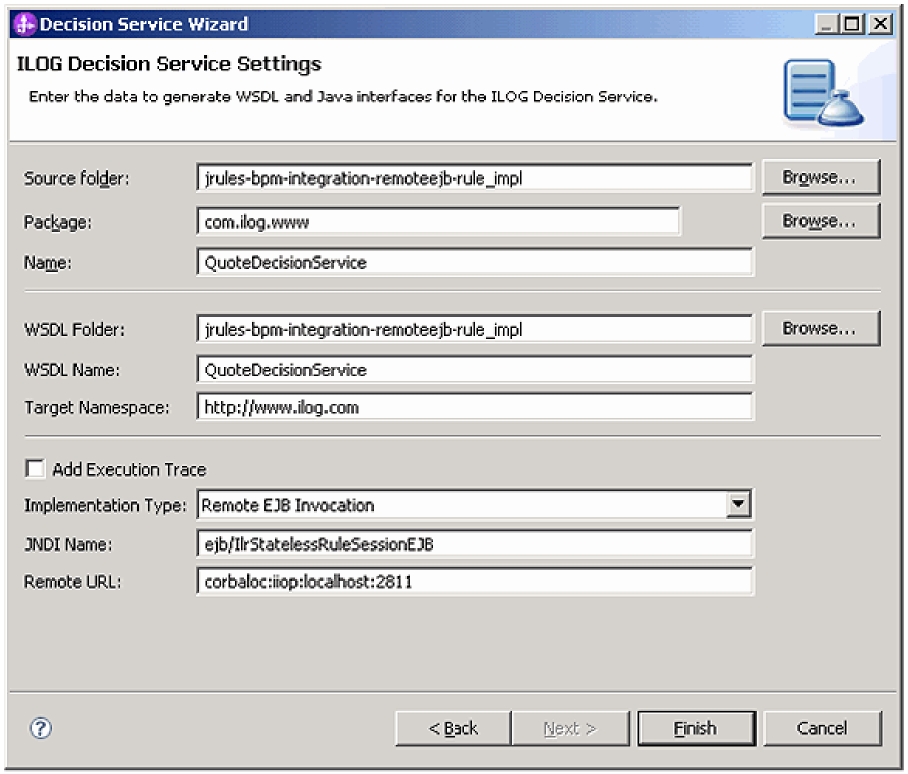
   `jrules-bpm-integration-remoteejb-rule_impl`

2. Select **Remote EJB Invocation** as the Implementation Type, as shown in Figure 10-30 on page 269. Use the following values:

   – `ejb/IlrStatelessRuleSessionEJB` as the JNDI Name
   – `entercorbaloc:iiop:localhost:` *`portnumber`* for the remote URL

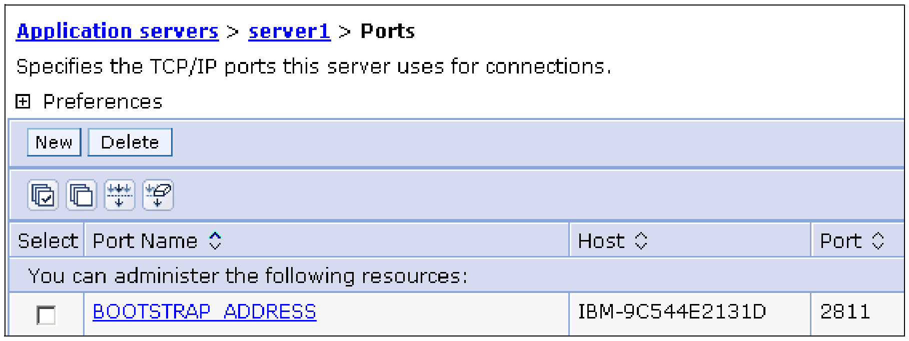
   Be sure to substitute the bootstrap *`portnumber`* with the bootstrap port number of your WebSphere Process Server run time.

   Tip: The bootstrap port number is in the administrative console of your WebSphere Process Server runtime environment, shown in Figure 10-31.


Decision Service Wizard
ILOG Decision Service Settings
Enter the data to generate WSDL and Java interfaces for the ILOG Decision Service.
Browse...
Browse...
Browse...
< Back
Next >
Finish
Cancel

Source folder: jrules-bpm-integration-remoteejb-rule_impl
Package: com.ilog.www
Name: QuoteDecisionService
WSDL Folder: jrules-bpm-integration-remoteejb-rule_impl
WSDL Name: QuoteDecisionService
Target Namespace: http://www.ilog.com
Add Execution Trace [ ]
Implementation Type: Remote EJB Invocation
JNDI Name: ejb/IlrStatelessRuleSessionEJB
Remote URL: corbaloc:iiop:localhost:2811

Figure 10-30 Configure binding options and set remote EJB invocation method



Specifies the TCP/IP ports this server uses for connections.

Select	Port Name	Host	Port
You can administer the following resources:			
☐	BOOTSTRAP_ADDRESS	IBM-9C544E2131D	2811

Figure 10-31 Change bootstrap port number to WebSphere Process Server port number

3. Click **Finish** to complete.

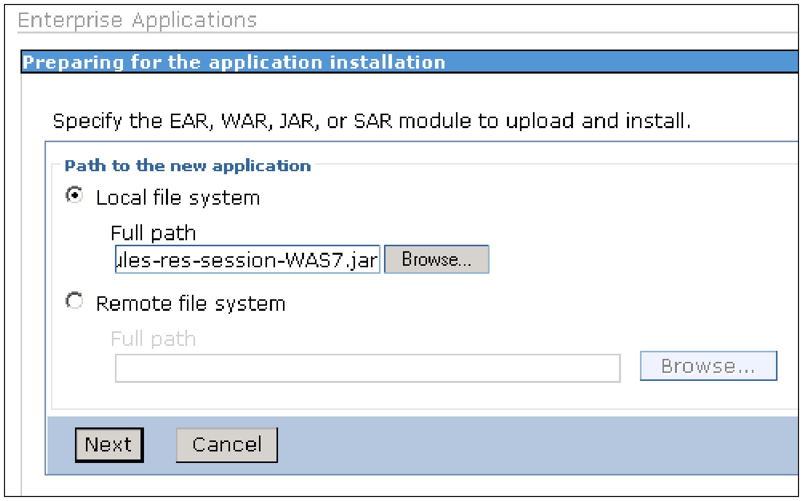
4. The mediation flow and related transformations created for the POJO interface can be reused for this step also. Follow steps 1 on page 257 through 20 on page 262 to create the required transforms, and deploy the modules.

Tip: You can also copy the artifacts from the following project to this project and recreate the binding in the assembly diagram:

```
jrules-bpm-integration-pojo-rule_impl
```

5. Ensure that the JRules session EJB JAR file is deployed as a stand-alone application on the process server instance by performing the following steps:

   a. From administrative console, navigate to Install a New Application (**Applications** → **New Application** → **New Enterprise Application**) and select the file `jrules-bpm-integration-ruleapp.jar` file. See Figure 10-32.


Enterprise Applications
Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

(●) Local file system

Full path: ules-res-session-WAS7.jar

( ) Remote file system

Full path
Browse...
Next
Cancel

Figure 10-32 New Enterprise Applications

   b. Click **Next**, expand **Choose to generate default bindings and mappings** and select the **Generate Default Bindings** check box. Click **Next**. Keep the defaults for the subsequent windows, and save the application.

   c. Ensure that the `jrules-res-session-WAS7_jar` application is started before testing the remote EJB invocation.

Integration by using web services and WSDL

In this section, we implement the integration component for executing the Data Validation rule set from the *Validate* process, shown in the BPEL diagram in Figure 10-3 on page 243 using HTDS service provided by JRules.

We assume the rule set is already deployed in the RES and the BPEL process, which will call the rule set, is available in WebSphere Integration Developer. We also assume that HTDS is deployed on the RES and the WSDL for the rule set is already generated.

We implement the Integration Component shown in Figure 10-33 on page 272.

Note: Use RES console to generate the WSDL. For details, see the JRules documentation.

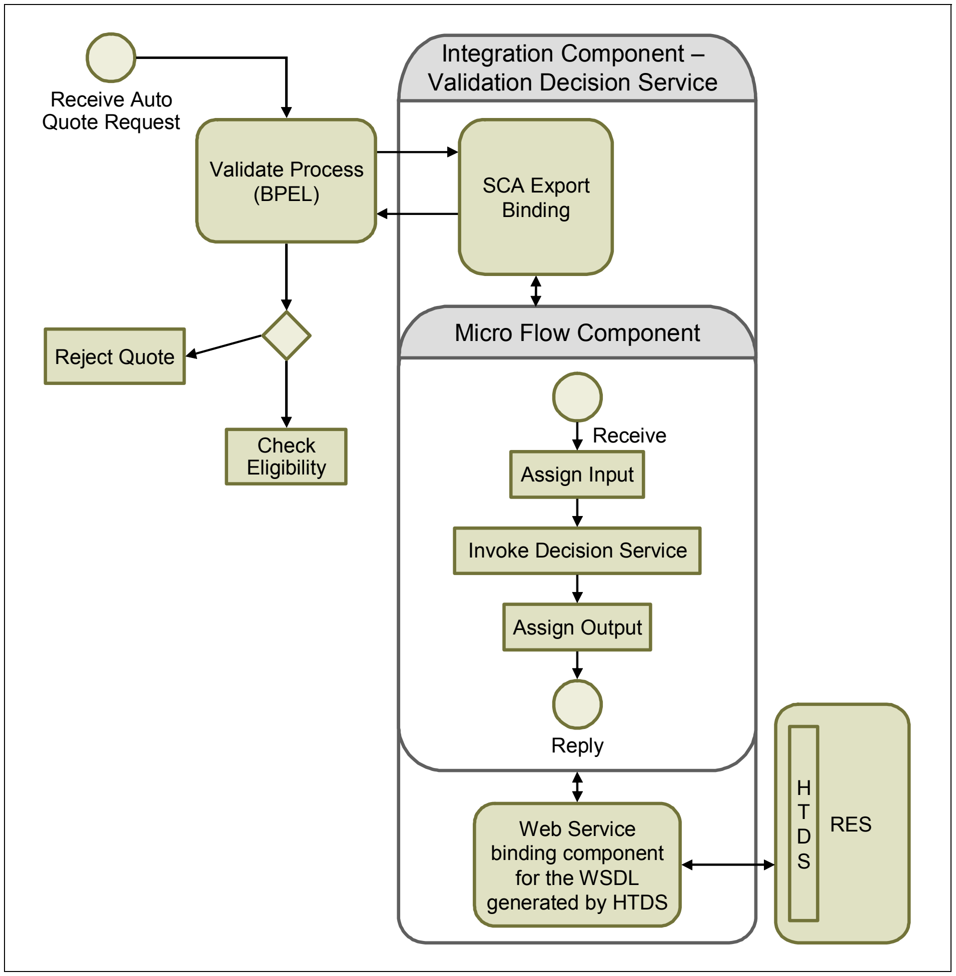
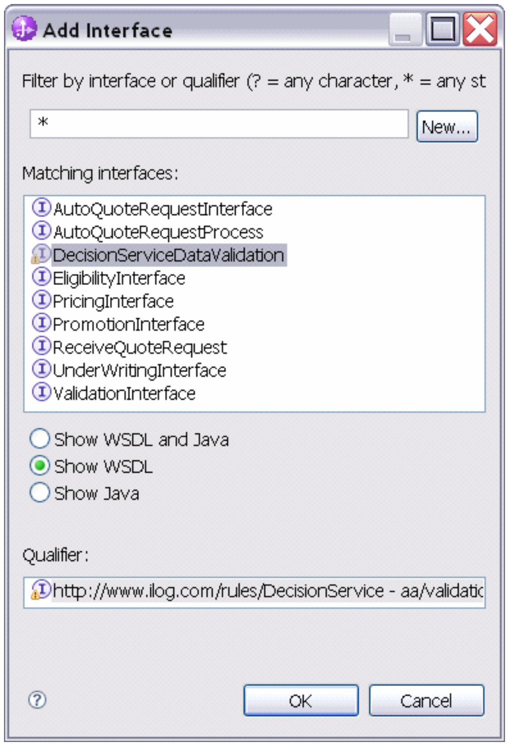


Figure 10-33 Implementation steps to integrate decision into process using web services

Use the following procedure to develop the web service integration component:

1. Save the WDSL that is generated from the RES console in your file system (`validation.wsdl`).
2. Create a module in WebSphere Integration Developer (we named it `jrules-bpm-integration-validation-wsdl-rule_impl`). This module should refer to the library project (`jrules-bpm-integration-model_lib`) already created.
3. Right-click the module and select **Import**, and then select the **WSDL and XSD** option under Business Integration. Navigate to the directory where the WSDL file is saved and select the `validation.wsdl` file.

4. Drag **Import** from the left palette to the assembly diagram.
   Rename it to `DecisionServiceDataValidationImport`.
   Add the `DecisionServiceDataValidation` interface to it, as shown in Figure 10-34


Add Interface

Filter by interface or qualifier (? = any character, * = any st

*

Matching interfaces:

- AutoQuoteRequestInterface
- AutoQuoteRequestProcess
- DecisionServiceDataValidation
- EligibilityInterface
- PricingInterface
- PromotionInterface
- ReceiveQuoteRequest
- UnderWritingInterface
- ValidationInterface

( ) Show WSDL and Java
[✓ Show WSDL]
( ) Show Java

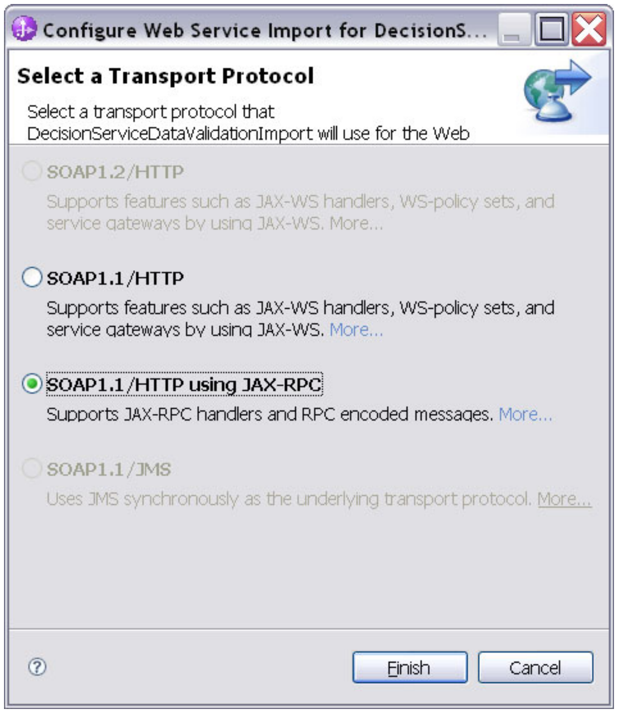
Qualifier:

http://www.ilog.com/rules/DecisionService - aa/validatic

Figure 10-34 Add new interface for Validation decision

5. Right-click **DecisionServiceDataValidationImport** and select **Generate Binding,** and then select **Web Service Binding**.
6. Select **Use an existing web service port** and click **Browse.** Select the port, generated by the WSDL import.

7. Select the SOAP format **JAX-RPC**, as shown in Figure 10-35. Click **Finish**.


Configure Web Service Import for DecisionS...

Select a Transport Protocol

Select a transport protocol that DecisionServiceDataValidationImport will use for the Web

( ) SOAP1.2/HTTP
Supports features such as JAX-WS handlers, WS-policy sets, and service gateways by using JAX-WS. More...

( ) SOAP1.1/HTTP
Supports features such as JAX-WS handlers, WS-policy sets, and service gateways by using JAX-WS. More...

(●) SOAP1.1/HTTP using JAX-RPC
Supports JAX-RPC handlers and RPC encoded messages. More...

( ) SOAP1.1/JMS
Uses JMS synchronously as the underlying transport protocol. More...

Figure 10-35 Select SOAP protocol

8. Create a simple process flow by dragging the **Process** icon from the palette to the assembly diagram and rename it to the following name:

   `ValidationInterfaceToValidationDecisionProcess`

9. Add the ValidationInterface to the process by right-clicking **Process Add-Interface**.

10. Create a reference partner link between the process and `DecisionServiceDataValidationImport`.

> **Note:** Use a Microflow with Assign (or Datamap) to implement the mediation instead of a mediation flow, because we encountered namespace conflicts with HTDS-generated WSDLs when used with XSLT transforms.

11. Generate an SCA binding for the process by right-clicking **Process Generate Export - SCA Binding,** and save the assembly diagram.

12. Launch the Process Flow editor to implement the process by double-clicking the process in assembly diagram, selecting **Yes** to implement it, and clicking **OK** to leave the default folder.

13. Add a step to Invoke the DecisionServicePartner, as shown in Figure 10-36:

    a. Expand the following path: **ReferencePartners** → **DecisionServiceDataValidationPartner** → **DecisionServiceDataValidation** → **executeDecisionService**

    b. Drag **executeDecisionService** between the Receive and Reply steps of the flow. Rename it to `InvokeDecisionService`.

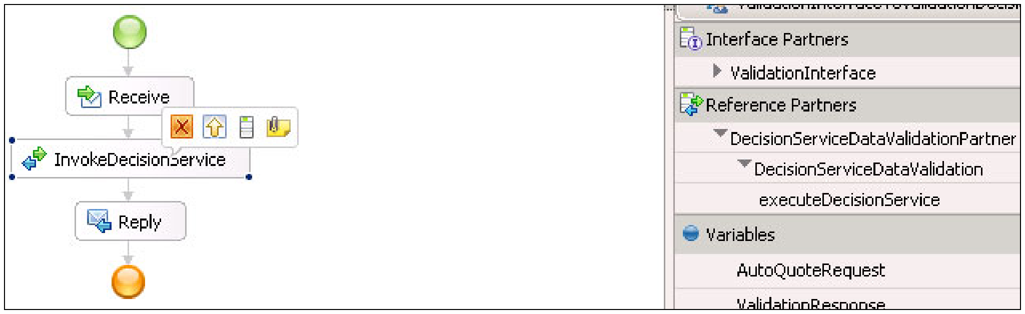
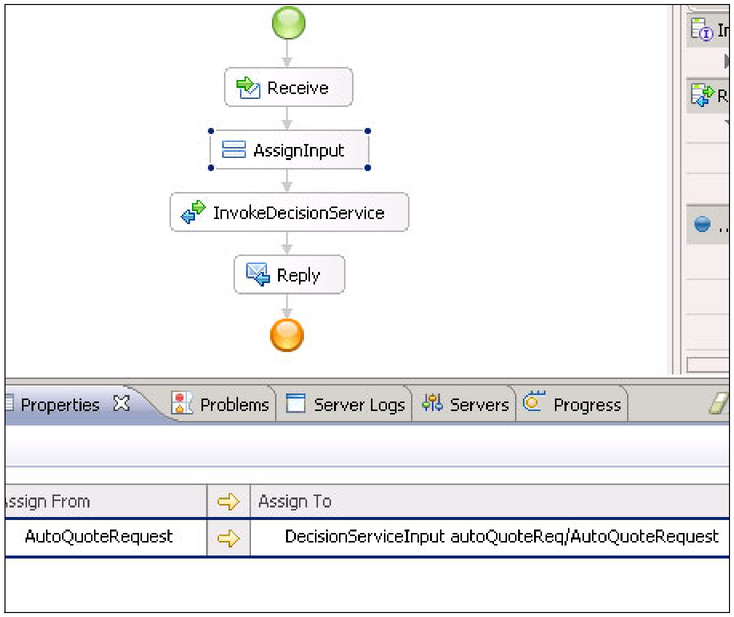


Figure 10-36 Add Invoke decision service step

14. Assign input and output to the Invoke decision. Click the InvokeDecisionService **Properties** tab and select **Details**. Click **None** for Inputs, click **New** from the drop-down menu, and enter `DecisionServiceInput` as the Name. Repeat the steps for Outputs and name it `DecisionServiceOutput`.

15. Map the AutoQuoteRequest Input to the DecisionServiceInput. Drag the assign task from the palette before the Invoke step. Rename it to `AssignInput`. Select **AutoQuoteRequest** for Assign From.

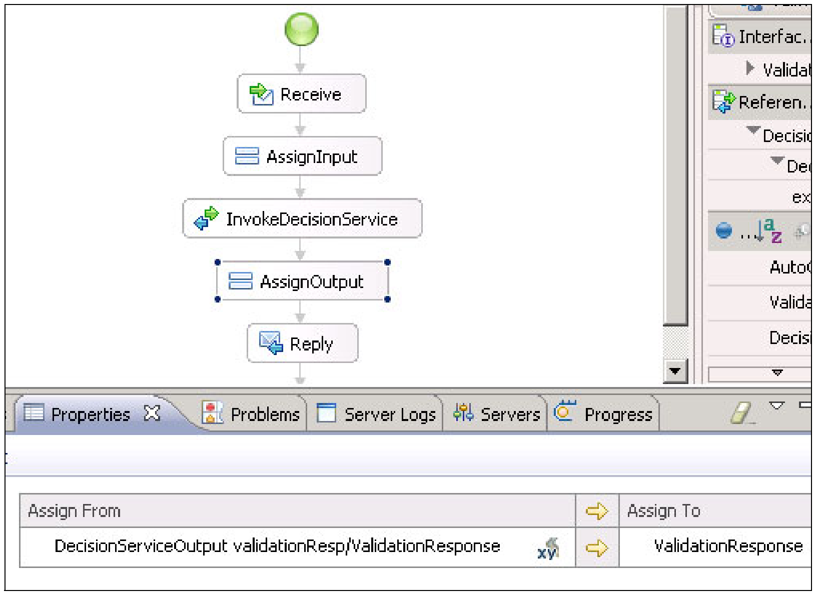
    For Assign To, expand **DecisionServiceInput** → **autoQuoteReq** and select AutoQuoteRequest, as shown in Figure 10-37 on page 276.



ssign From		Assign To
AutoQuoteRequest	⇨	DecisionServiceInput autoQuoteReq/AutoQuoteRequest

Figure 10-37 Add AssignInput

16. Map the DecisionServiceOutput to ValidationResponse. Drag the assign task from the palette after the Invoke step. Rename it to `AssignOutput`. For Assign From, expand **DecisionServiceOutput** → **validationResp** and select **ValidationResponse**. For Assign To, pick **ValidationResponse**, as shown in Figure 10-38 on page 277.



Assign From		Assign To
DecisionServiceOutput validationResp/ValidationResponse		ValidationResponse

Figure 10-38 Add AssignOutput

17. Save and close the process and the assembly diagram.
18. Follow standard procedures to build and deploy the modules to WebSphere Process Server.

Note: You must create a separate module for each WSDL that is generated in previous steps. See the Known Limitations link at the following address:

`ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/la71_readme.html`

10.6 Summary

We have shown a few ways to integrate WebSphere Process Server with ILOG JRules. At the time of writing this book, we encountered issues with using Local EJB integration. Because a similar behavior can be obtained by using a Remote EJB invocation against local host, that approach is the better one.

In addition, the Remote EJB java code that is generated contains a hard-coded reference to the host name and port of the remote EJB container. Therefore, be sure to change the Java code to use a property file or WebSphere variable to obtain the reference. This approach can help you more easily migrate to higher environments (preproduction, production) without having to change the code.

11

Integrating JRules with WebSphere Message Broker

In this chapter, we provide scenarios for integrating JRules with WebSphere Message Broker.

This chapter contains the following topics:

- Introduction
- Scenario 1: J2SE
- Scenario 2: Web services (HTDS) integration
- Scenario 3: JMS Integration
- Scenario 4: SCA integration
- Summary

11.1 Introduction

The IT architecture team within IBM Redbooks Company Insurance has recognized that implementing business rules within its WebSphere Message Broker flows is not good practice. The new approach for projects is to develop an appropriate business rule within WebSphere ILOG JRules and invoke that from within WebSphere Message Broker.

In the previous sections of this book, we identified the type of integration pattern that is appropriate for your requirements in the connectivity layer. This step can probably also be determined by other topology and deployment requirements and considerations. You have probably identified one or two integration patterns you now need to implement. In this chapter, we explore several of these patterns by using WebSphere Message Broker V7.0. We assume that you are familiar with the implementation of solutions based on this product, but that you may not necessarily have experience in several product features and technologies we use in these scenarios.

11.1.1 Scenarios overview

WebSphere Message Broker can use WebSphere ILOG JRules in a wide range of scenarios.

The following patterns help to categorize major aspects of these scenarios:

- Integration pattern
  - Tightly coupled: J2SE
  - Loosely coupled: Web services, Java Messaging Service (JMS), or SCA
- Decision pattern

  Message routing, message enrichment, or both
- Development pattern
  - Development of new message flows that use JRules business rules
  - Change of existing message flows (to replace existing business logic in message flows with JRules business rules)

In this chapter, we describe the four scenarios listed in Table 11-1.

Table 11-1 Pattern scenarios

Scenario	**Integration pattern**	**Decision pattern**	**Development pattern**
1	Tightly coupled: J2SE	Message routing	New
2	Loosely coupled: Web service	Message routing	New
3	Loosely coupled: JMS	Message routing and message enrichment	Change
4	Loosely coupled: SCA	Message routing and message enrichment	New

11.1.2 Decision patterns

We explore two decision patterns in this chapter:

- Message Routing
- Message Enrichment

These patterns are used in the IBM Redbooks Company Insurance auto insurance quote system.

IBM Redbooks Company Insurance receives requests for car insurance quotes from a variety of channels. These channels might be from the general public using the IBM Redbooks Company Insurance web site, from external insurance broker systems, or from existing applications within IBM Redbooks Company Insurance. These requests are all initially processed within WebSphere Message Broker, which transforms the request into the correct format for the back-office quote system and then routes the request to that quote system for execution. The response is then passed back through WebSphere Message Broker for delivery to the requesting client.

Message routing

Various business studies have concluded that the risks associated with certain types of vehicles are excessive, and a business decision has been made to avoid that risk by offering such business to partners of IBM Redbooks Company Insurance. Those partners are specialists in certain types of higher risk policies. However, IBM Redbooks Company Insurance wants to offer a seamless insurance quote experience to its clients so the company wants to *route* such quote requests to the most appropriate partner automatically, without it being obvious that a third-party insurer is involved.

Message enrichment

Recent analysis has concluded that customers typically over-exaggerate the value of their vehicle on average by 15% when making their quote. Because IBM Redbooks Company Insurance currently uses this value to calculate maximum liability under the auto insurance policy, this position represents an incorrect, overly pessimistic position. The company therefore carries out its own, more accurate, estimation of the value of cars in insurance quote requests. The calculation is based on data that is supplied in the prospective customer's quote request and the more accurate estimate is used as the basis for the quote.

11.1.3 Development and use of the download file

The accompanying download file for this chapter contains all of the WebSphere Message Broker Toolkit, JRules Studio, and WebSphere Integration Developer workspaces. This file enables you to more easily set up and test each scenario.

This chapter is focused on WebSphere Message Broker development. It therefore details how the WebSphere Message Broker artifacts were created, but does not detail how to create the JRules Studio artifacts.

All three scenarios in this chapter use the following rule application project:

```
jrules-messaging-integration-ruleapp
```

The rule application contains rules from the following rule set projects:

- For message routing rules:

  ```
  jrules-routing-integration-rules
  ```

- For message enrichment rules:

  ```
  jrules-enrichment-integration-rules
  ```

- The same rule set, but uses a Java XOM-based BOM for the SCA scenario (In the `jrules-messaging-integration.zip` file, use the workspace workspaceJavaForSCA).

11.2 Scenario 1: J2SE

In this section, we look at the first scenario, J2SE.

11.2.1 Introduction

Following a review of the profitability of the company's auto insurance business, IBM Redbooks Company Insurance has decided to focus on the insuring of modern vehicles only. The company has negotiated with partners that will underwrite insurance for vehicles over a certain age. To support this new requirement, all quote requests will now be subject to an invocation of a business rule, which checks the vehicle age and determines whether to route the request to the company's default internal quote system or a partner quote system. Currently, auto quote requests are received on a WebSphere MQ queue, and you have decided to implement a ESB gateway message flow to invoke the rules and to route the request accordingly.

You have collaborated with the WebSphere ILOG JRules developer to agree on an appropriate deployment pattern. You select a tightly coupled J2SE integration because of the following factors:

- Message flow performance is a key factor in the implementation, because of requirements for quote request processing times. An imperative is to have the shortest possible latency for invocation of a rule. Performance is one of the highest priority requirements over other non-functional aspects, such as the ease of solution implementation.
- Your developers are suitably experienced in Java development in WebSphere Message Broker and using the JRules APIs.

Overview of products

This J2SE integration scenario uses the following products and version numbers:

- WebSphere ILOG JRules 7.0.3
- WebSphere Message Broker 7.0
- WebSphere Message Broker Toolkit 7.0
- WebSphere MQ 7.0.1

The scenario uses a single machine for the development and testing of the integration. Clearly, this is not representative of a physical topology that would be used in a production deployment, but it does not materially affect the integration use case or the guidance in this scenario.

The tooling and development are supported by the following products:

- WebSphere ILOG JRules Rule Studio 7.0.3
- WebSphere Message Broker Toolkit 7.0

The runtime environment is supported by the following products:

- A WebSphere Message Broker broker, named BRKR
- A WebSphere MQ queue manager, named BRKR.QM

Figure 11-1 shows the runtime and tooling environments in this scenario.

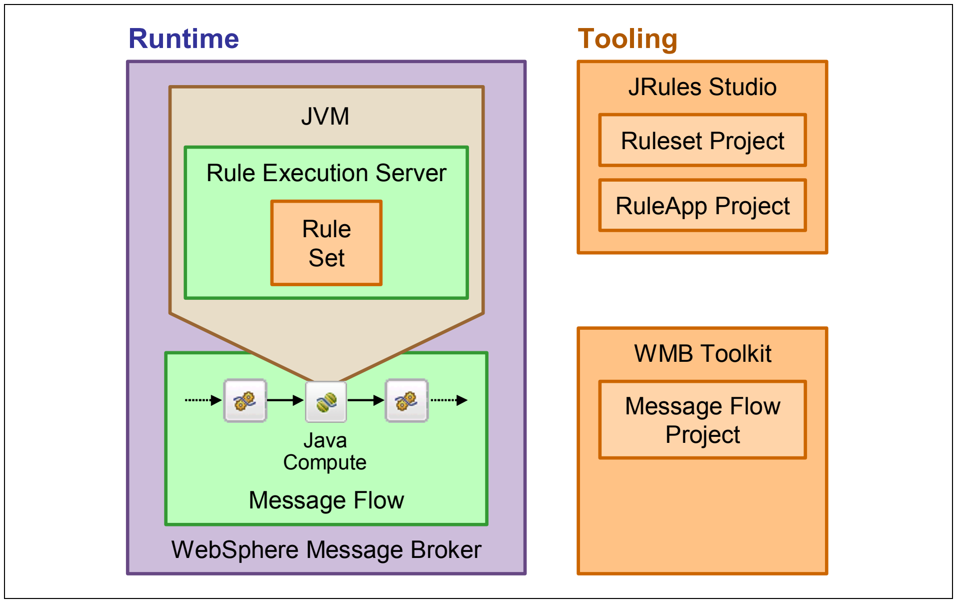


Figure 11-1 Scenario overview

11.2.2 Develop WebSphere Message Broker resources

In this section, we look at developing WebSphere Message Broker resources.

Overview of the message flow

The scenario involves creating a new message flow, as shown in Figure 11-2 on page 285. The Rule Execution Server (RES) will be embedded in the WebSphere Message Broker JVM and invoked from a JavaCompute node. The rule set response parameter is used to dynamically set the name of the output queue for the MQOutput node.

Two queues are used:

- One for the internal default quote system
- One for external partner quotes

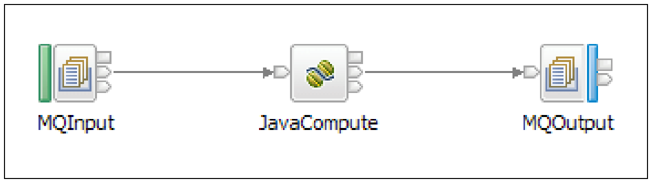


Figure 11-2 New flow

Message flow development is carried out in the WebSphere Message Broker Toolkit. The Toolkit is used to develop a new message flow called AutoQuoteGatewayJ2SE.

Develop the new message flow

We do not focus on all the development activities to build the solution, because the assumption is that you are familiar with message flow development and in particular the Java API that is provided by WebSphere Message Broker, which is used within a JavaCompute node and the ILOG JRules RES API. Instead, we focus on the significant and relevant parts of the solution.

The new AutoQuoteGatewayJ2SE message flow will get a WebSphere MQ input message containing an AutoQuoteRequest XML message and put it on one of two output queues based on a routing rule.

The function of each node is as follows.

- MQInput
  - Reads an AutoQuoteRequest XML message from a WebSphere MQ queue.
  - Sets the queue name (for example, JR_J2SE_IN).
- JavaCompute
  - Uses the RES API to execute routing rules. The message payload is used as input to the rules. The rule response is used to dynamically set a queue name used by the MQOutput node.

- MQOutput
  - Writes the AutoQuoteRequest XML message to the WebSphere MQ queue set in the JavaCompute node.
  - Sets the queue manager name (for example. BRKR.QM). Setting a queue name is not necessary because it is set dynamically in the JavaCompute node.
  - Sets the destination mode to distribution list.

Develop the JavaCompute node code and project resources

The JavaCompute code is designed to perform the following tasks:

- Extract the payload from the input message.
- Prepare the input rule set parameters.
- Call the rule execution.
- Get the output rule set parameters.
- Use the output rule set parameters to dynamically set the queue name for the MQOutput node.
- Pass the input message to the out terminal.

To use the RES API from within the JavaCompute code, the Java project needs access to various JRules resources, for example, JRules JAR files.

Notes:

- The Java code and resource configuration that are required to implement this scenario is beyond the scope of this book. For examples of J2SE RES API code, usage, and resources, see the Java SE rule session sample.
- Creating a message set is not necessary because, in this scenario, the entire message flow input message is passed to the RES and therefore the broker does not need to understand the detailed format of the message payload.

Approaches to J2SE deployment

The two possible approaches to a JS2E solution involving WebSphere Message Broker are as follows:

- J2SE Rule Execution Server (J2SE RES)

  With this approach, the RES is embedded in the broker JVM and accessed directly by, for example, a JavaCompute node, using the RES API as suggested and described in this book.

- J2SE rule engine

  This approach uses the JRules rule engine API (for example, IlrContext and IlrRuleset) and is not described in this book.

We examine integration with J2SE RES instead of a solution using the J2SE rule engine because of the following comparative deployment considerations:

- The J2SE RES approach:
  - Offers better performance through connection pooling of execution engines.
  - Means sharing decision services externally from WebSphere Message Broker is difficult.
- The J2SE rule engine approach:
  - Offers reduced capability in rule management features:
    - No hot deployment from Rule Team Server
    - No connection pooling to execution engine
    - No web management console
  - Does not support decision warehousing functionality.

Message flow design

The Java user-defined node can be used as an alternative to the JavaCompute node. The JavaCompute node is generally simpler to develop, but the Java user-defined node offers the potential for enhanced functionality, for example, custom node terminals. It also offers the potential for improved reusability because the capability can be made available to message flow developers who are unfamiliar with Java.

The message routing that is described in the scenario relies on the MQOutput node queue name being set dynamically in the JavaCompute node. A preferred way is to design the flow to route messages by using a RouteToLabel node, as shown in Figure 11-3 on page 288.

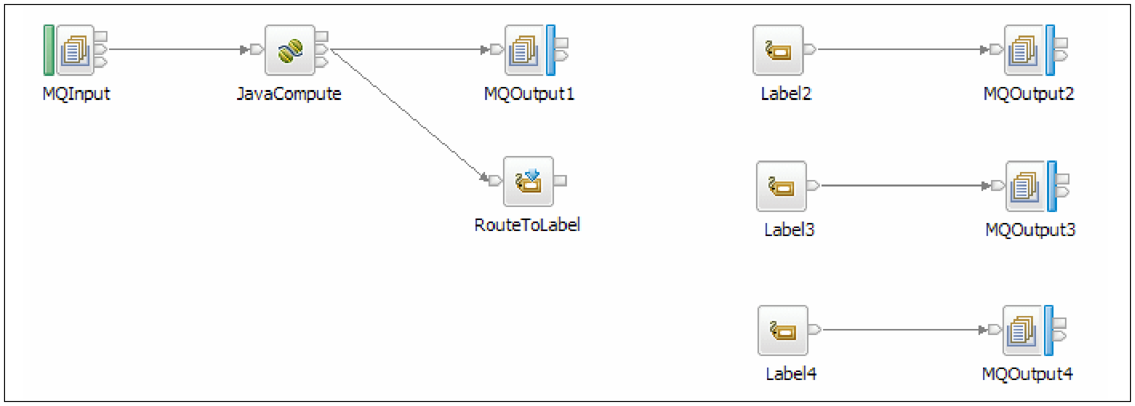


Figure 11-3 RouteToLabel node routing alternative

The message flow has been kept simple for illustrative purposes and therefore does not include important features and function, for example, error checking, failure/catch processing, and trace.

The scenario uses an identical format of messages for the input to the message flow and the input to the JRules rule application. Reasons exist why other scenarios would use differing formats, for example, the input message is large and only a subset of its content is required by the rule application. In cases where the message formats differ, WebSphere Message Broker provides powerful features to transform message content.

11.3 Scenario 2: Web services (HTDS) integration

In this section, we look at the second scenario, integration web services.

11.3.1 Introduction

Following a review of the profitability of its auto insurance business, IBM Redbooks Company Insurance has decided to focus on the insuring of modern vehicles only. The company has negotiated with partners who will underwrite insurance for vehicles over a certain age. To support this new requirement, all quote requests will now be subject to an invocation of a business rule, which checks the vehicle age and determines which is the most appropriate partner to route the request to. Currently, auto quote requests are received on a WebSphere MQ queue, and you have decided to implement an ESB gateway message flow to invoke the rules and to route the request accordingly.

You have collaborated with the WebSphere ILOG JRules developer to agree on an appropriate interface and the rule is already deployed. You now have to construct the message flow to support the requirement.

Overview of products

This web services (hosted transparent decision services) integration scenario requires the following products:

- WebSphere ILOG JRules 7.0.3
- A supported J2EE application server runtime environments, such as WebSphere Application Server 7.0
- WebSphere Message Broker 7.0
- WebSphere Message Broker Toolkit 7.0
- WebSphere MQ 7.0.1

The scenario uses a single machine for the development and testing of the integration. Clearly, this configuration is not representative of a physical topology that would be used in a production deployment, but it does not materially affect the integration usecase or the guidance in this scenario.

The tooling and development are supported by the following products:

- WebSphere ILOG JRules Rule Studio 7.0.3
- WebSphere Message Broker Toolkit 7.0

The runtime environment is supported by the following products:

- A WebSphere Message Broker broker, named BRKR
- A WebSphere MQ queue manager, named BRKR.QM
- A J2EE runtime environments that is supported by ILOG JRules, for example, WebSphere Application Server, which hosts RES with the following deployed rule application:

  ```
  jrules-messaging-integration-ruleapp
  ```

Figure 11-4 on page 290 shows the runtime and tooling environments in this scenario.

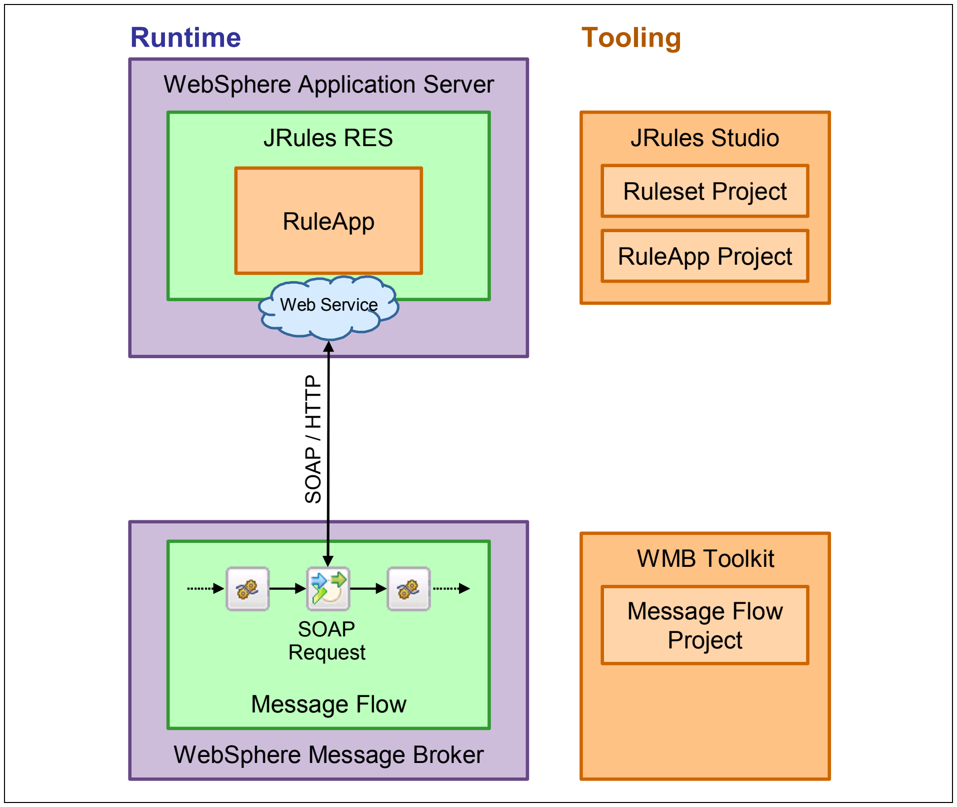


Figure 11-4 Scenario overview

11.3.2 Prepare JRules

This section gives an overview of deploying the RES to WebSphere Application Server and deploying the rule sets to the RES. The steps are as follows:

1. Deploy the RES to your chosen runtime environment. For instructions, see “Installing Rule Execution Server on WebSphere Application Server V7.0” (under WebSphere ILOG JRules 7.0.3: Java EE add-ons in the information center) at the following location:

   http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m3/topic/com.ibm.websphere.ilog.jrules.install.doc/Content/Business_Rules/Documentation/_pubskel/JRules_Application_Servers/ps_Installing_JRules_IC99.html

2. Start Rules Studio and open the JRulesStudio_Workspace_XML (included in the download).

3. Click the Project menu and select **Clean**.

4. Select **Clean All Projects** and click **OK**.
5. Right-click the **jrules-messaging-integration-ruleapp** project in the Rule Explorer and select **RuleApp**, then select **Deploy**.
6. Leave the deployment type as the default and click **Next**.
7. Click the **Create a temporary Rule Execution Server** configuration and enter the URL, Login, and Password for the RES.
8. Click **Finish** to deploy the rule application.
9. Open the RES console in your web browser (for example, `http://machine:9082/res`).
10. Enter the username and password of the RES administrator that you configured when deploying the RES and click **Sign in** to open the RES console.
11. Click **Explorer** and check that **JRulesMessagingIntegrationRuleApp** is listed in the RuleApps.

Your JRules run time is now ready to be used by WebSphere Message Broker. Perform *either* of the following steps:

- Use predeveloped message broker resources in the download, by following the instructions in 11.3.3, “Use predeveloped WebSphere Message Broker resources” on page 291
- Develop the message broker resources by following the instructions in 11.3.5, “Develop WebSphere Message Broker resources” on page 296.

11.3.3 Use predeveloped WebSphere Message Broker resources

You may use predeveloped message broker resources in the download, as described here, or develop the message broker resources by following the instructions first in 11.3.4, “Obtain the WSDL” on page 292, and then in 11.3.5, “Develop WebSphere Message Broker resources” on page 296.

To use predeveloped message broker resources, perform the following steps:

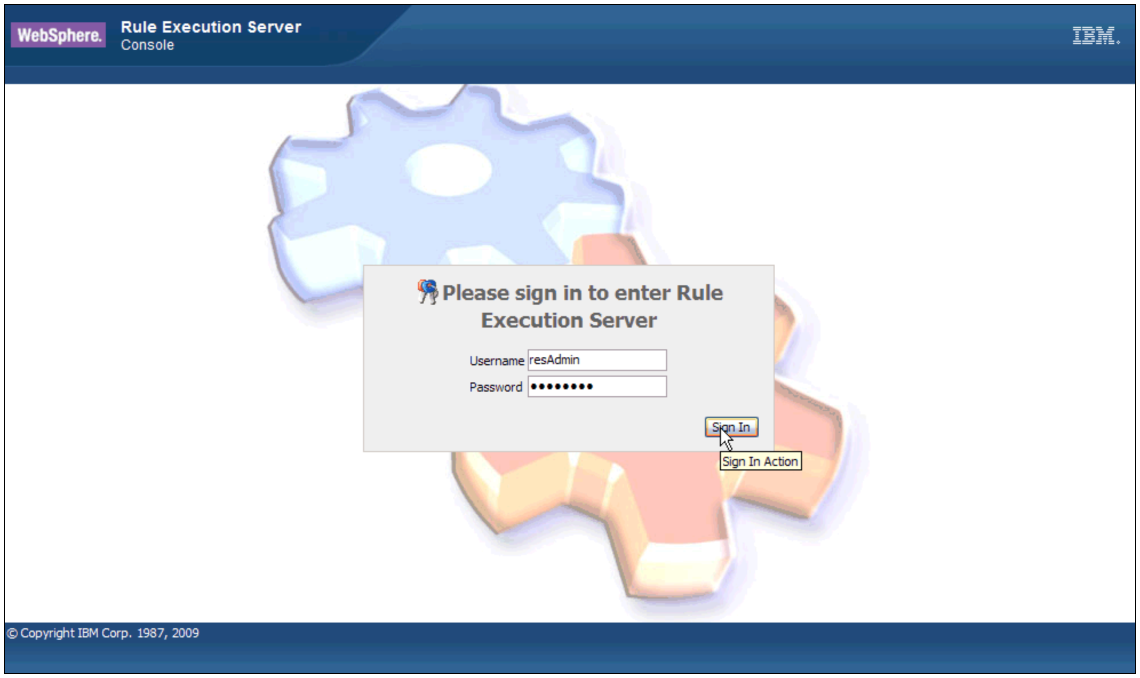
1. Start WebSphere Message Broker Toolkit and open the WMB_Workspace_WS file (included in the download, described in Appendix A, “Additional material” on page 359). You see the AutoQuoteGateway project in the Projects pane of your workspace.
2. Click **Project** → **Clean**, select **clean all projects** and click **OK**.

You can now go to 11.3.6, “Deploy the WebSphere Message Broker message flows” on page 309 to deploy the resources that are required for the solution.

11.3.4 Obtain the WSDL

The WebSphere Message Broker message flow uses WSDL that is provided by WebSphere ILOG JRules. The WSDL describes the interface for the web service (hosted transparent decision service). Obtain it as follows:

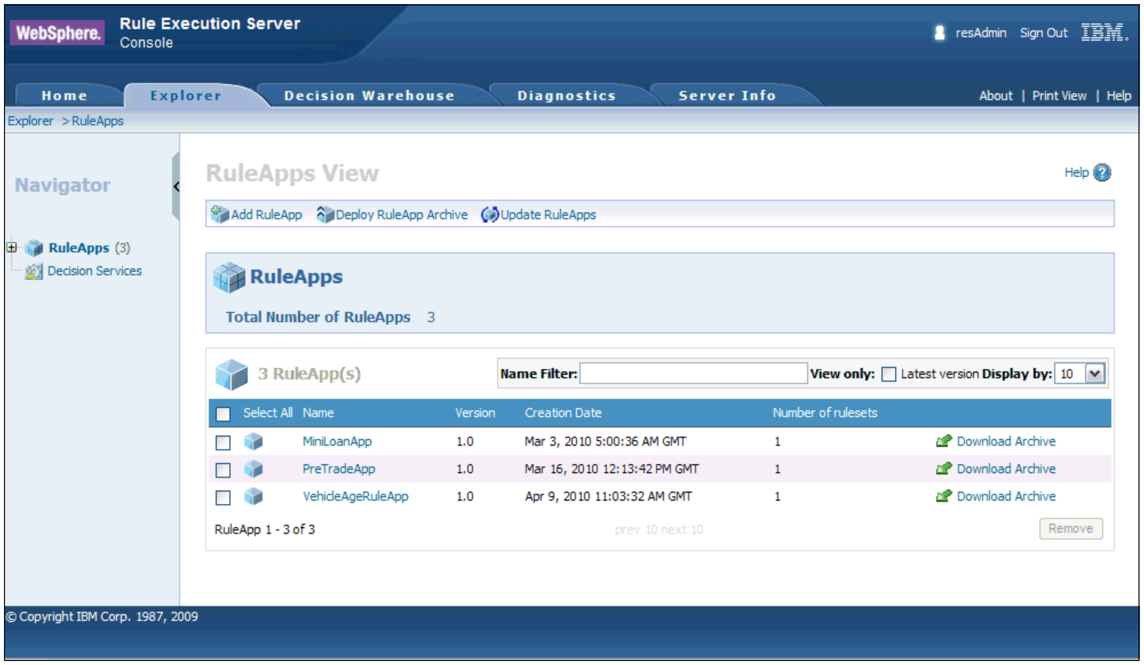
1. Log in to the Rule Execution Server (RES), as shown in Figure 11-5.


WebSphere.
Rule Execution Server
Console
IBM.
Please sign in to enter Rule Execution Server
Sign In
Sign In Action
© Copyright IBM Corp. 1987, 2009

Username resAdmin
Password ••••••••

Figure 11-5 Login to the Rule Execution Server

2. Select the **Explorer** tab. The deployed RuleApps are listed, as shown in Figure 11-6.



RuleApps

Total Number of RuleApps 3

3 RuleApp(s)

Name Filter: View only: Latest version Display by: 10

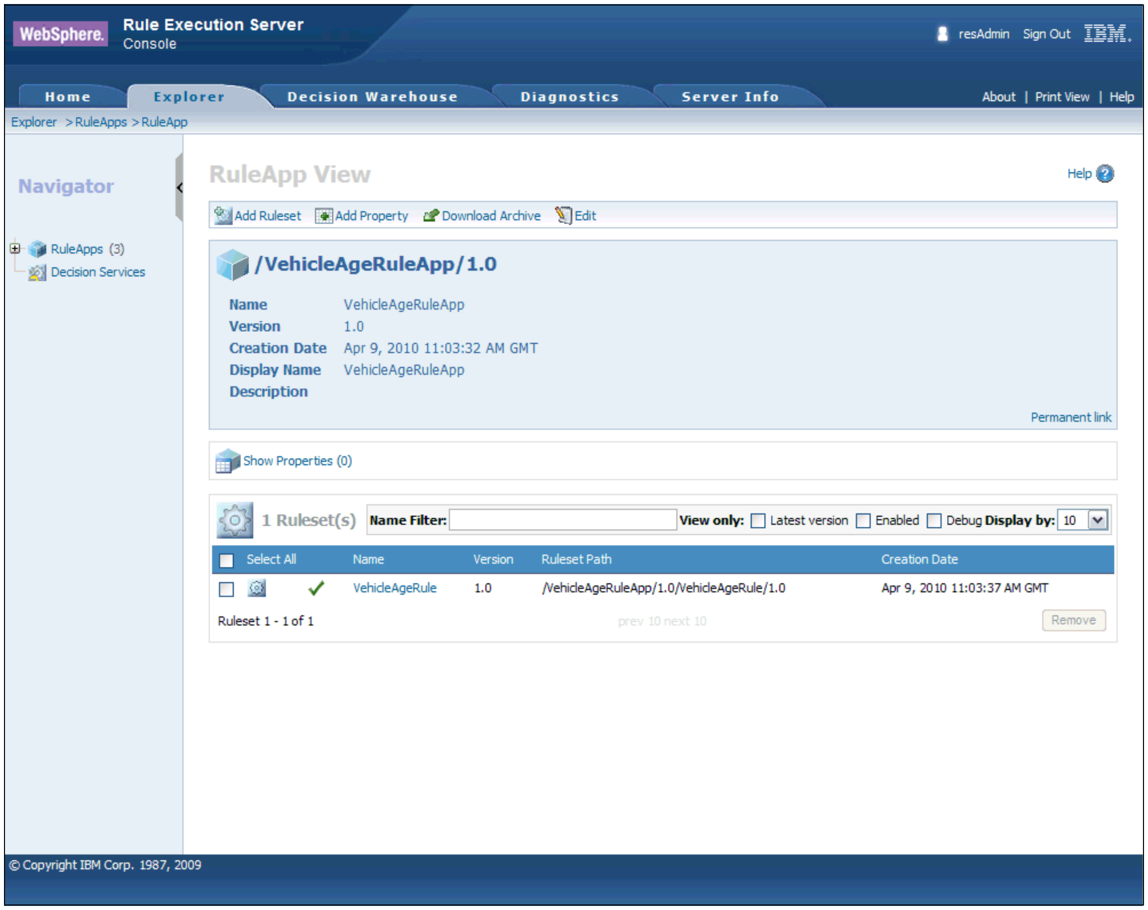
Select All	Name	Version	Creation Date	Number of rulesets	
☐	MiniLoanApp	1.0	Mar 3, 2010 5:00:36 AM GMT	1	Download Archive
☐	PreTradeApp	1.0	Mar 16, 2010 12:13:42 PM GMT	1	Download Archive
☐	VehicleAgeRuleApp	1.0	Apr 9, 2010 11:03:32 AM GMT	1	Download Archive

RuleApp 1 - 3 of 3 prev 10 next 10 Remove

© Copyright IBM Corp. 1987, 2009

Figure 11-6 Deployed RuleApps in Explorer view

3. Click the link for our rule, which in this case is VehicleAgeRuleApp. The view details page for that rule opens, as shown in Figure 11-7.


WebSphere. Rule Execution Server Console
resAdmin Sign Out IBM.
Home Explorer Decision Warehouse Diagnostics Server Info
About | Print View | Help
Explorer > RuleApps > RuleApp
Navigator
RuleApps (3)
Decision Services
RuleApp View
Help
Add Ruleset Add Property Download Archive Edit

/VehicleAgeRuleApp/1.0

Name	VehicleAgeRuleApp
Version	1.0
Creation Date	Apr 9, 2010 11:03:32 AM GMT
Display Name	VehicleAgeRuleApp
Description	

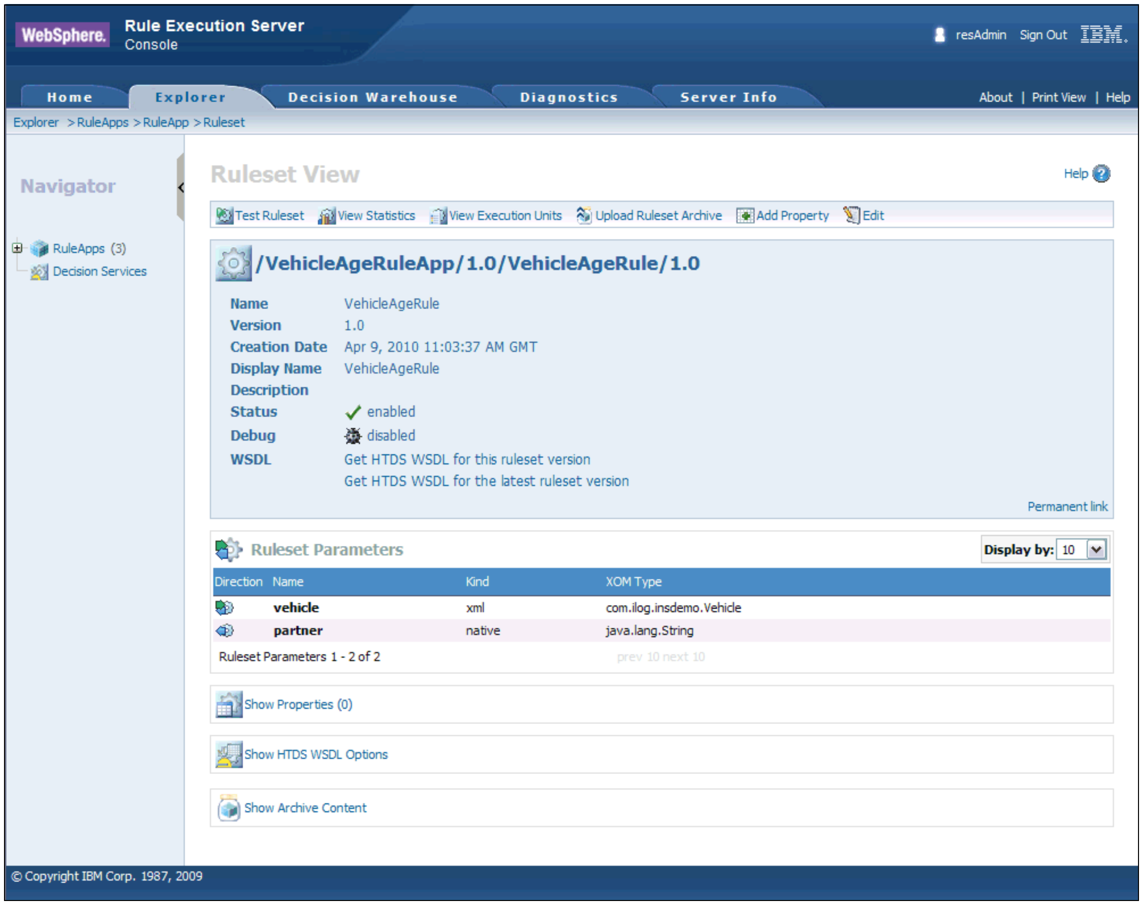
Permanent link
Show Properties (0)
1 Ruleset(s) Name Filter: View only: Latest version Enabled Debug Display by: 10

Select All	Name	Version	Ruleset Path	Creation Date
✓	VehicleAgeRule	1.0	/VehicleAgeRuleApp/1.0/VehicleAgeRule/1.0	Apr 9, 2010 11:03:37 AM GMT

Ruleset 1 - 1 of 1
prev 10 next 10
Remove
© Copyright IBM Corp. 1987, 2009

Figure 11-7 Individual RuleApp view

4. Select the rule set name to view the individual rule set, as shown in Figure 11-8.


WebSphere. Rule Execution Server Console
resAdmin Sign Out IBM.
Home Explorer Decision Warehouse Diagnostics Server Info
About | Print View | Help
Explorer > RuleApps > RuleApp > Ruleset
Navigator
RuleApps (3)
Decision Services
Ruleset View
Help
Test Ruleset View Statistics View Execution Units Upload Ruleset Archive Add Property Edit
Permanent link
Display by: 10
Show Properties (0)
Show HTDS WSDL Options
Show Archive Content
© Copyright IBM Corp. 1987, 2009

/VehicleAgeRuleApp/1.0/VehicleAgeRule/1.0

Name	VehicleAgeRule
Version	1.0
Creation Date	Apr 9, 2010 11:03:37 AM GMT
Display Name	VehicleAgeRule
Description	
Status	✓ enabled
Debug	disabled
WSDL	Get HTDS WSDL for this ruleset version Get HTDS WSDL for the latest ruleset version

Ruleset Parameters

Direction	Name	Kind	XOM Type
	vehicle	xml	com.ilog.insdemo.Vehicle
	partner	native	java.lang.String

Ruleset Parameters 1 - 2 of 2 prev 10 next 10

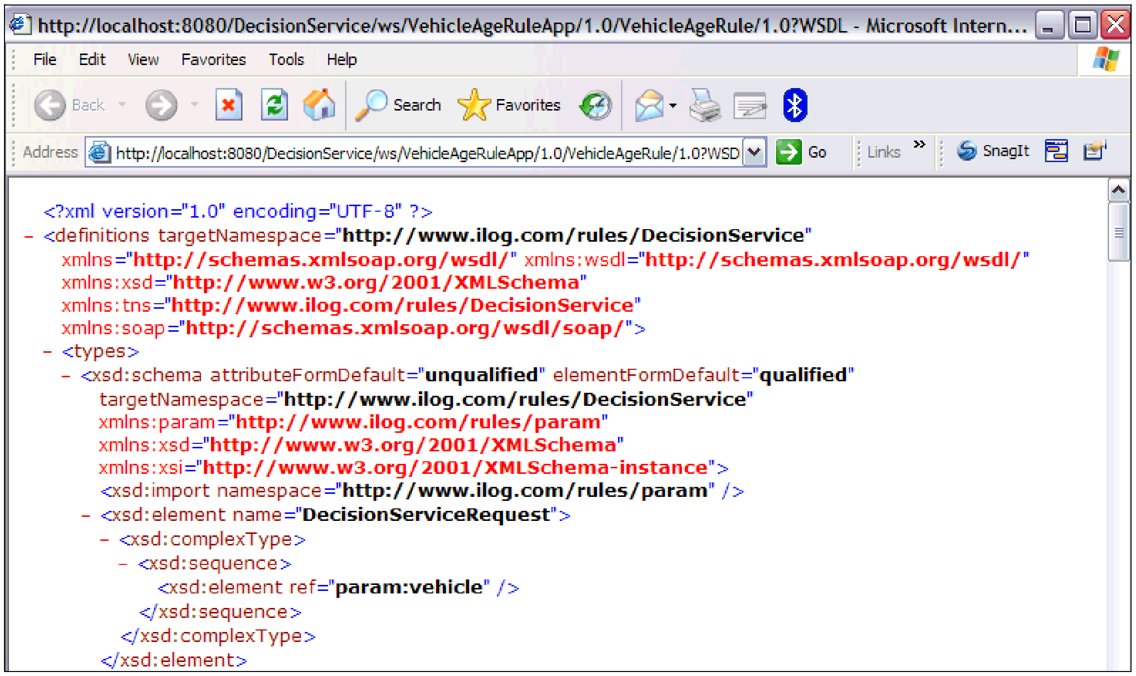
Figure 11-8 Individual rule set view

5. The rule set parameters, kind, and data type are listed. Also listed are two links to obtain the WSDL for the relevant web service (hosted transparent decision service):

   – The first link allows the WSDL for the *selected version* of the rule set to be obtained.

   – The second obtains the WSDL for the current *latest version.*

   Because we have only one version of the rule set in this example, click the first link. The WSDL document opens in a new browser window, as shown in Figure 11-9 on page 296.


http://localhost:8080/DecisionService/ws/VehicleAgeRuleApp/1.0/VehicleAgeRule/1.0?WSDL - Microsoft Intern...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8080/DecisionService/ws/VehicleAgeRuleApp/1.0/VehicleAgeRule/1.0?WSD Go Links SnagIt

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions targetNamespace="http://www.ilog.com/rules/DecisionService"
    xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://www.ilog.com/rules/DecisionService"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  - <types>
    - <xsd:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://www.ilog.com/rules/DecisionService"
        xmlns:param="http://www.ilog.com/rules/param"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <xsd:import namespace="http://www.ilog.com/rules/param" />
      - <xsd:element name="DecisionServiceRequest">
        - <xsd:complexType>
          - <xsd:sequence>
              <xsd:element ref="param:vehicle" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
```

Figure 11-9 Web services (hosted transparent decision services) WSDL document

6. To save this WSDL document to import it into the WebSphere Message Broker toolkit later, click **File** → **Save As** and specify a target file name, for example, `VehicleAgeRule.wsdl`.

11.3.5 Develop WebSphere Message Broker resources

This section describes the steps required to develop the WebSphere Message Broker resources.

Perform *either* of the following steps:

- Develop the message broker resources by using the instructions in this section.
- Use the predeveloped message broker resources in the download, by following the instructions in 11.4.3, “Use predeveloped WebSphere Message Broker resources” on page 318,

Message flow development is carried out in the WebSphere Message Broker Toolkit, which is used to perform the following tasks:

- Create a message set for the insurance quote message by importing the schema for it.
- Create a message set for the web service (hosted transparent decision service) by importing the WSDL which defines its interface.
- Develop the message flow itself.

Create a message set for the insurance quote message

Perform the following steps:

1. Ensure that the WebSphere Message Broker Toolkit 7.0 is started.
2. Although you can use a variety of techniques for creating projects and building broker artifacts, in our example, we start from the Quick Starts wizard in the Projects panel, as shown in Figure 11-10.

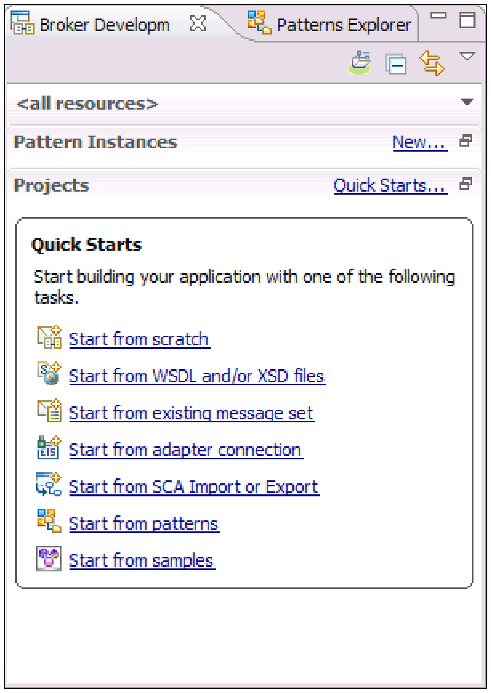
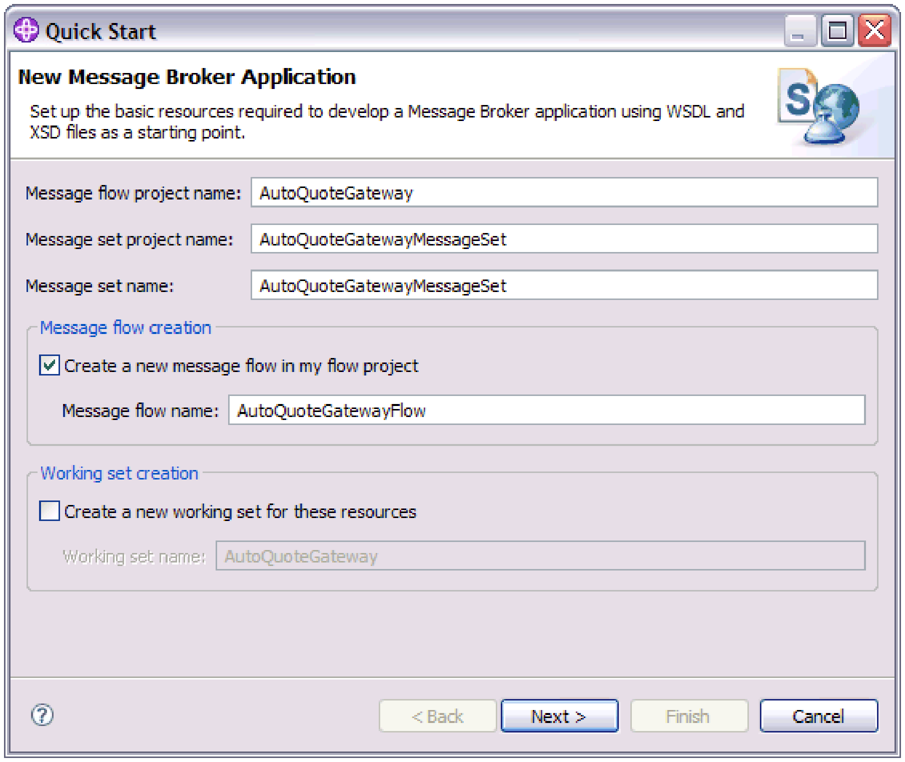

Broker Developm
Patterns Explorer
<all resources>
Pattern Instances
New...
Projects
Quick Starts...
Quick Starts
Start building your application with one of the following tasks.
Start from scratch
Start from WSDL and/or XSD files
Start from existing message set
Start from adapter connection
Start from SCA Import or Export
Start from patterns
Start from samples

Figure 11-10 The Quick Starts wizard launcher

3. Click **Start with WSDL and/or XSD files**.

   The Quick Start window opens (Figure 11-11).



New Message Broker Application

Set up the basic resources required to develop a Message Broker application using WSDL and XSD files as a starting point.

Message flow project name: AutoQuoteGateway

Message set project name: AutoQuoteGatewayMessageSet

Message set name: AutoQuoteGatewayMessageSet

Message flow creation

☑ Create a new message flow in my flow project

Message flow name: AutoQuoteGatewayFlow

Working set creation

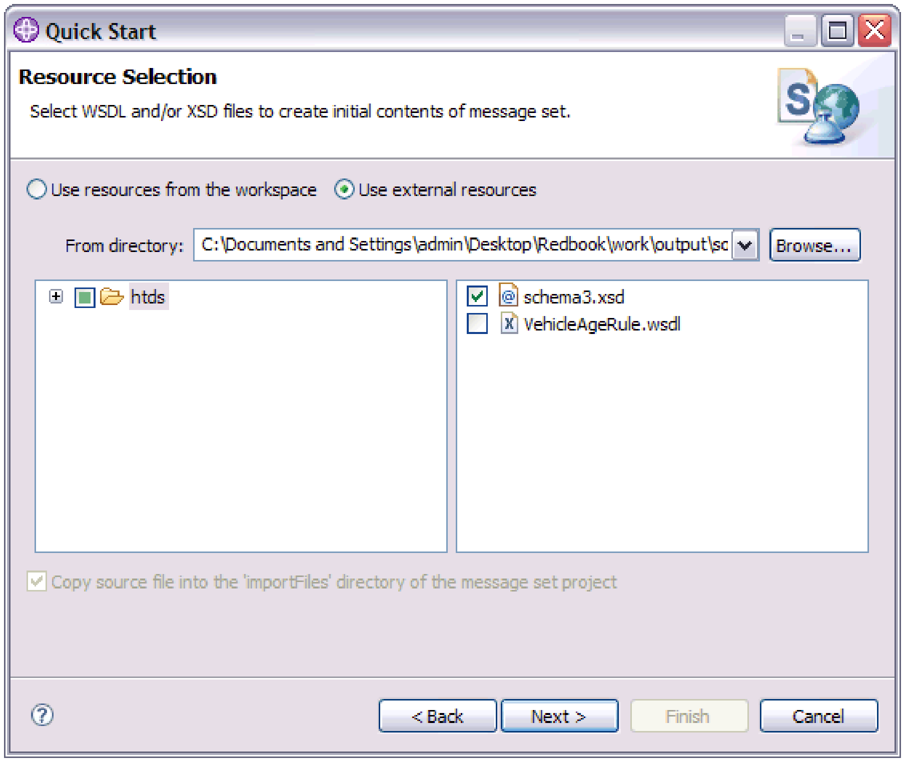
☐ Create a new working set for these resources

Working set name: AutoQuoteGateway

Figure 11-11 The Quick Start: New Application dialog

4. Specify the project name `AutoQuoteGateway`. We do not want to create a working set at this time, so clear the check box. Click **Next**.

   The Quick Start, Resource Selection panel opens (Figure 11-12).



Quick Start

Resource Selection

Select WSDL and/or XSD files to create initial contents of message set.

( ) Use resources from the workspace (●) Use external resources

From directory: C:\Documents and Settings\admin\Desktop\Redbook\work\output\sc

htds

[✓] schema3.xsd
[ ] VehicleAgeRule.wsdl

[✓] Copy source file into the 'importFiles' directory of the message set project

Figure 11-12 Selecting resources to import

5. Select **Use external resources**, and locate the schema for the quote request. Select the check box next to the schema name and then click **Next**.

   The Quick Start, Message Selection panel opens (Figure 11-13).

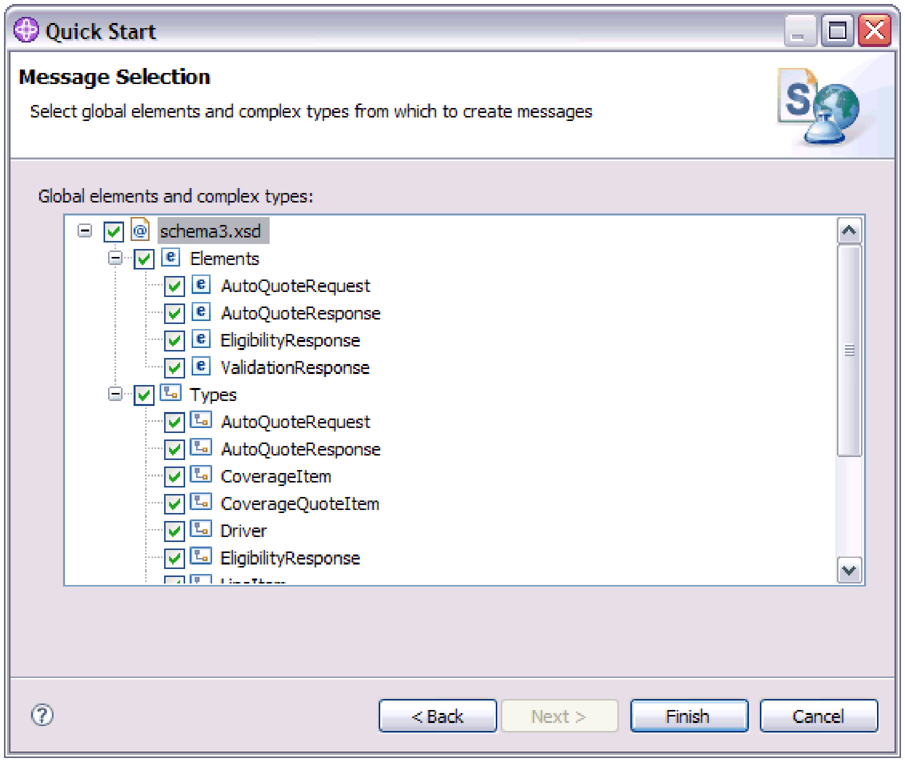


Figure 11-13 Message selection dialog

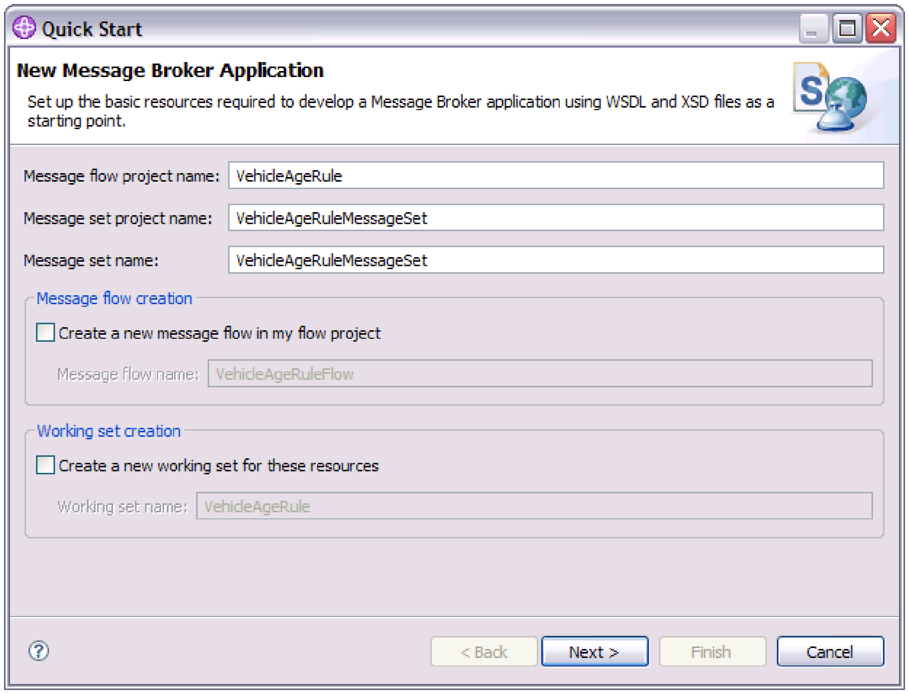
6. Select the elements and types to import. For this example, we selected all of them. Click **Finish**.

You have now created the message set and an empty message flow. We will come back to this later.

Create a message set for the web service

Perform the following steps:

1. Import the WSDL for the web service (hosted transparent decision service) by returning to the Quick Start wizard and clicking **Start with WSDL and/or XSD files** again. The next window opens (Figure 11-14).



Set up the basic resources required to develop a Message Broker application using WSDL and XSD files as a starting point.

Message flow project name: VehicleAgeRule

Message set project name: VehicleAgeRuleMessageSet

Message set name: VehicleAgeRuleMessageSet

Message flow creation

☐ Create a new message flow in my flow project

Message flow name: VehicleAgeRuleFlow

Working set creation

☐ Create a new working set for these resources

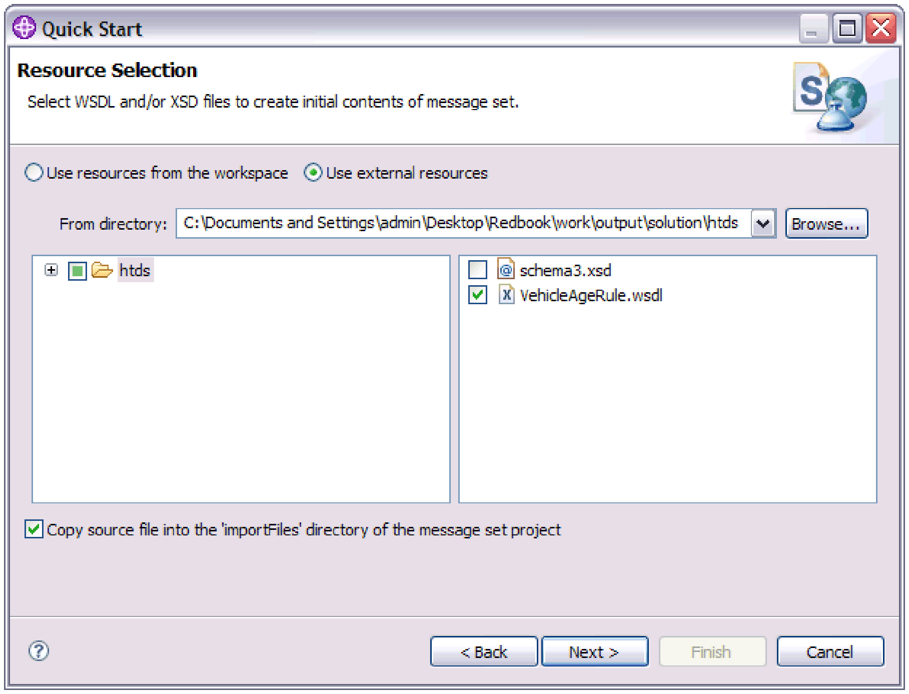
Working set name: VehicleAgeRule

Figure 11-14 The Quick Start: New Application dialog

2. Specify the message flow project name `VehicleAgeRule`. Because we are simply importing WSDL and will use this in the message flow that we previously created, we do not want to create a new one now, so clear both check boxes, as shown.

3. Click **Next.**

   The Quick Start, Resource Selection panel opens (Figure 11-15).


Quick Start

Resource Selection

Select WSDL and/or XSD files to create initial contents of message set.

( ) Use resources from the workspace (●) Use external resources

From directory: C:\Documents and Settings\admin\Desktop\Redbook\work\output\solution\htds

htds

☐ schema3.xsd

☑ VehicleAgeRule.wsdl

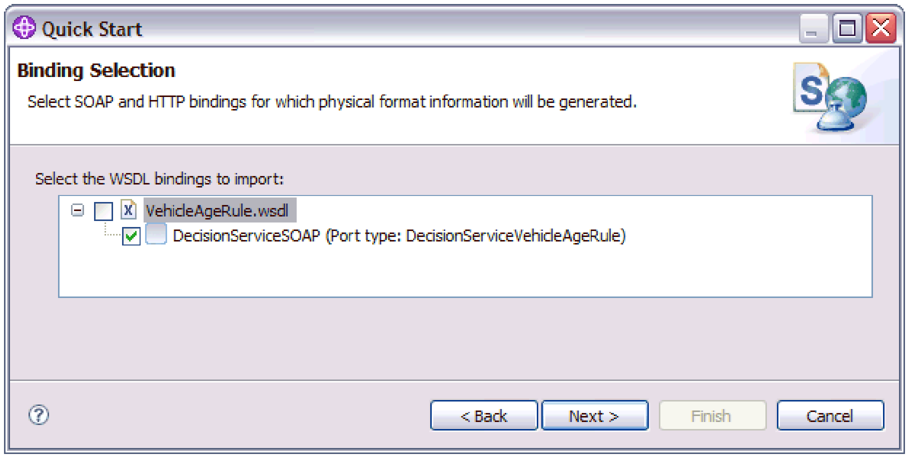
☑ Copy source file into the 'importFiles' directory of the message set project

Figure 11-15 Selecting resources to import

4. Select **Use external resources**, and locate the WSDL for the web service (hosted transparent decision service).

5. Select the check box next to the WSDL file name and click **Next**.

   The Quick Start, Binding Selection panel opens (Figure 11-16).



Quick Start

Binding Selection

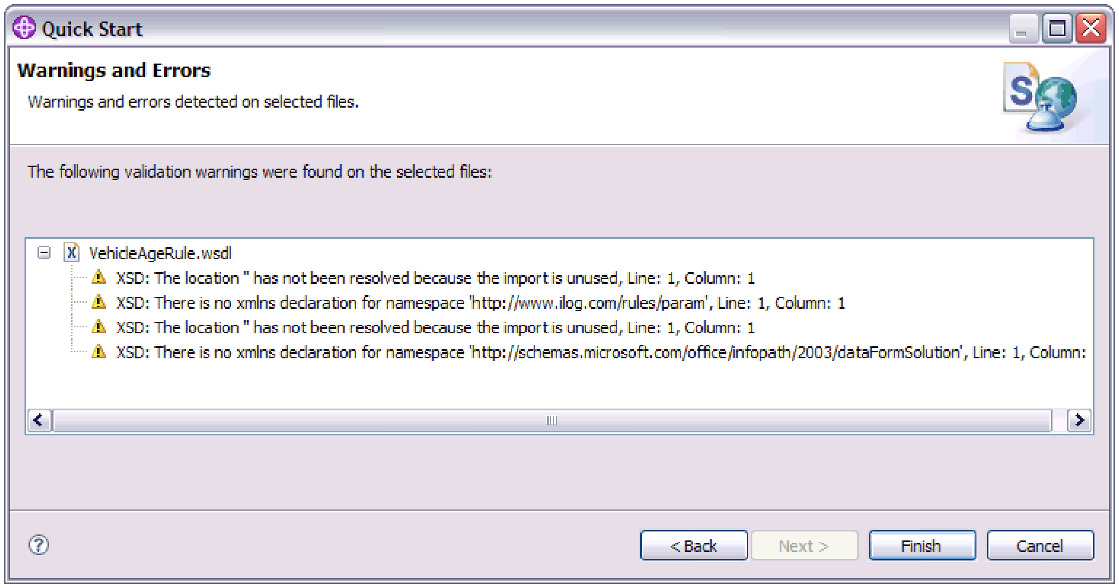
Select SOAP and HTTP bindings for which physical format information will be generated.

Select the WSDL bindings to import:

- [ ] VehicleAgeRule.wsdl
  - [✓] DecisionServiceSOAP (Port type: DecisionServiceVehicleAgeRule)

Figure 11-16 Binding selection dialogue

6. Select the binding and click **Next**. Warnings such as those in Figure 11-17 might be listed. You can ignore them.



Quick Start

Warnings and Errors

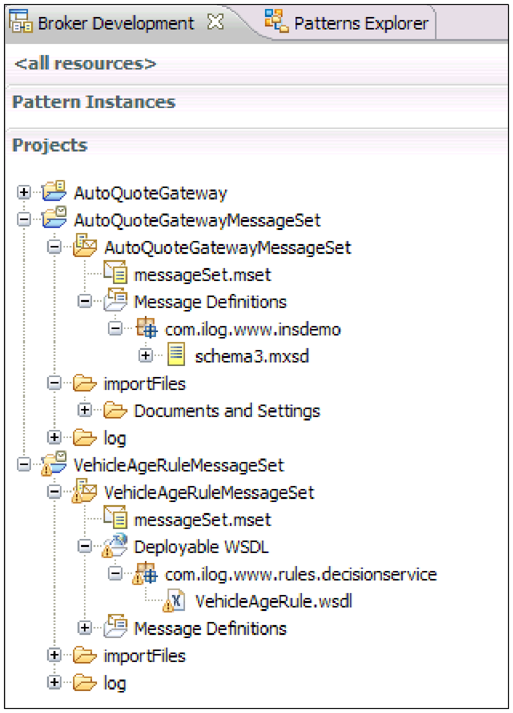
Warnings and errors detected on selected files.

The following validation warnings were found on the selected files:

VehicleAgeRule.wsdl
- ⚠ XSD: The location '' has not been resolved because the import is unused, Line: 1, Column: 1
- ⚠ XSD: There is no xmlns declaration for namespace 'http://www.ilog.com/rules/param', Line: 1, Column: 1
- ⚠ XSD: The location '' has not been resolved because the import is unused, Line: 1, Column: 1
- ⚠ XSD: There is no xmlns declaration for namespace 'http://schemas.microsoft.com/office/infopath/2003/dataFormSolution', Line: 1, Column:

Figure 11-17 Warnings during import

7. Click **Finish** to complete the import. The project view opens (Figure 11-18).



```
AutoQuoteGateway
AutoQuoteGatewayMessageSet
    AutoQuoteGatewayMessageSet
        messageSet.mset
        Message Definitions
            com.ilog.www.insdemo
                schema3.mxsd
    importFiles
        Documents and Settings
    log
VehicleAgeRuleMessageSet
    VehicleAgeRuleMessageSet
        messageSet.mset
        Deployable WSDL
            com.ilog.www.rules.decisionservice
                VehicleAgeRule.wsdl
        Message Definitions
    importFiles
    log
```

Figure 11-18 Example of projects view

Develop the message flow

As mentioned previously, we do not focus on all the development activities to build the solution, because we assume that you are familiar with message flow development. Instead, we draw your attention to the significant and relevant parts of the solution. The final request processing flow is shown in Figure 11-19.

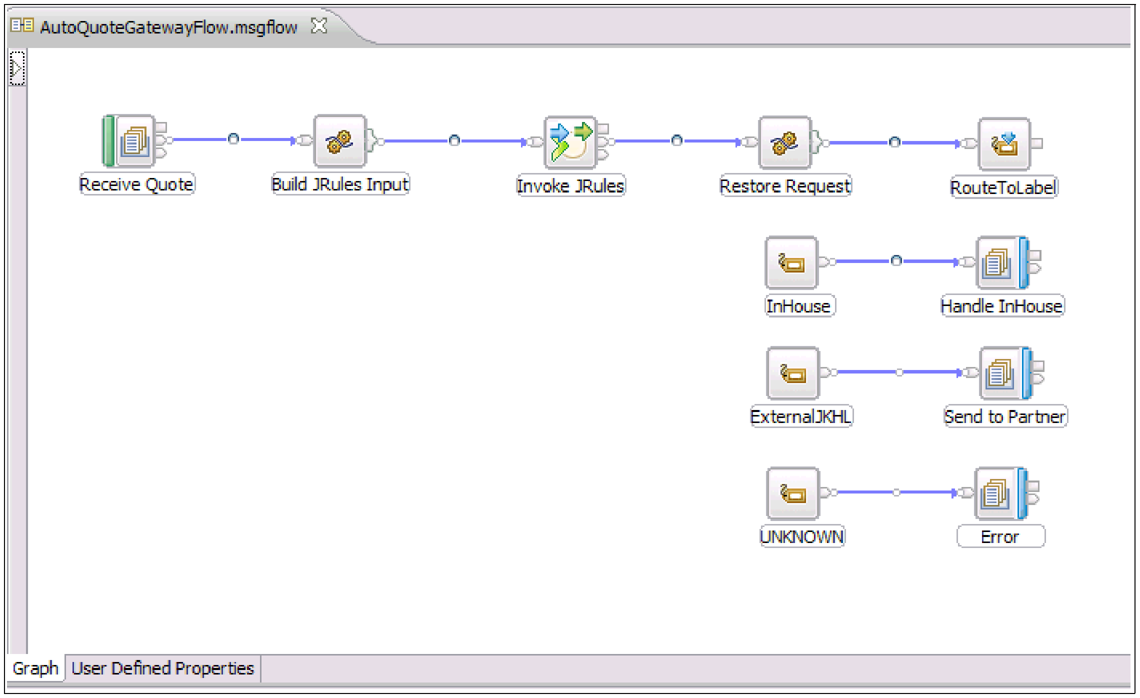


Figure 11-19 Our message flow solution

The flow processing (Figure 11-19) for the solution is as follows:

- The inbound quote request is sent to the WebSphere MQ queue which is read by the MQ input node *Receive Quote*.
- The compute node *Build JRules Input* creates a SOAP request message to execute the rule as a web service (hosted transparent decision service). We use message parts from the original input message to form that request. Because the formats of the input message and the rule input differ, we must temporarily save the input request in the broker local environment.
- The invocation of the web service (hosted transparent decision service) rule is performed in the *Invoke JRules* node.
- The compute node *Restore Request* restores the input request, and determines and sets the routing information, based on the output from the rule.

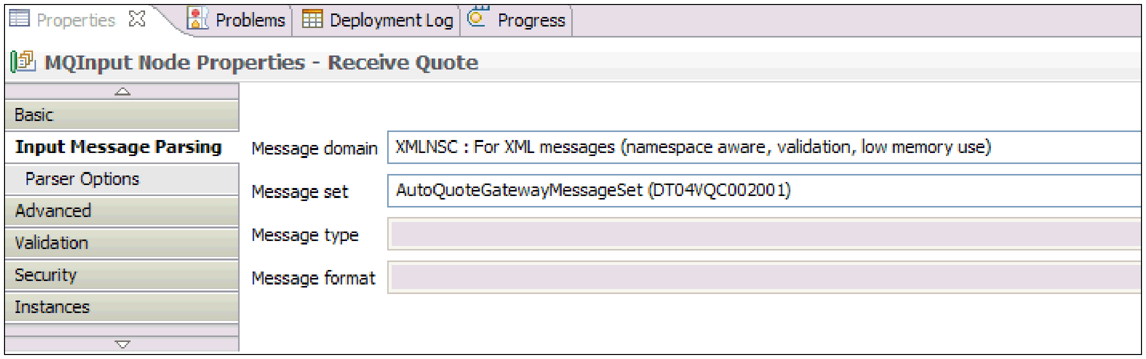
- The *RouteToLabel* node routes the request to the label node, specified in the previous node.
- The existing back office application and each valid partner is represented by an associated, and known, label node to which the message is routed and then processed accordingly. In our example:
  - Quote requests to be handled within IBM Redbooks Company Insurance are routed to the *InHouse* node and then sent to the input queue for the existing quote application.
  - Quote requests for partners are routed to the *ExternalJKHL* label node and then routed through MQ queues to the partner.
  - Any unexpected return value from the rule is handled by the *UNKNOWN* label node.

Notes:

- To focus on the solution elements, our example does not include the necessary robust error handling recommended for production deployments.
- Response flow processing is not examined or considered because this is a standard broker pattern and is unaffected by the JRules integration.

In the remainder of this section, we look at specific parts of the solution that are worthy of note or examination.

In Figure 11-20 on page 306, we show how the Receive Quote MQ input node is configured. Because the input message is described by an XML schema, we configure the node to use the XMLNSC parser and specify the name of the message set that we created earlier.

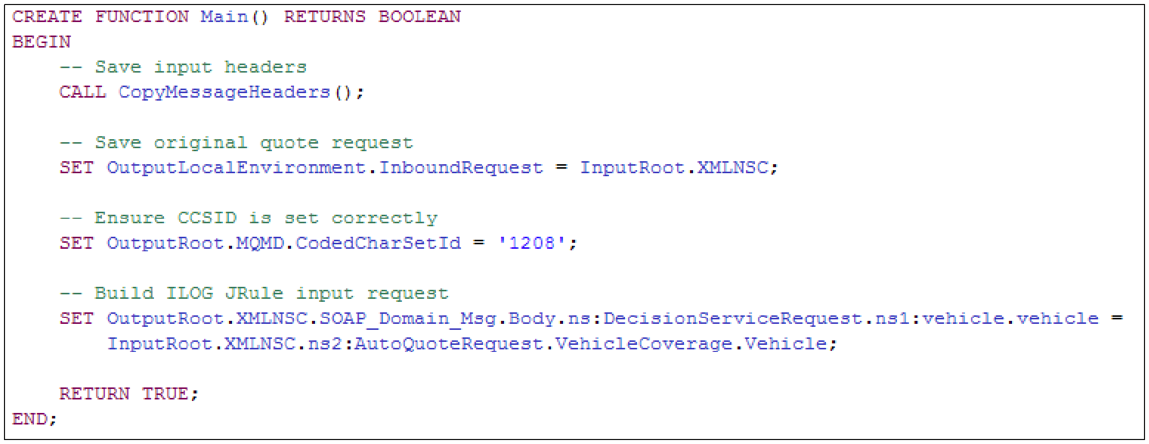


Message domain	XMLNSC : For XML messages (namespace aware, validation, low memory use)
Message set	AutoQuoteGatewayMessageSet (DT04VQC002001)
Message type	
Message format	

Figure 11-20 Message parsing properties on MQ input node

Figure 11-21 shows the implementation of the compute node *Build JRules Input*:

- Copies any message headers, which is standard practice.
- Copies the entire input message tree to the LocalEnvironment.
- Forces the CCSID to be 1208.
- Constructs the SOAP request message, required by the web service (hosted transparent decision service), from the input message. The XMLNSC parser is used to build the SOAP message.

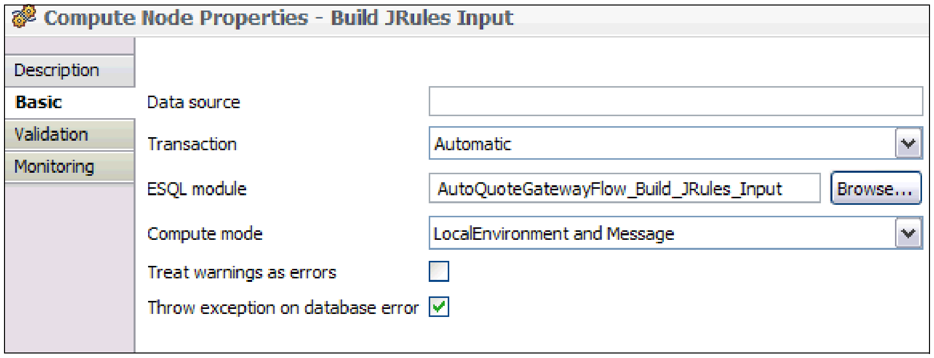


```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- Save input headers
    CALL CopyMessageHeaders();

    -- Save original quote request
    SET OutputLocalEnvironment.InboundRequest = InputRoot.XMLNSC;

    -- Ensure CCSID is set correctly
    SET OutputRoot.MQMD.CodedCharSetId = '1208';

    -- Build ILOG JRule input request
    SET OutputRoot.XMLNSC.SOAP_Domain_Msg.Body.ns:DecisionServiceRequest.ns1:vehicle.vehicle =
        InputRoot.XMLNSC.ns2:AutoQuoteRequest.VehicleCoverage.Vehicle;

    RETURN TRUE;
END;
```

Figure 11-21 Building the rule input from the quote request

Important: Setting the CCSID is necessary in our solution to force the required code page conversions because the web service consumer (the broker) and the provider (RES and HTDS) are deployed in separate runtime environments, which are configured with separate locales and code sets. This step might be necessary depending on the way your own environment is configured.

In Figure 11-22, we show that to cause the (updated) local environment to be propagated to “downstream” processing nodes, the *Compute mode* must be set to `LocalEnvironment and Message`.



Data source

Transaction: Automatic

ESQL module: AutoQuoteGatewayFlow_Build_JRules_Input

Compute mode: LocalEnvironment and Message

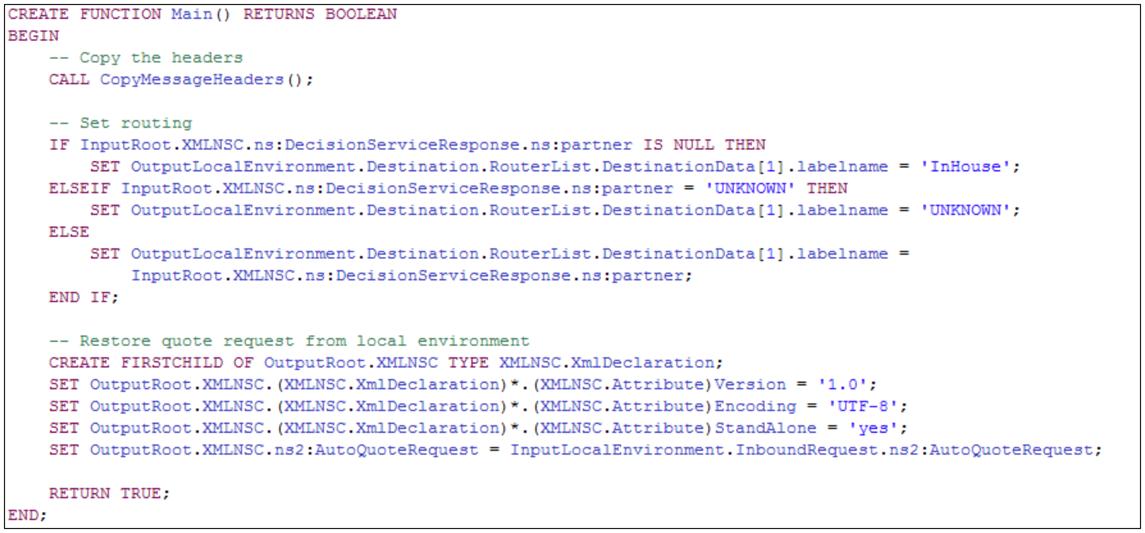
Treat warnings as errors: ☐

Throw exception on database error: ✓

Figure 11-22 Compute node properties

Figure 11-23 shows the implementation of the compute node *Restore Request* which performs the following functions:

- Copies any message headers which is standard practice.
- Sets the routing information for the subsequent Route To Label node in the local environment of the appropriate folders.
- Restores the input message from the local environment, which was previously stored there.



```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- Copy the headers
    CALL CopyMessageHeaders();

    -- Set routing
    IF InputRoot.XMLNSC.ns:DecisionServiceResponse.ns:partner IS NULL THEN
        SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname = 'InHouse';
    ELSEIF InputRoot.XMLNSC.ns:DecisionServiceResponse.ns:partner = 'UNKNOWN' THEN
        SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname = 'UNKNOWN';
    ELSE
        SET OutputLocalEnvironment.Destination.RouterList.DestinationData[1].labelname =
            InputRoot.XMLNSC.ns:DecisionServiceResponse.ns:partner;
    END IF;

    -- Restore quote request from local environment
    CREATE FIRSTCHILD OF OutputRoot.XMLNSC TYPE XMLNSC.XmlDeclaration;
    SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Version = '1.0';
    SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)Encoding = 'UTF-8';
    SET OutputRoot.XMLNSC.(XMLNSC.XmlDeclaration)*.(XMLNSC.Attribute)StandAlone = 'yes';
    SET OutputRoot.XMLNSC.ns2:AutoQuoteRequest = InputLocalEnvironment.InboundRequest.ns2:AutoQuoteRequest;

    RETURN TRUE;
END;
```

Figure 11-23 Compute node routing processing

11.3.6 Deploy the WebSphere Message Broker message flows

Perform the following steps:

1. In the WebSphere Message Broker Toolkit, right-click the **AutoQuoteGateway** project and select **New and Message Broker Archive**. Enter `AutoQuoteGateway` in the Name field and click **Finish**.
2. Select the following items:
   - **AutoQuoteGatewayFlow** message flow
   - **AutoQuoteQuoteGatewayMessageSet** message set
   - **VehicleAgeRuleMessageSet** message sets.

   Click **Build broker archive** and click **OK**.
3. Select the brokers pane and ensure that the BRKR broker and default execution group are running.

4. Expand the tree in the project explorer pane to show `AutoQuoteGateway.bar` (Figure 11-24).

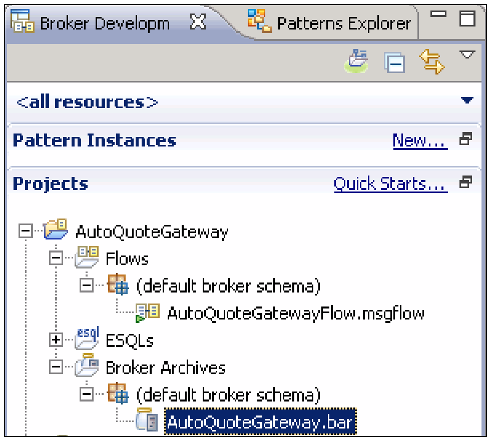


Figure 11-24 Deploy the message flow

5. Click and drag the bar file to the default execution group.

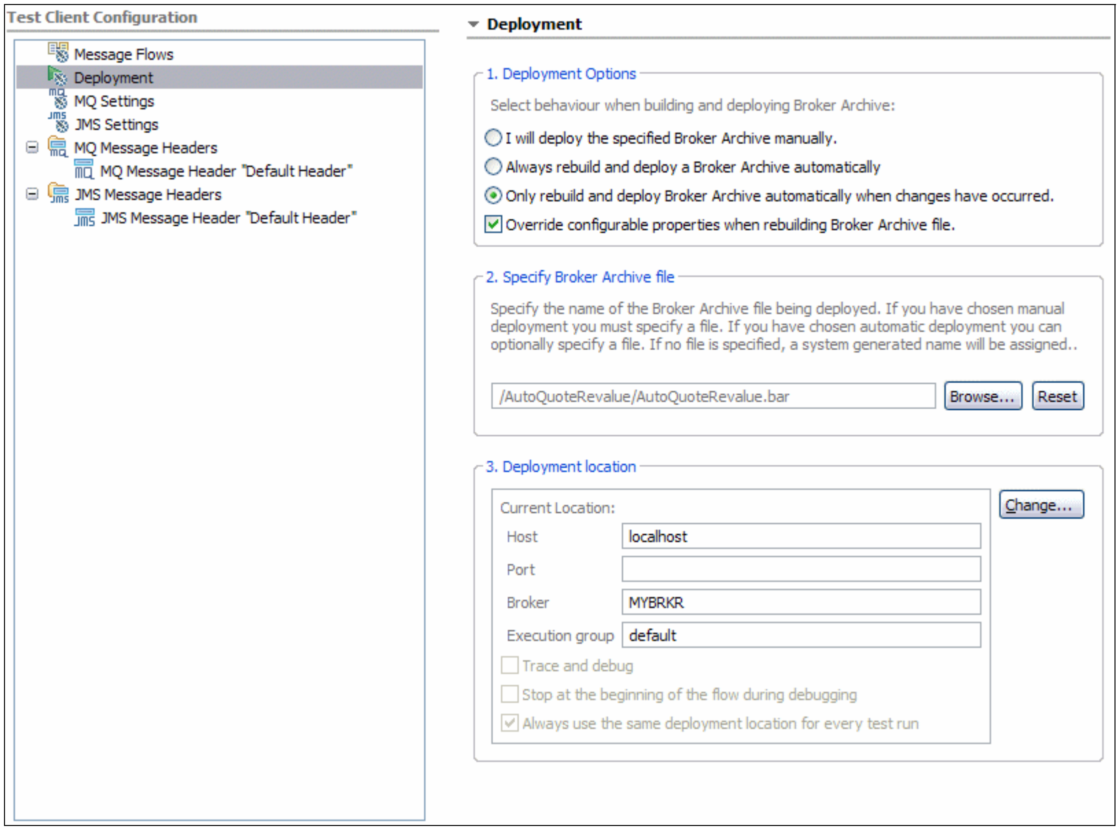
11.3.7 Test the scenario

In this section, you use the WebSphere Message Broker Toolkit test client to test the scenario. The test client is used to put messages to the message flow's MQInput node and get messages from the MQOutput nodes.

To test the scenario, use the `AutoQuoteGatewayRequest.xml` input file that is located in the AutoQuoteGateway project.

Execute the test as follows:

1. In the WebSphere Message Broker Toolkit, right-click the MQInput node of the AutoQuoteGatewayFlow message flow and select **Test**. The Events tab opens in the main pane.
2. Click the **Configuration** tab in the main pane.
3. Click **Deployment** in the Test Client Configuration (Figure 11-25 on page 311).
4. In section 2, specify the `AutoQuoteGateway.bar` broker archive file, which contains the request and response flows.
5. In section 3, click **Change** and select the execution group for the test.


Test Client Configuration

- Message Flows
- Deployment
- MQ Settings
- JMS Settings
- MQ Message Headers
  - MQ Message Header "Default Header"
- JMS Message Headers
  - JMS Message Header "Default Header"

Deployment

1. Deployment Options

Select behaviour when building and deploying Broker Archive:

- ○ I will deploy the specified Broker Archive manually.
- ○ Always rebuild and deploy a Broker Archive automatically
- ● Only rebuild and deploy Broker Archive automatically when changes have occurred.
- ☑ Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/AutoQuoteRevalue/AutoQuoteRevalue.bar

3. Deployment location

Current Location:

Host	localhost
Port	
Broker	MYBRKR
Execution group	default

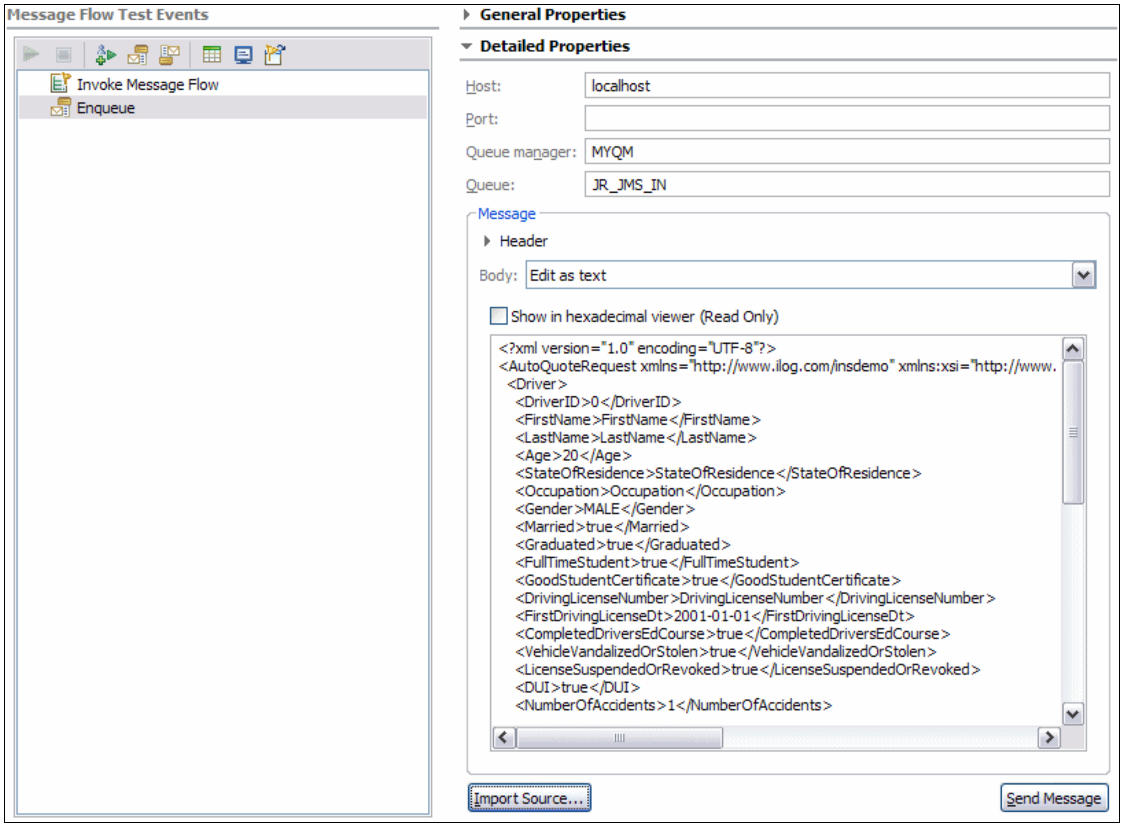
- ☐ Trace and debug
- ☐ Stop at the beginning of the flow during debugging
- ☑ Always use the same deployment location for every test run

Figure 11-25 Test configuration

6. Save the configuration using Ctrl+S.
7. Click the **Events** tab in the main pane.
8. Click **Enqueue**.
9. In the detailed Properties section, set the following properties (Figure 11-26 on page 312):

Host:	`localhost`
Port:	(Leave this field blank)
Queue manager:	MYQM
Queue:	AUTOQUOTE.IN

10. To set the payload, click **Import Source.**
11. Navigate to the AutoQuoteGateway project directory, select the `AutoQuoteGatewayRequest.xml` input file, and click **Open**.



Host: localhost
Port:
Queue manager: MYQM
Queue: JR_JMS_IN

Body: Edit as text

Show in hexadecimal viewer (Read Only)

```
<?xml version="1.0" encoding="UTF-8"?>
<AutoQuoteRequest xmlns="http://www.ilog.com/insdemo" xmlns:xsi="http://www.
 <Driver>
  <DriverID>0</DriverID>
  <FirstName>FirstName</FirstName>
  <LastName>LastName</LastName>
  <Age>20</Age>
  <StateOfResidence>StateOfResidence</StateOfResidence>
  <Occupation>Occupation</Occupation>
  <Gender>MALE</Gender>
  <Married>true</Married>
  <Graduated>true</Graduated>
  <FullTimeStudent>true</FullTimeStudent>
  <GoodStudentCertificate>true</GoodStudentCertificate>
  <DrivingLicenseNumber>DrivingLicenseNumber</DrivingLicenseNumber>
  <FirstDrivingLicenseDt>2001-01-01</FirstDrivingLicenseDt>
  <CompletedDriversEdCourse>true</CompletedDriversEdCourse>
  <VehicleVandalizedOrStolen>true</VehicleVandalizedOrStolen>
  <LicenseSuspendedOrRevoked>true</LicenseSuspendedOrRevoked>
  <DUI>true</DUI>
  <NumberOfAccidents>1</NumberOfAccidents>
```

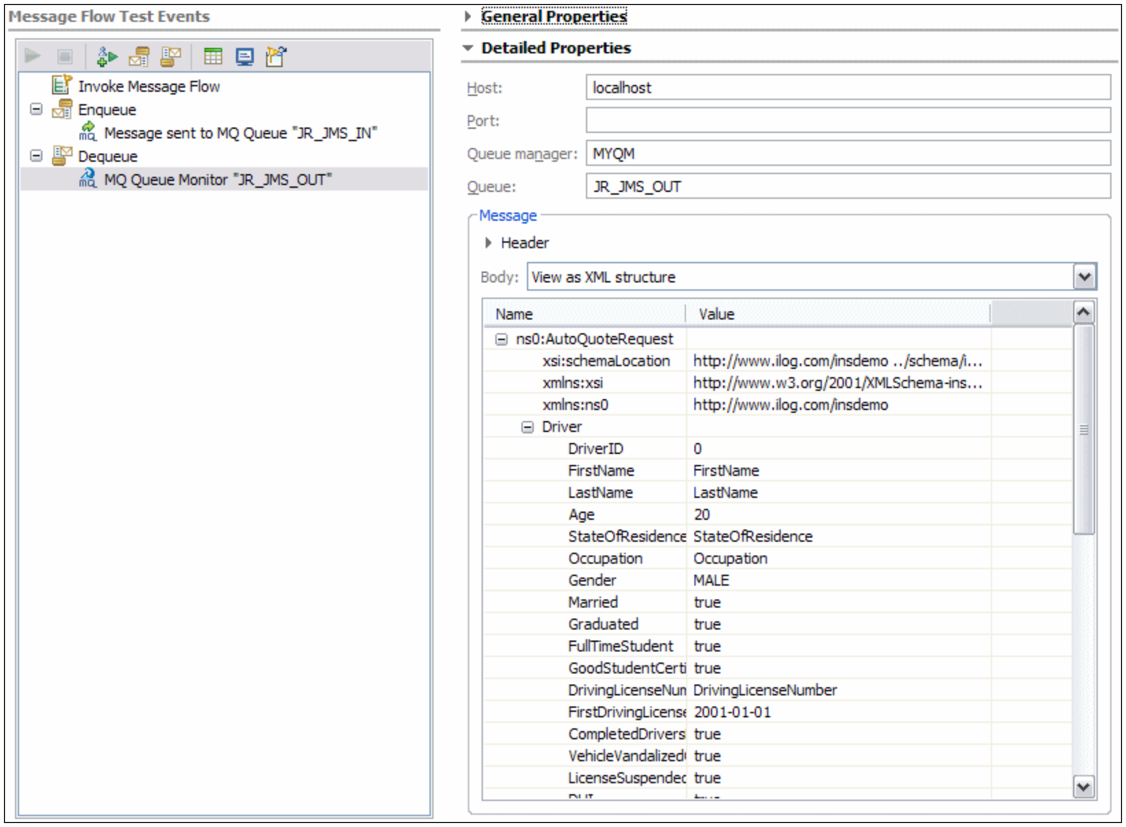
Figure 11-26 Test properties

12. Click **Send Message** to put the message on the queue. The result is that the message flow AutoQuoteGatewayFlow processes the message and invokes the JRules rule. The message flow use the output of the rule to route the message to either the `AUTOQUOTE.OUT` or `AUTOQUOTE.partner` queue, depending on the age of the vehicle. Unexpected data in the rule output can cause the message to be written to the `AUTOQUOTE.UNKNOWN` queue.
13. Use the test client to get the response message.
14. Click **Dequeue**.

15. In the detailed Properties section, set the following properties (Figure 11-27):

Host:	`localhost`
Port:	(Leave this field blank)
Queue manager:	MYQM
Queue:	AUTOQUOTE.OUT or AUTOQUOTE.PARTNER depending on the age of the vehicle specified in the inbound quote request.

16. Click **Get Message**; the response message is displayed in the Message section.



Message Flow Test Events

- Invoke Message Flow
- Enqueue
  - Message sent to MQ Queue "JR_JMS_IN"
- Dequeue
  - MQ Queue Monitor "JR_JMS_OUT"

General Properties

Detailed Properties

Host: localhost

Port:

Queue manager: MYQM

Queue: JR_JMS_OUT

Message

Header

Body: View as XML structure

Name	Value
ns0:AutoQuoteRequest	
xsi:schemaLocation	http://www.ilog.com/insdemo ../schema/i...
xmlns:xsi	http://www.w3.org/2001/XMLSchema-ins...
xmlns:ns0	http://www.ilog.com/insdemo
Driver	
DriverID	0
FirstName	FirstName
LastName	LastName
Age	20
StateOfResidence	StateOfResidence
Occupation	Occupation
Gender	MALE
Married	true
Graduated	true
FullTimeStudent	true
GoodStudentCerti	true
DrivingLicenseNun	DrivingLicenseNumber
FirstDrivingLicens	2001-01-01
CompletedDrivers	true
VehicleVandalized	true
LicenseSuspendec	true

Figure 11-27 Test response message

The supplied XML input file sets the registered year of the quoted vehicle to 2010, so the rule causes the message to be routed to the `AUTOQUOTE.OUT` queue for processing within IBM Redbooks Company Insurance.

17. Modify the XML input file to set the vehicle age so that it is 10 years old.

18. Return to the earlier steps and retry the test. The quote request message should now be routed to the `AUTOQUOTE.PARTNER` queue for processing.

11.4 Scenario 3: JMS Integration

This section describes how to call business rules running in WebSphere ILOG JRules exposed through an MDB from WebSphere Message Broker using JMS.

11.4.1 Introduction

The IBM Redbooks Company Insurance has known for a long time that the business rules, which recalculate the value of cars in auto insurance quotes, are inflexible to change. Business analysts must frequently change the rules in response to new cars appearing on the market, sales and repair market factors, and evolving government legislation. The rules are coded as transformations in WebSphere Message Broker message flows, which means that when business analysts want to make updates to the rules, the IT department must redevelop, retest, and redeploy message flows.

The business analysts have observed that the amount of time to deploy changes to rules is too long, which means that many customers are able to buy auto insurance with incorrect car valuations. This way results in significant costs to business is terms of over-inflated claims.

However, the IT department complains that it is constantly required to update its message flows. As soon as the department tests and deploys the last set of changes that the business demands, a business analyst presents the IT department with new business rule requirements. This inefficiency results a negative impact on the IT department's other projects.

To resolve these problems, IBM Redbooks Company Insurance plans to replace business rules that are coded in WebSphere Message Broker message flows with rules that are deployed in WebSphere ILOG JRules. The existing WebSphere Message Broker message flows will be updated to invoke the WebSphere ILOG JRules rules.

In this scenario, you have collaborated with the WebSphere ILOG JRules developer to agree on an appropriate interface and the rule is already deployed. You must now develop the new message flows to support the requirement.

Overview of products

This JMS integration scenario requires the following products and versions:

- WebSphere ILOG JRules 7.0.3
- One of the supported J2EE application server runtime environments, such as WebSphere Application Server 7.0
- WebSphere Message Broker 7.0
- WebSphere Message Broker Toolkit 7.0
- WebSphere MQ 7.0.1 + APAR IC68305, located at the following address:

  `http://www.ibm.com/support/docview.wss?uid=swg1IC68305`

The scenario used a single machine for the development and testing of the integration. Clearly, this is not representative of a physical topology which would be used in a production deployment, but this does not materially affect the integration use-case or the guidance in this scenario.

The tooling and development are supported by the following products:

- WebSphere ILOG JRules Rule Studio 7.0.3
- WebSphere Message Broker Toolkit 7.0

The runtime environment is supported by the following products:

- A WebSphere Message Broker broker, named BRKR
- A WebSphere MQ queue manager, named BRKR.QM
- One of the J2EE runtime environments supported by ILOG JRules, for example, WebSphere Application Server, which hosts RES with the deployed `jrules-messaging-integration-ruleapp` rule application.

Figure 11-28 shows the runtime and tooling environments in this scenario.

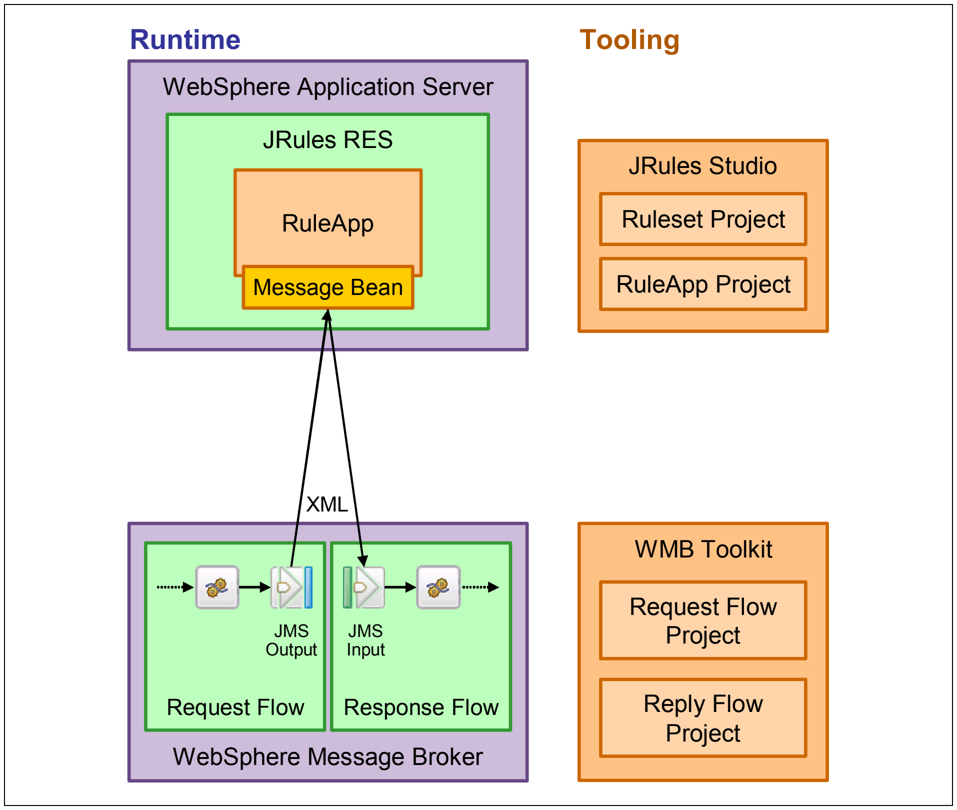


Figure 11-28 Scenario overview

11.4.2 Prepare JRules

This section gives an overview of deploying the RES and JRules MDB to WebSphere Application Server and deploying the rule sets to the RES.

Perform the following steps:

1. Deploy the RES to your chosen runtime environment. For instructions, see "Installing Rule Execution Server on WebSphere Application Server V7.0" (under WebSphere ILOG JRules 7.0.3: Java EE add-ons in the information center):

   `http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m3/topic/com.ibm.websphere.ilog.jrules.install.doc/Content/Business_Rules/Documentation/_pubskel/JRules_Application_Servers/ps_Installing_JRules_IC99.html`

2. Deploy the JRules MDB. For instructions, see “Integrating WebSphere MQ in WebSphere Application Server to support asynchronous execution in JRules” (under WebSphere ILOG JRules 7.0.3: Java EE add-ons information center):

   `http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m3/topic/com.ibm.websphere.ilog.jrules.install.doc/Content/Business_Rules/Documentation/_pubskel/JRules_Application_Servers/ps_Installing_JRules_IC122.html`

3. Start Rules Studio and open the JRulesStudio_Workspace_XML (included in the download).
4. Click the **Project** menu and select **Clean**.
5. Select **Clean All Projects** and click **OK**.
6. Right-click the **jrules-messaging-integration-ruleapp** project in the Rule Explorer, select **RuleApp**, then click **Deploy**.
7. Leave the deployment type as the default, and click **Next**.
8. Click the **Create a temporary Rule Execution Server** configuration and enter the URL, Login, and Password for the RES.
9. Click **Finish** to deploy the rule application.
10. Open the Rule Execution Server console in your web browser (for example, `http://machine:9082/res`).
11. Enter the user name and password of the RES administrator that you configured when deploying the Rule Execution Server and click **Sign in** to open the Rule Execution Server console.
12. Click **Explorer** and check that the `JRulesMessagingIntegrationRuleApp` is listed in the RuleApps.

Your JRules run time is now ready to be used by WebSphere Message Broker. Follow the instructions in *either* of the following sections:

- 11.4.3, “Use predeveloped WebSphere Message Broker resources” on page 318
- 11.4.4, “Develop WebSphere Message Broker resources” on page 318 to develop the message broker resources.

11.4.3 Use predeveloped WebSphere Message Broker resources

You may use predeveloped message broker resources in the download, as described here, or go to 11.4.4, “Develop WebSphere Message Broker resources” on page 318 to develop the message broker resources.

Perform the following steps:

1. Start WebSphere Message Broker Toolkit and open WMB_Workspace_JMS (included in the download).

   The AutoQuoteRevalue and AutoQuoteRevalueJava projects are listed in the Projects panel of your workspace.

2. Click the **Project** menu and select **Clean**.
3. Select **Clean All Projects** and click **OK**.

You can now go to 11.4.6, “Deploy the WebSphere Message Broker message flows” on page 323 to deploy the resources that are required for the solution.

11.4.4 Develop WebSphere Message Broker resources

As stated previously, you may either use the predeveloped message broker resources in the download, using the instructions in 11.4.3, “Use predeveloped WebSphere Message Broker resources” on page 318, or develop the message broker resources using the instructions described in this section.

Overview of the message flow redesign

The scenario involves replacing an original message flow (as shown in Figure 11-29) with two new message flows. This section provides a high-level view of the original and new message flows.

The original message flow contains business logic, which is coded using extended SQL (ESQL) in a compute node, and the new flow contains no business logic. Instead, the new flow uses JMS to call the Rule Execution Server (RES), where business rules reside. The rules in the RES provide equivalent decision function to the original ESQL, but with the added benefit related to the use of BRMS, as discussed in previous chapters.

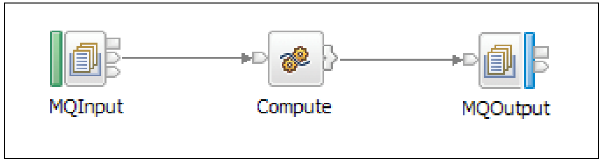


Figure 11-29 Original flow

Although the message flows are redesigned, the original and new flows are identical in the following respects:

- The flows link front-end systems where auto quote request messages are generated (for example, in the web application server) and back-end systems where auto quote request messages are sent for processing (for example, to the transaction processing server).
- The flows use WebSphere MQ for input (from front end systems) and output (to backend systems).
- The messages for input and output are of identical format (for example, XML auto quote request messages).
- The flows are used to route messages from front end systems to back end systems and to adjust the value of the vehicle (in the auto insurance quote request messages) based on information about the vehicle and driver.

Message flow development is carried out in the WebSphere Message Broker Toolkit. The Toolkit is used to develop the request and reply message flows. Creating a message set is not necessary because in this scenario the entire message flow input message is passed to JRules and therefore the broker does not need to understand the detailed format of the message payload.

Develop the new request message flow

As mentioned earlier, we do not focus on all of the development activities to build the solution, because we assume that you are familiar with message flow development. Instead, we draw your attention to the significant and relevant parts of the solution. The new request message flow is shown in Figure 11-30.

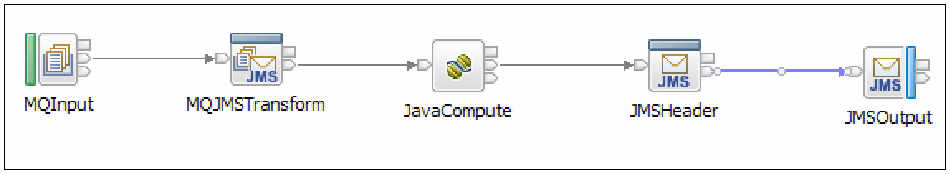


Figure 11-30 New request message flow

The new request message flow will take a WebSphere MQ input message containing an AutoQuoteRequest XML message and pass it as a JMS message to the WebSphere ILOG JRules RES.

The function of each node is as follows.

- MQInput
  - Reads an AutoQuoteRequest XML message from a WebSphere MQ queue.
  - Sets the queue name (for example JR_JMS_IN).
- MQJMSTransform
  - Transforms the WebSphere MQ message into JMS format.
- JavaCompute
  - Converts the AutoQuoteRequest XML message into a JMS ObjectMessage suitable for the RES MDB.
- JMSHeader:
  - Sets JMS header application properties required by the RES MDB to identify which rule application to run.
  - Sets `ilog_rules_bres_mdb_status` to `request`.
  - Sets `ilog_rules_bres_mdb_rulesetPath` to the rule set path, as in the following example:

    ```
    '/JRulesMessagingIntegrationRuleApp/1.0/JRulesEnrichmentIntegrati
    onRules/1.0'
    ```
- JMSOutput
  - Writes the JMS ObjectMessage to the WebSphere MQ queue being read by the RES MDB.
  - Sets the Destination queue (for example JR_JMS_JRULES_REQ).
  - Sets the Reply to destination queue (for example JR_JMS_JRULES_REP).
  - Sets the Location JNDI bindings (for example `file:/C:\JNDI-Directory`).
  - Sets the Connection Factory Name (for example JRulesQcf).
  - Sets the Message Type to ObjectMessage.

Develop the new response message flow

The new response message flow is shown in Figure 11-31.

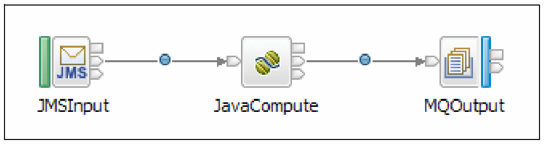


Figure 11-31   New response message flow

The new response message flow takes a JMS input message that contains an AutoQuoteRequest XML message (from the RES, where it was enriched) and passes it as a WebSphere MQ message to a queue. The functions are as follows:

- JMSInput
  - Reads a JMS ObjectMessage from the WebSphere MQ queue written to by the RES.
  - Sets the Source queue (R_JMS_JRULES_REP).
  - Sets the Location JNDI bindings (`file:/C:\JNDI-Directory`).
  - Sets the Connection Factory Name (JRulesQcf).
- JavaCompute
  - Converts the JMS ObjectMessage into an AutoQuoteRequest XML message.
- MQOutput
  - Writes the AutoQuoteRequest XML message to the WebSphere MQ queue.
  - Sets the Queue manager name (BRKR.QM).
  - Sets the Queue name (JR_JMS_OUT).

Now, the AutoQuoteRevalue and AutoQuoteRevalueJava projects should be free from errors (and might include various acceptable warnings).

11.4.5 Create the WebSphere MQ and JNDI resources

This section describes how to deploy the scenario components. We assume that the `BRKR.QM` queue manager is already defined and running.

This section describes how to create the WebSphere MQ queues and JNDI objects, and how to deploy the rule application and flow.

Create the WebSphere MQ queues

Perform the following steps:

1. Open a command prompt and enter the **`runmqsc BRKR.QM`** command to start the WebSphere MQ MQSC administration tool.
2. Enter the following MQ script commands (MQSC):

   ```
   DEF QL(JR_JMS_JRULES_REQ)
   DEF QL(JR_JMS_JRULES_REP)
   DEF QL(JR_JMS_IN)
   DEF QL(JR_JMS_OUT)
   END
   ```

Create the JNDI artefacts

Perform the following steps:

1. Open the `%MQInstallPath%\Java\bin\JMSAdmin.config` file and ensure that the following name-value pairs are defined:
   - `INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory`
   - `PROVIDER_URL=file:/C:/JNDI-Directory`
2. Create a folder named `JNDI-Directory` under `C:\` location.
3. Open a command prompt and run the following command:

   ```
   mq_install_path\Java\bin\JMSAdmin
   ```

4. Enter the following commands:

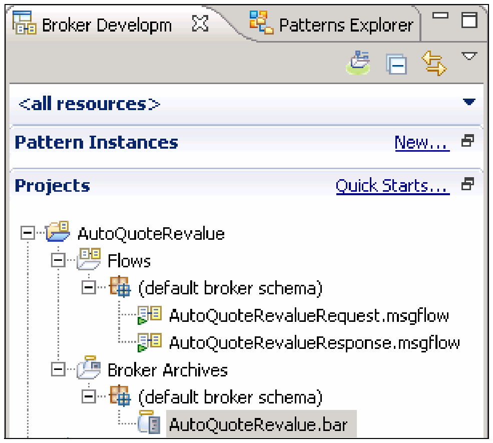
   ```
   DEFINE QCF(JRulesQcf) QMGR(BRKR.QM)
   DEFINE Q(JR_JMS_JRULES_REQ) QMGR(BRKR.QM)
   QUEUE(JR_JMS_JRULES_REQ)
   DEFINE Q(JR_JMS_JRULES_REP) QMGR(BRKR.QM)
   QUEUE(JR_JMS_JRULES_REP)
   END
   ```

The WebSphere MQ and JNDI resource setup is now complete.

11.4.6 Deploy the WebSphere Message Broker message flows

Perform the following steps:

1. In the WebSphere Message Broker Toolkit, right-click the **AutoQuoteRevalue** project and select **New and Message Broker Archive**. Enter `AutoQuoteRevalue` in the Name field and click **Finish**.
2. Select the following items and then click **Build broker archive** and click **OK**:
   - AutoQuoteRevalueRequest message flow
   - AutoQuoteRevalueResponse message flow
   - AutoQuoteRevalueJava
3. Select the brokers pane and ensure that the BRKR broker and default execution group are running.
4. Expand the tree in the project explorer panel to show the `AutoQuoteRevalue.bar`, as shown in Figure 11-32.



AutoQuoteRevalue
Flows
(default broker schema)
AutoQuoteRevalueRequest.msgflow
AutoQuoteRevalueResponse.msgflow
Broker Archives
(default broker schema)
AutoQuoteRevalue.bar

Figure 11-32 Deploy the message flow

5. Drag the bar file to the default execution group.

11.4.7 Test the scenario

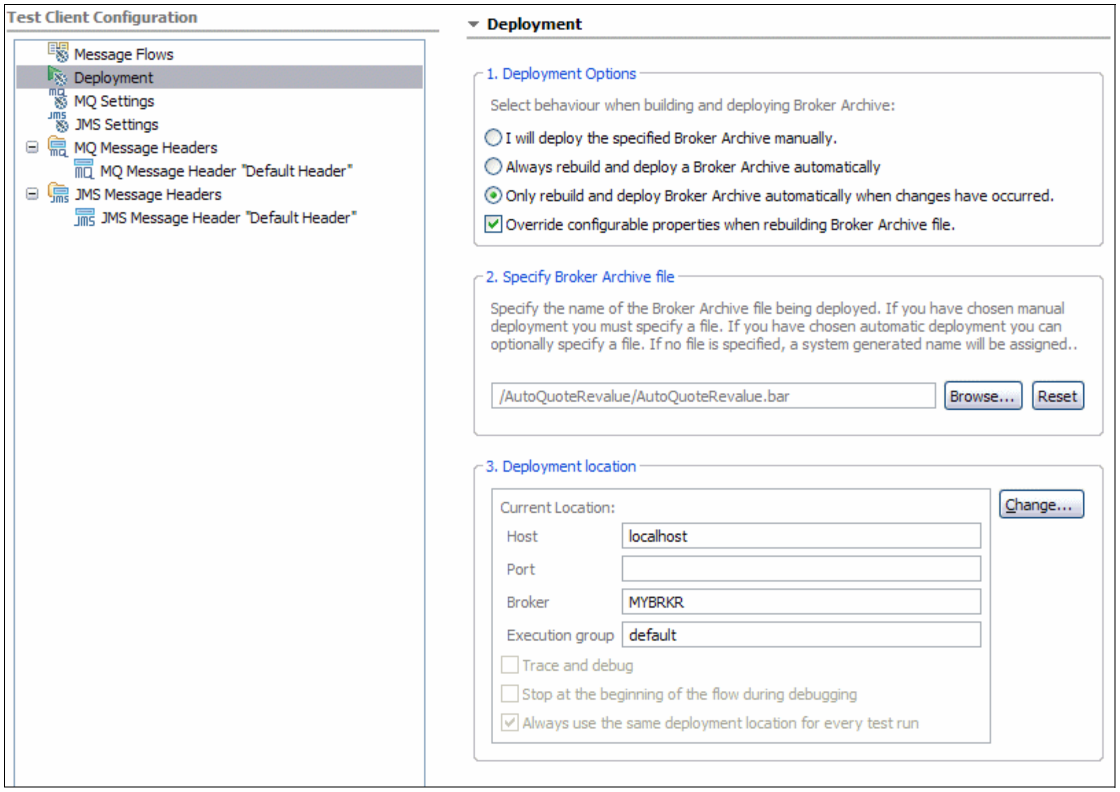
In this section, you use the WebSphere Message Broker Toolkit test client to test the scenario. The test client is used to put messages to the request flow's MQInput node, and get messages from the response flow's MQOutput node.

The following AutoQuoteRequest data, in the XML files in the rule set project (jrules-enrichment-integration-rules) is used as the payload for the test request messages:

- `data/xml/insdemoObjectModel_case_1.xml`
- `data/xml/insdemoObjectModel_case_2.xml`
- `data/xml/insdemoObjectModel_case_3.xml`
- `data/xml/insdemoObjectModel_case_4.xml`

Execute the test as follows:

1. In the WebSphere Message Broker Toolkit, right-click the MQInput node of the AutoQuoteRevalueRequest message flow and select **Test**. The Events tab opens in the main pane.
2. Click the **Configuration** tab in the main pane.
3. Click **Deployment in the Test Client Configuration**.
4. In section 2, specify the `AutoQuoteRevalue.bar` broker archive file, which contains the request and response flows.
5. In section 3, click **Change** and select the execution group for the test, as shown in Figure 11-33 on page 325.


Test Client Configuration

- Message Flows
- Deployment
- MQ Settings
- JMS Settings
- MQ Message Headers
  - MQ Message Header "Default Header"
- JMS Message Headers
  - JMS Message Header "Default Header"

Deployment

1. Deployment Options

Select behaviour when building and deploying Broker Archive:

- ( ) I will deploy the specified Broker Archive manually.
- ( ) Always rebuild and deploy a Broker Archive automatically
- (✓) Only rebuild and deploy Broker Archive automatically when changes have occurred.
- [✓] Override configurable properties when rebuilding Broker Archive file.

2. Specify Broker Archive file

Specify the name of the Broker Archive file being deployed. If you have chosen manual deployment you must specify a file. If you have chosen automatic deployment you can optionally specify a file. If no file is specified, a system generated name will be assigned..

/AutoQuoteRevalue/AutoQuoteRevalue.bar

3. Deployment location

Current Location:

Host	localhost
Port	
Broker	MYBRKR
Execution group	default

- [ ] Trace and debug
- [ ] Stop at the beginning of the flow during debugging
- [✓] Always use the same deployment location for every test run

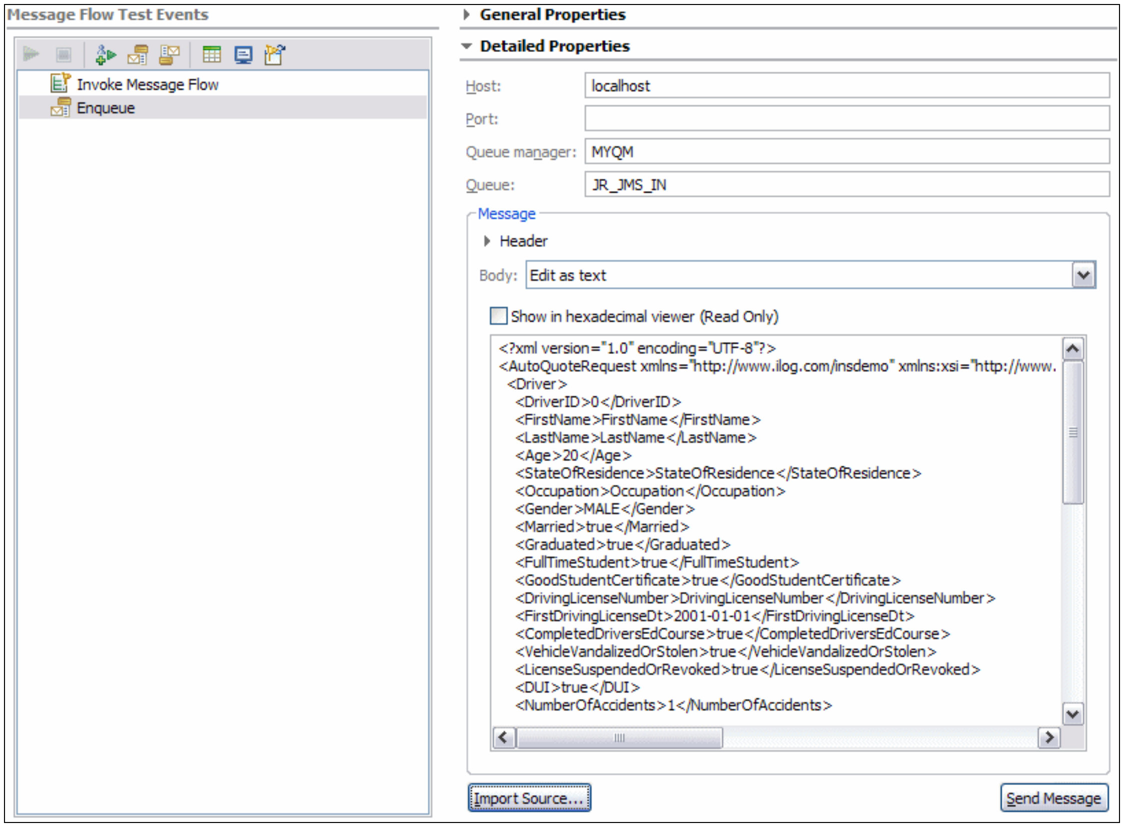
Figure 11-33 Test configuration

6. Save the configuration by pressing Ctrl+S.
7. Click the **Events** tab in the main pane.
8. Click **Enqueue**.
9. In the detailed Properties section, set the following properties (Figure 11-34 on page 326):

Host:	`localhost`
Port:	(Leave this field blank)
Queue manager:	BRKR.QM
Queue:	JR_JMS_IN

10. To set the payload, click **Import Source**.

11. Navigate to the `jrules-routing-integration-rules/data/xml` JRules project directory and select the XML payload file (for example, `insdemoObjectModel_case_1.xml`) and click **Open**. The XML should not be displayed in the Message section, as shown in Figure 11-34.



Host: localhost
Port:
Queue manager: MYQM
Queue: JR_JMS_IN

Message
Header
Body: Edit as text
Show in hexadecimal viewer (Read Only)

```
<?xml version="1.0" encoding="UTF-8"?>
<AutoQuoteRequest xmlns="http://www.ilog.com/insdemo" xmlns:xsi="http://www.
 <Driver>
  <DriverID>0</DriverID>
  <FirstName>FirstName</FirstName>
  <LastName>LastName</LastName>
  <Age>20</Age>
  <StateOfResidence>StateOfResidence</StateOfResidence>
  <Occupation>Occupation</Occupation>
  <Gender>MALE</Gender>
  <Married>true</Married>
  <Graduated>true</Graduated>
  <FullTimeStudent>true</FullTimeStudent>
  <GoodStudentCertificate>true</GoodStudentCertificate>
  <DrivingLicenseNumber>DrivingLicenseNumber</DrivingLicenseNumber>
  <FirstDrivingLicenseDt>2001-01-01</FirstDrivingLicenseDt>
  <CompletedDriversEdCourse>true</CompletedDriversEdCourse>
  <VehicleVandalizedOrStolen>true</VehicleVandalizedOrStolen>
  <LicenseSuspendedOrRevoked>true</LicenseSuspendedOrRevoked>
  <DUI>true</DUI>
  <NumberOfAccidents>1</NumberOfAccidents>
```

Figure 11-34 Test properties

12. Click **Send Message** to put the message on the queue. The result is that the AutoQuoteRevalueRequest flow processes the message and invokes the JRules RES. The RES processes the rules and response. the response is processed by the AutoQuoteRevalueResponse flows. The response message is now on the JR_JMS_OUT queue.

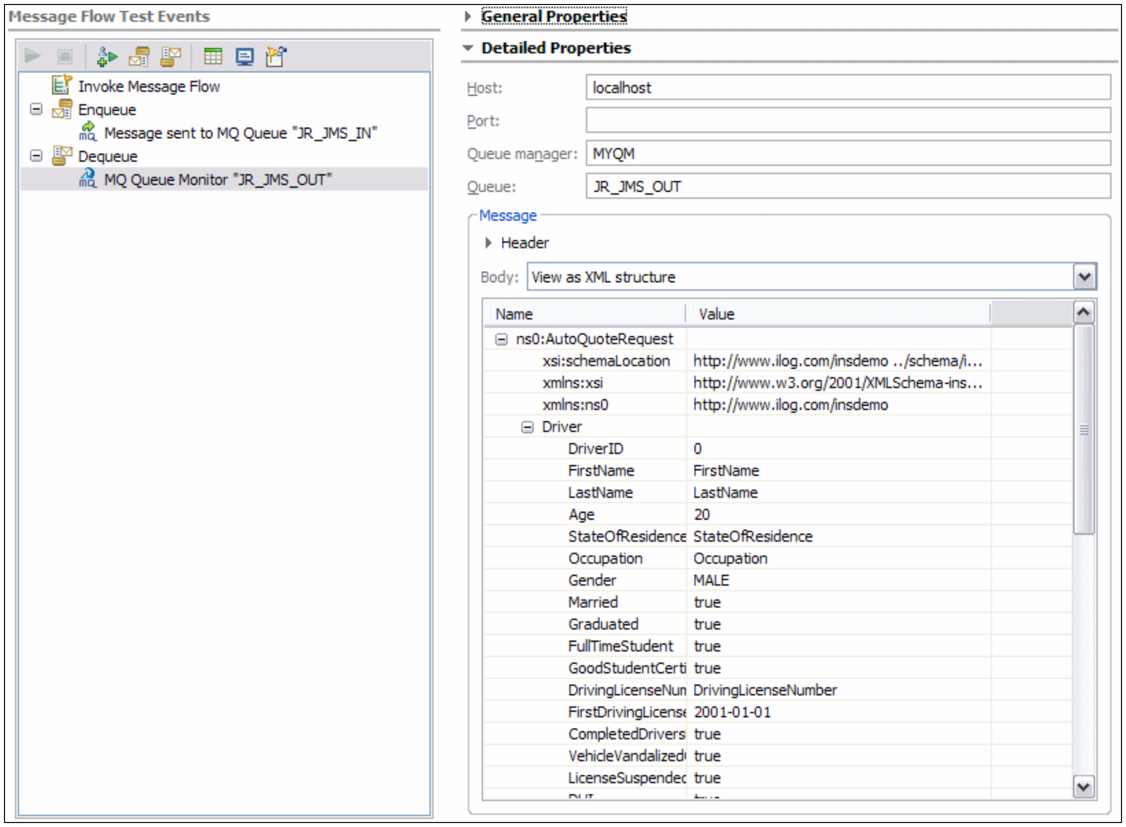
13. Use the test client to get the response message.

14. Click **Dequeue**.

15. In the Detailed Properties section, set the following properties (Figure 11-35):

Host:	`localhost`
Port:	(Leave this field blank)
Queue manager:	BRKR.QM
Queue:	JR_JMS_OUT

16. Click **Get Message.** The response message is displayed in the Message section, as shown in Figure 11-35.



Host: localhost
Port:
Queue manager: MYQM
Queue: JR_JMS_OUT

Message
Header
Body: View as XML structure

Name	Value
ns0:AutoQuoteRequest	
xsi:schemaLocation	http://www.ilog.com/insdemo ../schema/i...
xmlns:xsi	http://www.w3.org/2001/XMLSchema-ins...
xmlns:ns0	http://www.ilog.com/insdemo
Driver	
DriverID	0
FirstName	FirstName
LastName	LastName
Age	20
StateOfResidence	StateOfResidence
Occupation	Occupation
Gender	MALE
Married	true
Graduated	true
FullTimeStudent	true
GoodStudentCerti	true
DrivingLicenseNun	DrivingLicenseNumber
FirstDrivingLicens	2001-01-01
CompletedDrivers	true
VehicleVandalized	true
LicenseSuspendec	true

Figure 11-35 Test response message

17. Check that the vehicle value has been updated by the rules.

18. Repeat the test for the other XML payload files and check that the vehicle value results are as shown in Table 11-2.

Table 11-2 Results

XML sample	Vehicle value
`data/xml/insdemoObjectModel_case_1.xml`	97.0
`data/xml/insdemoObjectModel_case_2.xml`	110.0
`data/xml/insdemoObjectModel_case_3.xml`	98.0
`data/xml/insdemoObjectModel_case_4.xml`	100.0

11.4.8 Implementation extensions and alternatives

In this section, we look at Implementation extensions and alternatives.

Flow design

The flows in the scenarios are kept simple for illustrative purposes and therefore do not include important features and functions. For example, error checking, failure/catch processing, and trace.

The request and reply flows do not correlate request and reply messages to and from the RES. The lack of correlation raises the risk of message processing problems related to “missing” messages and message ordering. In your environment, be sure that request and replies are correlated.

In the scenario, the reply flow creates a new message, meaning that data (for example, message header properties) from the original input message is not passed to the back-end system. If request and reply messages are correlated, the header data can be retained for the outbound message.

The JMSHeader node in the request flow provides an easier way to set JMS header properties. These properties may alternatively be set programmatically in the JavaCompute node.

The scenario uses an identical format of messages for the input to the request message flow and the input to the JRules rule application. There are reasons why other scenarios would use differing formats (for example, the input message is large and only a subset of its content is required by the rule application). In cases where the message formats differ, WebSphere Message Broker provides powerful features to transform message content.

WebSphere MQ configurations

For simplicity, the scenario uses a single WebSphere MQ queue manager for both WebSphere Message Broker and WebSphere Application Server. However, many alternatives to this messaging topology are available, as in the following list:

- Use separate WebSphere MQ queue managers for WebSphere Message Broker and WebSphere Application Server, connected by channels or a WebSphere MQ cluster.
- Use a WebSphere MQ queue manager for WebSphere Message Broker and use WebSphere Application Server default messaging.

Use of alternative JNDI is also an option.

JRules MDB

The JRules MDB that is provided with the WebSphere ILOG JRules product provides the basic function to route messages to the correct rule application in the RES. You may also write your own MDB if additional function is required, such as the following additional uses:

- To support message formats other than JMS object messages or to add logging features.
- To simplify the message flows (for example, by removing JavaCompute nodes that are used to transform the content for the MDB).

Tightly coupled patterns

Consider tight coupling of rules with WebSphere Message Broker when performance is one of the highest priorities in terms of requirements for a solution. Recognize that developing a solution with a J2SE approach is a more advanced development task, requiring appropriate skills. If skills are insufficient to implement the scenario and that performance service level agreements (SLA) allow, use the loosely coupled integration patterns described in this book instead.

Other considerations

Although security considerations are beyond the scope of the scenarios that are covered in this book, always consider them. Those considerations can include, for example, authority to access components and objects, encryption of messages, and so on.

Systems and application monitoring is a vital parts to any production environment.

11.5 Scenario 4: SCA integration

This section describes how to call business rules running in WebSphere ILOG JRules from WebSphere Message Broker by using SCA.

11.5.1 Introduction

For the same reasons that are explained in the introductions to Scenarios 1 and 2, IBM Redbooks Company Insurance has decided to implement message routing and enrichment decisions as rules in WebSphere ILOG JRules and to invoke these rules from WebSphere Message Broker.

In Scenario 4, IBM Redbooks Company Insurance is also in the process of deploying BPM products (WebSphere Process Server) to improve operational efficiency and business agility. The rules that are deployed in WebSphere ILOG JRules will be used by both BPM products and the connectivity layer. In this scenario, the BPM products are of most importance and are therefore key force when choosing the integration pattern. This integration pattern has resulted in WebSphere ILOG JRules rules being exposed as SCA components.

You have collaborated with the WebSphere ILOG JRules developer and BPM developers to understand the existing rules and SCA interfaces. You now have to develop the new message flows to invoke the SCA component rules. In this scenario, we also describe how to deploy the rule as an SCA component.

Service Component Architecture (SCA) and JRules

An SCA component is run inside an SCA run time. WebSphere Message Broker does not provide an SCA run time, but it does provide SCA nodes that can be used to invoke SCA components.

This scenario uses WebSphere Application Server V7 and the Feature Pack for Service Component Architecture as a run time, though other run times exist, as described in the following chapters:

- Chapter 7, “Business processes” on page 95
- Chapter 10, “Integrating WebSphere Process Server with JRules” on page 239

The ways to generate a rule application as an SCA component are as follows:

- WebSphere ILOG JRules 7.0.3 contains a code generator, which is a good starting point if you are not familiar with SCA:

  http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m3/index.jsp?topic=/com.ibm.websphere.ilog.jrules.doc/Content/Business_Rules/Documentation/_pubskel/JRules/ps_JRules_Global835.html

- SupportPac LA71 contains a code generator:

  ftp://public.dhe.ibm.com/software/integration/support/supportpacs/individual/la71_readme.html

The scenario uses SupportPac LA71 to easily integrate WebSphere ILOG JRules 7.0.3 in WebSphere Message Broker 7.0.

Overview of products

This SCA integration scenario requires the following products:

- WebSphere ILOG JRules 7.0.3
- One of the supported J2EE application server runtime environments, such as WebSphere Application Server 7.0 with the SCA FeaturePack
- WebSphere ILOG JRules SupportPac LA71
- WebSphere Message Broker 7.0
- WebSphere Message Broker Toolkit 7.0

The scenario uses a single machine for the development and testing of the integration. Clearly, this configuration is not representative of a physical topology that would be used in a production deployment, but it does not materially affect the integration use case or the guidance in this scenario.

The tooling and development are supported by the following products:

- WebSphere ILOG JRules Rule Studio 7.0.3
- WebSphere Message Broker Toolkit 7.0
- WebSphere Integration Developer 7.0 with the SCA FeaturePack

The runtime environment is supported by the following products:

- A WebSphere Message Broker broker
- A WebSphere MQ queue manager
- An SCA runtime server, for example, a WebSphere Application Server server with the SCA FeaturePack

Figure 11-36 shows the runtime and tooling environments in this scenario.

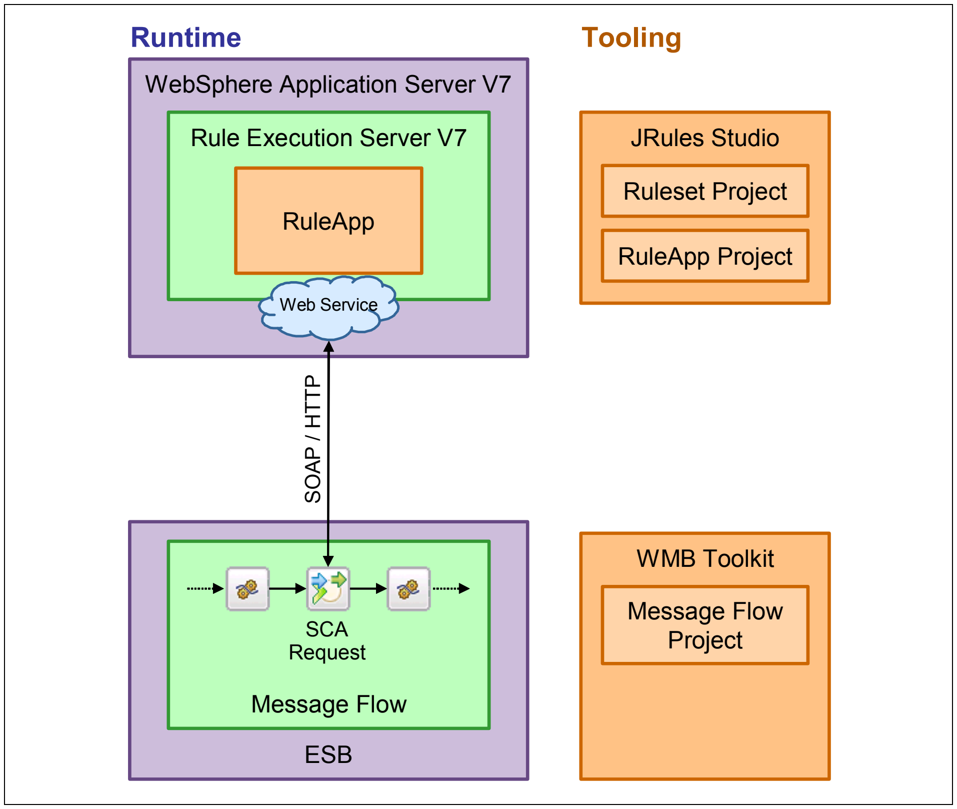


Figure 11-36 Scenario overview

11.5.2 Prepare JRules

This section gives an overview of deploying the RES to WebSphere Application Server and deploying the rule sets to the RES.

Perform the following steps:

1. Deploy the RES to your chosen runtime environment. For instructions, see “Installing Rule Execution Server on WebSphere Application Server V7.0” (under WebSphere ILOG JRules 7.0.3: Java EE add-ons in the information center):

   `http://publib.boulder.ibm.com/infocenter/brjrules/v7r0m3/topic/com.ibm.websphere.ilog.jrules.install.doc/Content/Business_Rules/Documentation/_pubskel/JRules_Application_Servers/ps_Installing_JRules_IC99.html`

2. Start Rules Studio and open the JRulesStudio_Workspace_Java (included in the download.)
3. Click the **Project** menu and select **Clean**.
4. Select **Clean All Projects** and click **OK**.
5. Right-click the **jrules-messaging-integration-ruleapp** project in the Rule Explorer, select **RuleApp**, and then click **Deploy**.
6. Leave the deployment type as the default and click **Next**.
7. Click the **Create a temporary Rule Execution Server** configuration and enter the URL, login, and password information for the RES.
8. Click **Finish** to deploy the rule application.
9. Open the RES console in your web browser (for example, `http://machine:9082/res`).
10. Enter the user name and password of the RES administrator that you configured when deploying the RES and click **Sign in** to open the RES console.
11. Click **Explorer** and verify that the **JRulesMessagingIntegrationRuleApp** is listed in the RuleApps.

Your JRules run time is now ready to be used by WebSphere Message Broker. Perform *either* of the following steps:

- Use predeveloped message broker resources in the download, by following the instructions in 11.5.3, “Use predeveloped WebSphere Message Broker resources” on page 334,
- Develop the message broker resources by following the instructions in 11.5.5, “Develop WebSphere Message Broker resources” on page 344.

11.5.3 Use predeveloped WebSphere Message Broker resources

You may *either* use predeveloped message broker resources in the download, by following the instructions in this section, or develop your own message broker resources by following the instructions in 11.5.5, “Develop WebSphere Message Broker resources” on page 344 to develop the message broker resources.

Perform the following steps to use predeveloped resources:

1. Start WebSphere Message Broker Toolkit and open WMB_Workspace_SCA (included in the download). The following projects are listed in the Projects pane of your workspace:

   ```
   jrules-messaging-integration-messageflow
   jrules-messaging-integration-messageset
   ```

2. Click the Project menu and select **Clean**.
3. Select **Clean All Projects** and click **OK**.

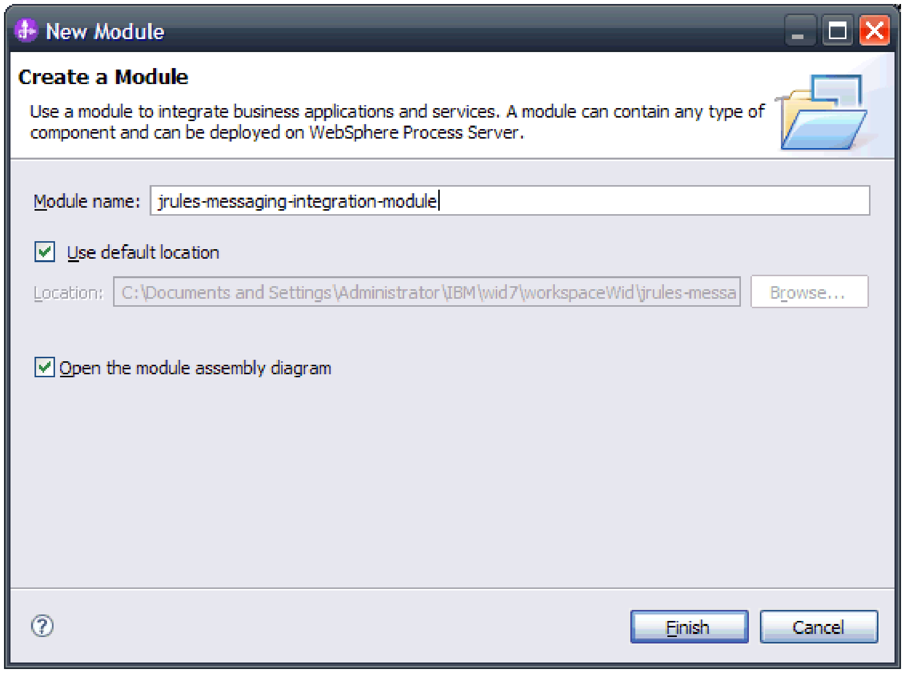
You may now deploy the resources required for the solution by following the instructions in 11.5.6, “Deploy the WebSphere Message Broker message flows” on page 355.

11.5.4 Generate a RES call using SupportPac LA71

Use either the predeveloped message broker resources in the download (11.5.3, “Use predeveloped WebSphere Message Broker resources” on page 334) or develop the message broker resources using the instructions in 11.5.5, “Develop WebSphere Message Broker resources” on page 344.

Generate an SCA component that calls the RES, using the SupportPac LA71:

1. Start WebSphere Integration Developer and create a new workspace (which we refer to as `workdir/workspaceWid` in the remaining text).
2. Click **File** → **New** → **Module**. Enter a name for the module and click **Finish**, as shown in Figure 11-37. We will later add business process and WebSphere ILOG JRules SCA component to this module.


New Module

Create a Module

Use a module to integrate business applications and services. A module can contain any type of component and can be deployed on WebSphere Process Server.

Module name: jrules-messaging-integration-module

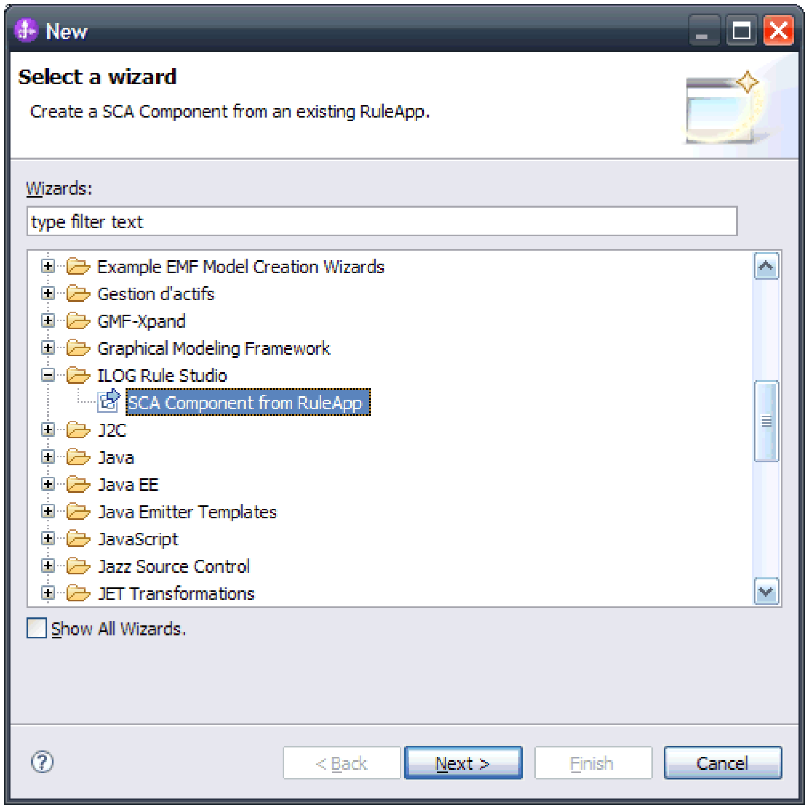
☑ Use default location

Location: C:\Documents and Settings\Administrator\IBM\wid7\workspaceWid\jrules-messa

☑ Open the module assembly diagram

Figure 11-37 Create a module

3. Click **File** → **New** → **Others**, and select **ILOG Rule Studio** → **SCA Component from RuleApp**, as shown in Figure 11-38. Click **Next** twice.



New

Select a wizard

Create a SCA Component from an existing RuleApp.

Wizards:

type filter text

- Example EMF Model Creation Wizards
- Gestion d'actifs
- GMF-Xpand
- Graphical Modeling Framework
- ILOG Rule Studio
  - SCA Component from RuleApp
- J2C
- Java
- Java EE
- Java Emitter Templates
- JavaScript
- Jazz Source Control
- JET Transformations

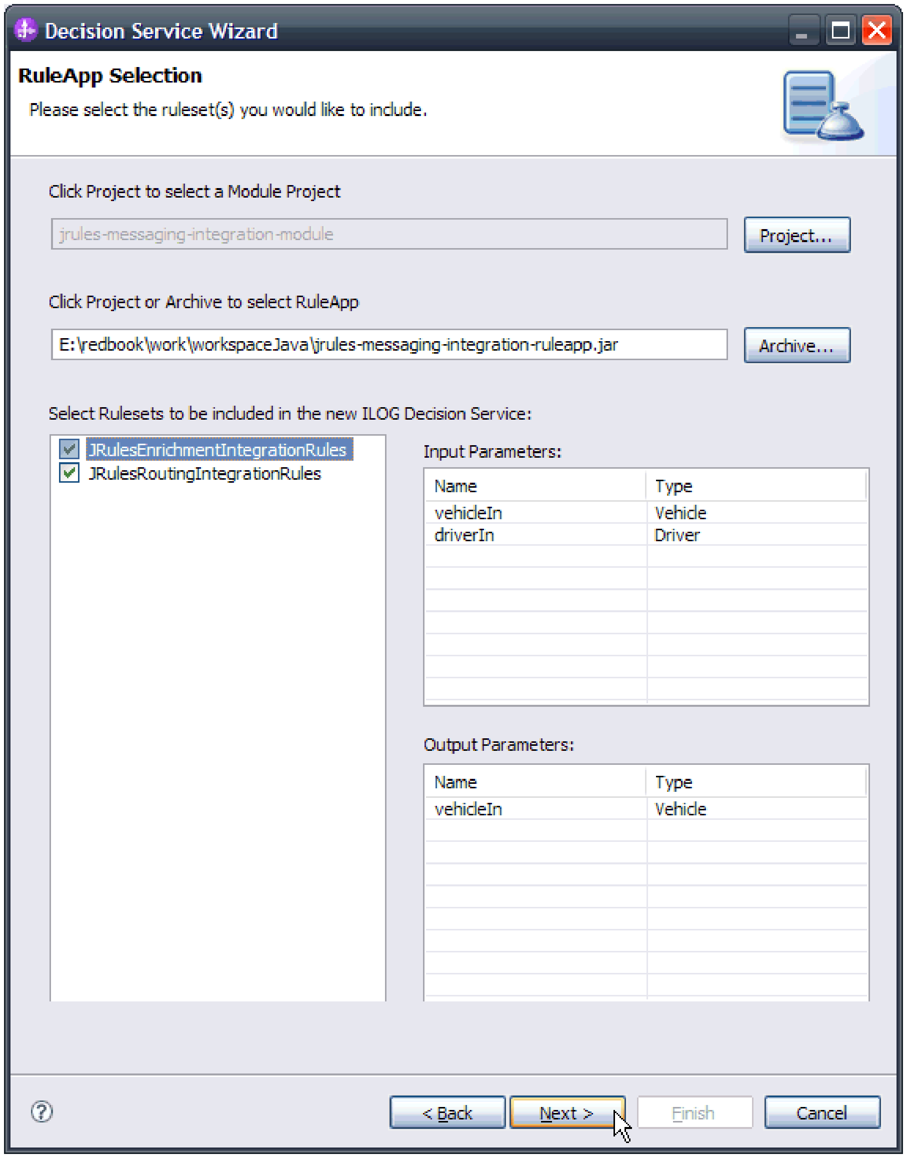
☐ Show All Wizards.

Figure 11-38 Select the SCA Component from RuleApp

4. Click **Archive** and browse to the following file, as shown in Figure 11-39 on page 337:

   ```
   workdir/workspaceJava/jrules-messaging-integration-ruleapp.jar
   ```

5. Click **Next**.



RuleApp Selection

Please select the ruleset(s) you would like to include.

Click Project to select a Module Project

jrules-messaging-integration-module

Click Project or Archive to select RuleApp

E:\redbook\work\workspaceJava\jrules-messaging-integration-ruleapp.jar

Select Rulesets to be included in the new ILOG Decision Service:

- ✓ JRulesEnrichmentIntegrationRules
- ✓ JRulesRoutingIntegrationRules

Input Parameters:

Name	Type
vehicleIn	Vehicle
driverIn	Driver

Output Parameters:

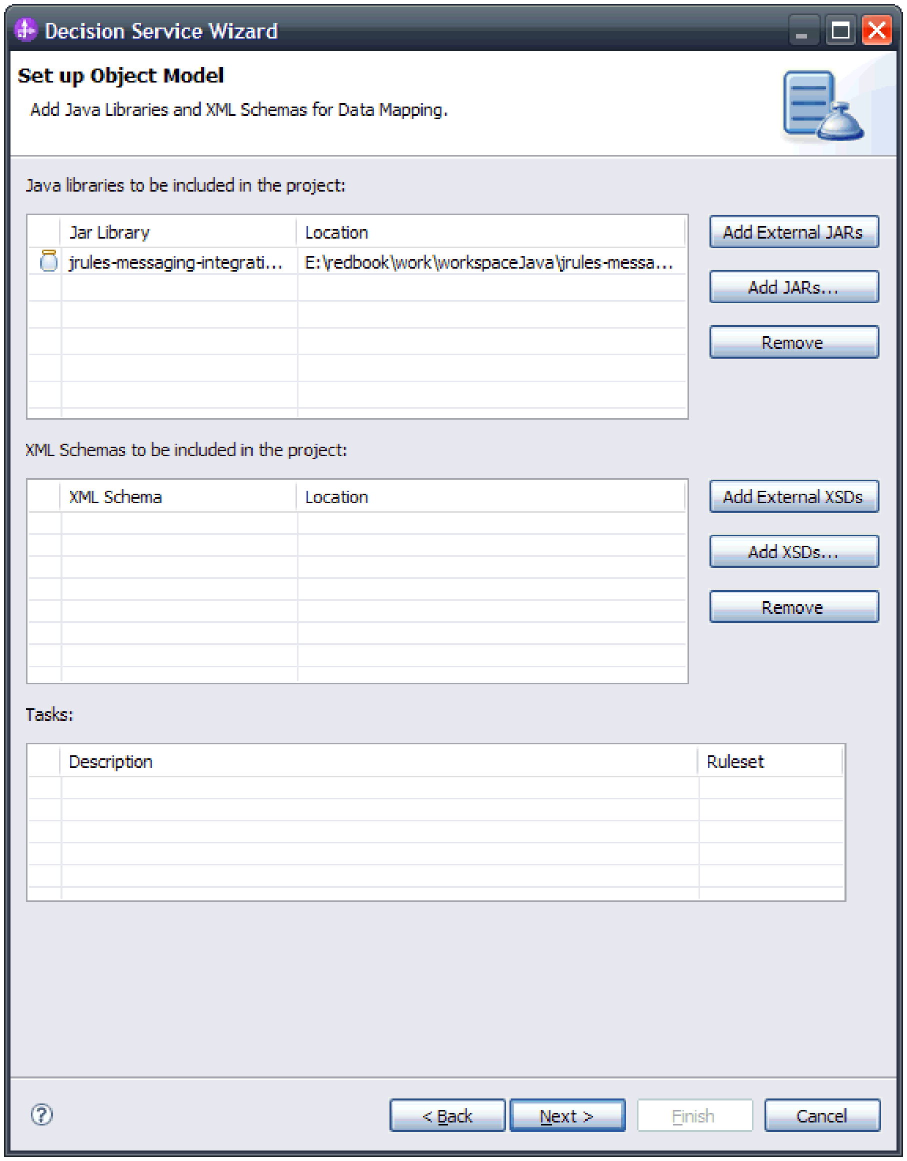
Name	Type
vehicleIn	Vehicle

Figure 11-39 Decision Service Wizard- Step 1

6. Add the following Java XOM file, as shown in Figure 11-40:

   `workdir/workspaceJava/jrules-messaging-integration-java-xom.jar`

7. Click **Next**.

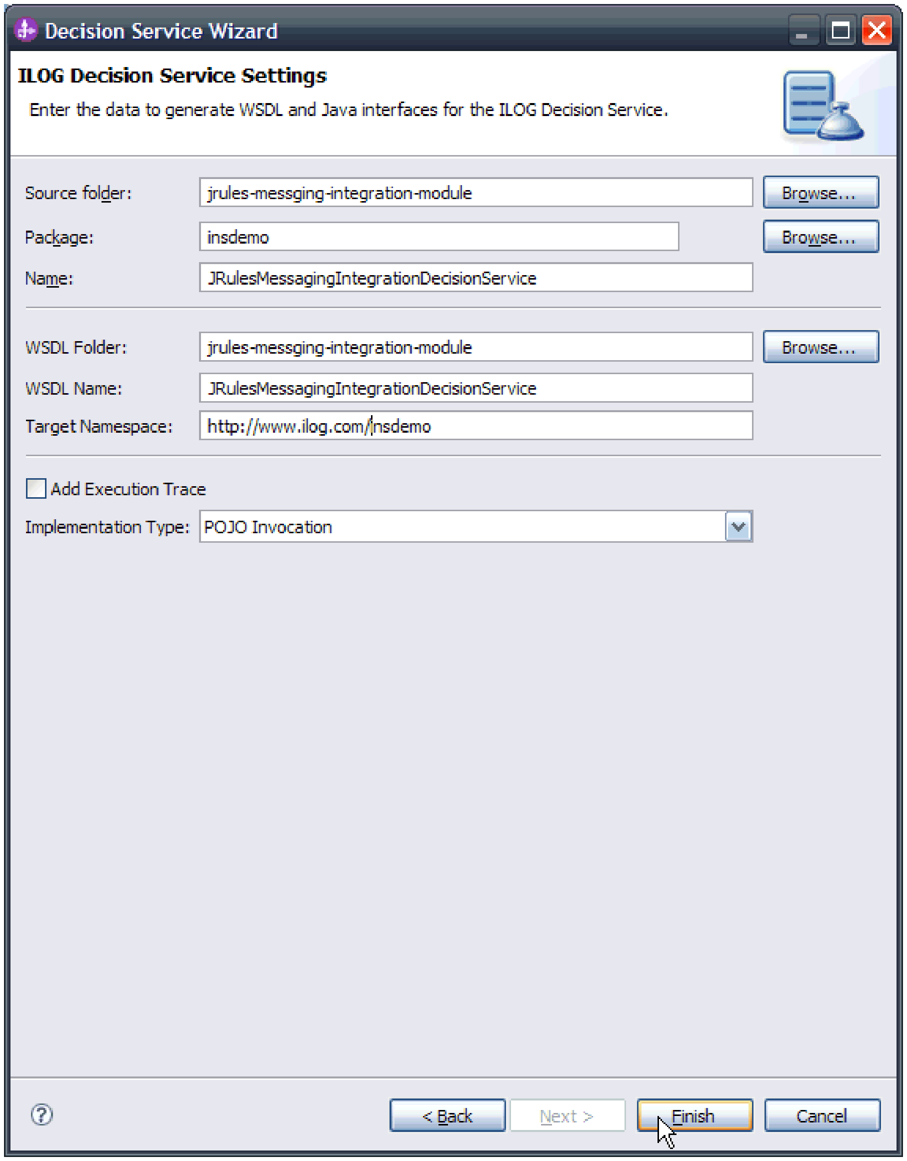


Decision Service Wizard
Set up Object Model
Add Java Libraries and XML Schemas for Data Mapping.
Java libraries to be included in the project:

Jar Library	Location
jrules-messaging-integrati...	E:\redbook\work\workspaceJava\jrules-messa...

Figure 11-40   Decision Service Wizard - Step 2

8. Complete the Name of the service and the POJO Implementation, as shown in Figure 11-41.


Decision Service Wizard

ILOG Decision Service Settings

Enter the data to generate WSDL and Java interfaces for the ILOG Decision Service.

Source folder: jrules-messging-integration-module

Package: insdemo

Name: JRulesMessagingIntegrationDecisionService

WSDL Folder: jrules-messging-integration-module

WSDL Name: JRulesMessagingIntegrationDecisionService

Target Namespace: http://www.ilog.com/insdemo

Add Execution Trace

Implementation Type: POJO Invocation

Figure 11-41 Decision Service Wizard - Step 3

Figure 11-42 shows the generation of the SCA component.

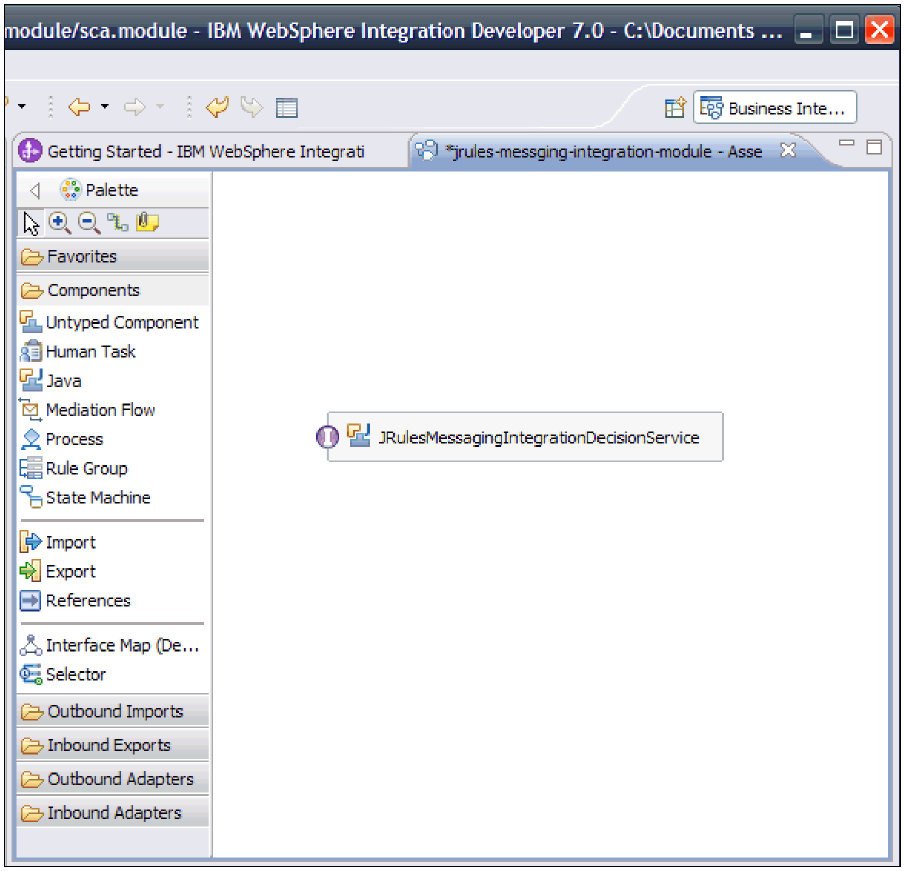


Figure 11-42 Generation of the SCA component

9. Generate an SCA binding, as shown in Figure 11-43.

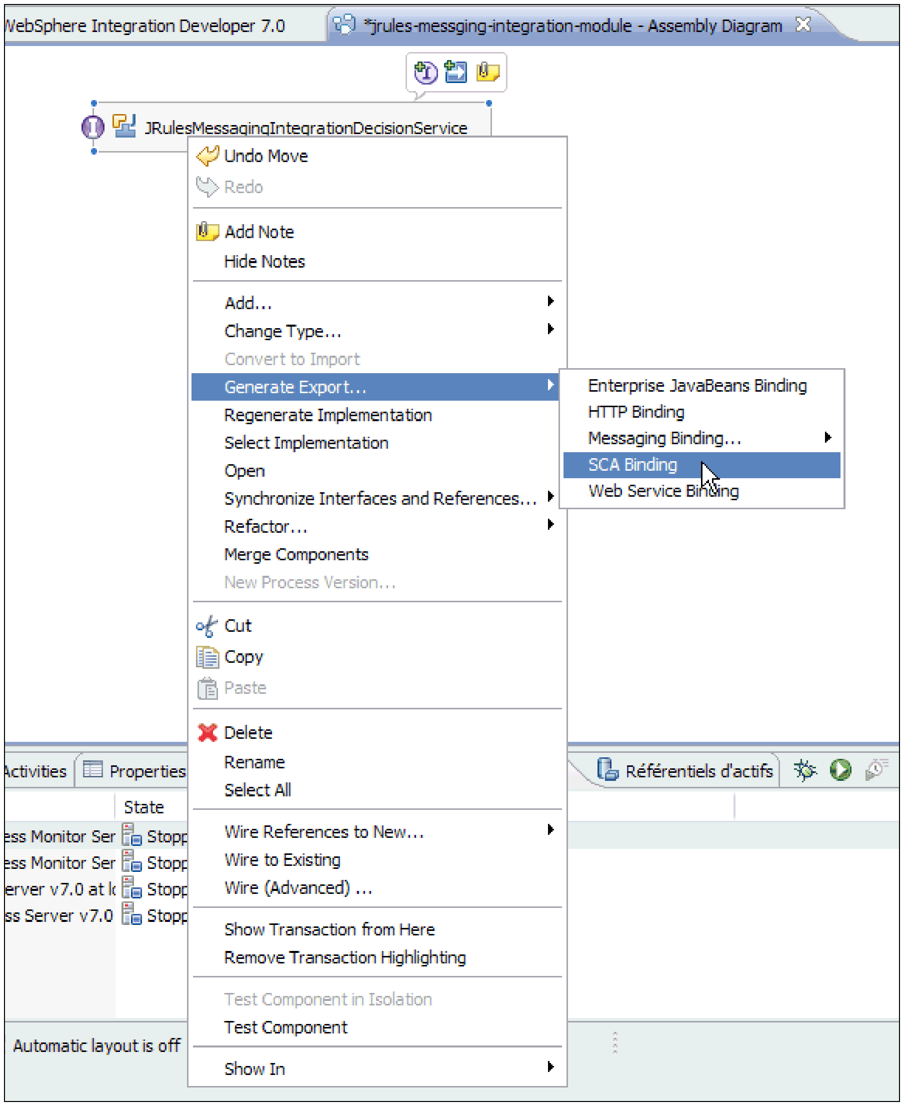


Figure 11-43 Generate an SCA Binding

Figure 11-44 shows the result.

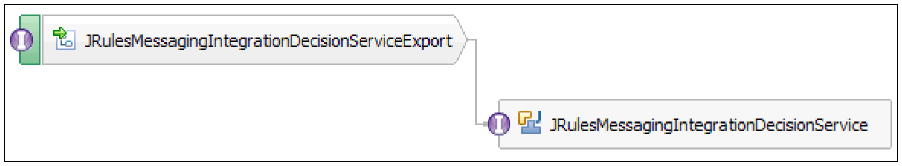
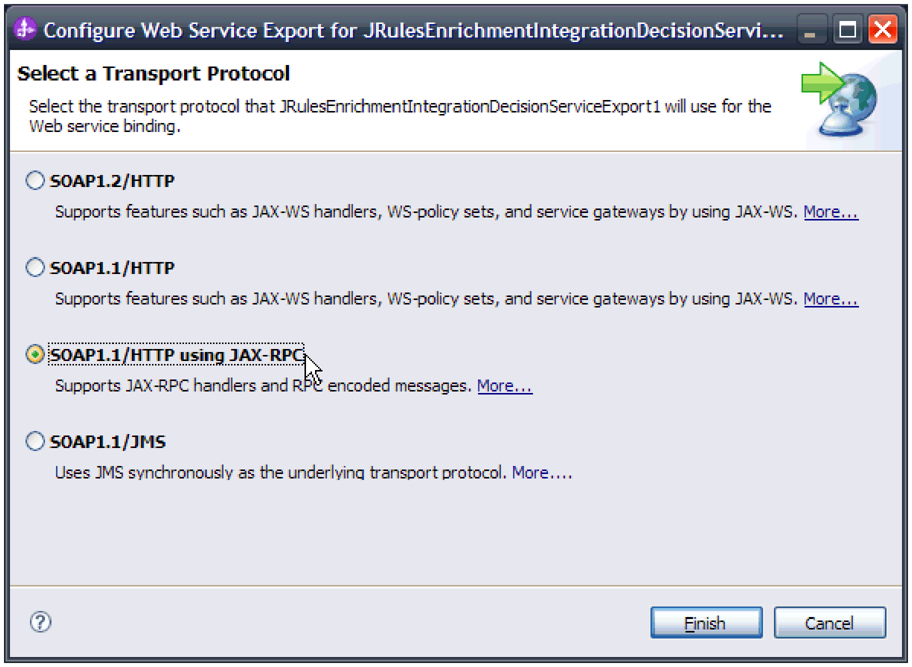

JRulesMessagingIntegrationDecisionServiceExport
JRulesMessagingIntegrationDecisionService

Figure 11-44 Generation of the SCA Binding

10. Generate a web service binding too, using the SAOP 1.1 / JAX-RPC support, as shown in Figure 11-45 and Figure 11-46.


Configure Web Service Export for JRulesEnrichmentIntegrationDecisionServi...

Select a Transport Protocol

Select the transport protocol that JRulesEnrichmentIntegrationDecisionServiceExport1 will use for the Web service binding.

- ○ **SOAP1.2/HTTP**
  Supports features such as JAX-WS handlers, WS-policy sets, and service gateways by using JAX-WS. More...
- ○ **SOAP1.1/HTTP**
  Supports features such as JAX-WS handlers, WS-policy sets, and service gateways by using JAX-WS. More...
- [✓ **SOAP1.1/HTTP using JAX-RPC**]
  Supports JAX-RPC handlers and RPC encoded messages. More...
- ○ **SOAP1.1/JMS**
  Uses JMS synchronously as the underlying transport protocol. More....

Figure 11-45 Select a Transport Protocol

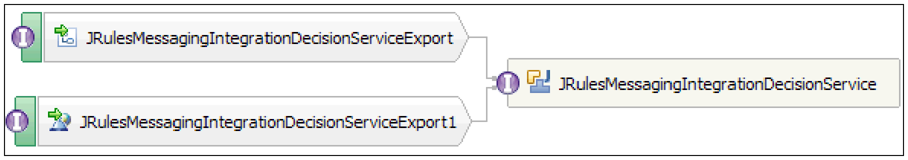

JRulesMessagingIntegrationDecisionServiceExport
JRulesMessagingIntegrationDecisionServiceExport1
JRulesMessagingIntegrationDecisionService

Figure 11-46 Generation of the web service interface

Tip: If an error message about the Java Facet that used is displayed, go to the Properties of your project and change the Java Compiler to a 1.6 version or use the Quick Fix resolution.

11. Export as a Project Interchange format (by clicking **Other** → **Project Interchange**). Click **Next**. See Figure 11-47.

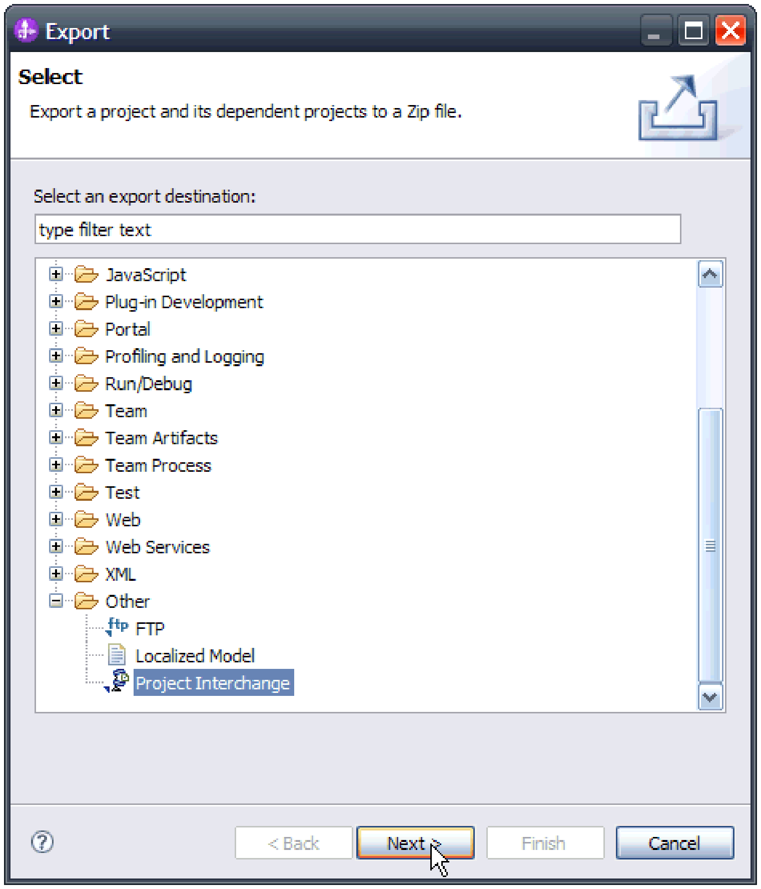
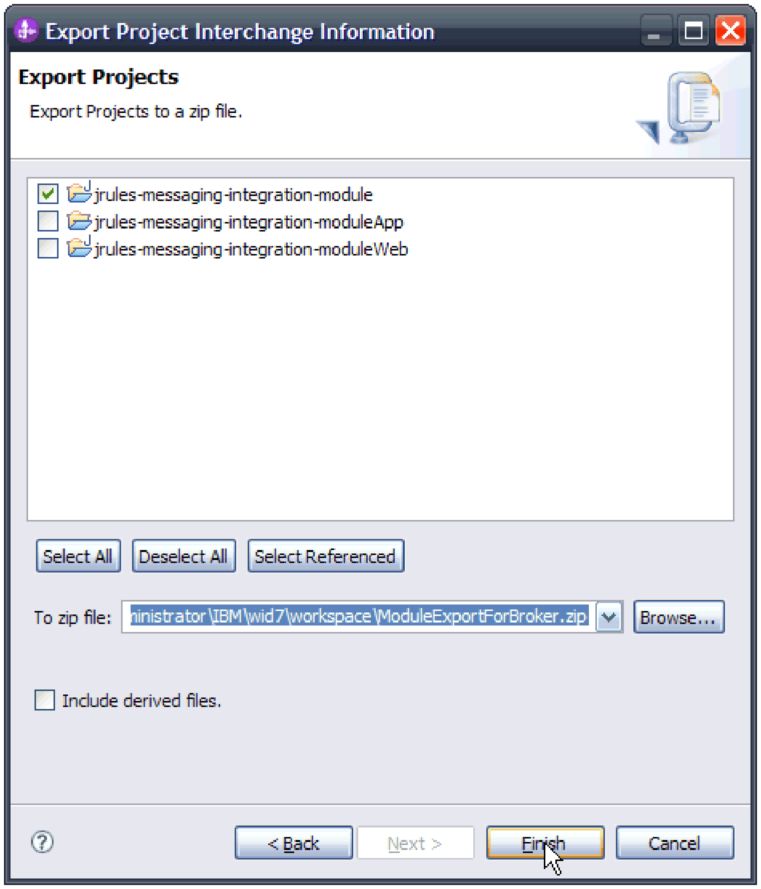


Figure 11-47 Export of the SCA component

12. Select your Module and enter the following path:

   workdir`/workspaceWid/ModuleExportForBroker.zip`

13. Deploy your SCA component into your WebSphere Application Server (with an SCA feature Pack), as shown in Figure 11-48 on page 344. Use the Installation Manager to update your environment, if needed. Click **Finish**.


Export Project Interchange Information

Export Projects

Export Projects to a zip file.

- [✓] jrules-messaging-integration-module
- [ ] jrules-messaging-integration-moduleApp
- [ ] jrules-messaging-integration-moduleWeb

To zip file: ninistrator\IBM\wid7\workspace\ModuleExportForBroker.zip

[ ] Include derived files.

Figure 11-48 Export Project Interchange Information wizard

11.5.5 Develop WebSphere Message Broker resources

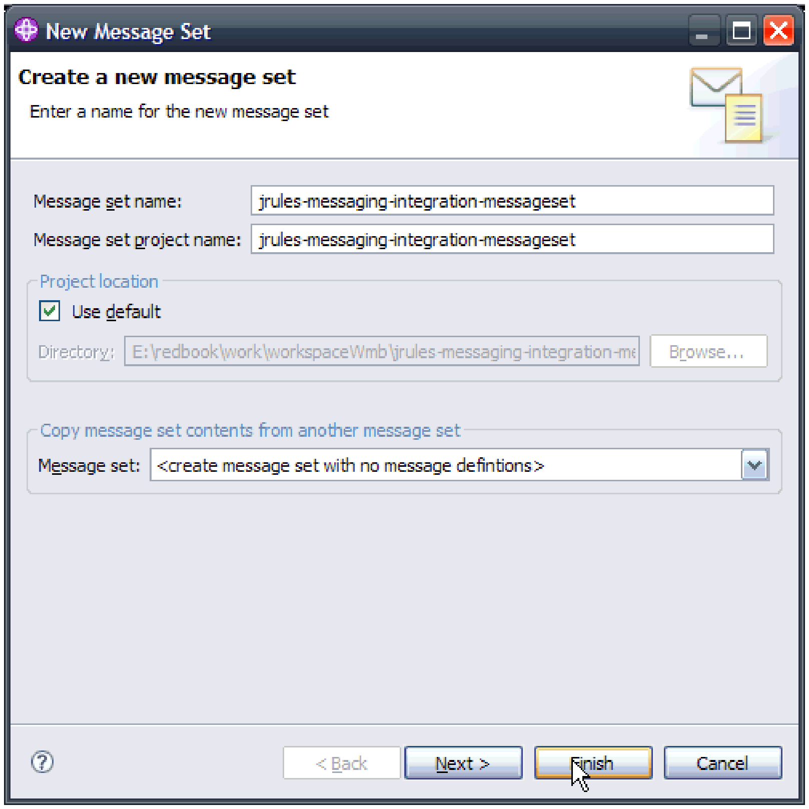
This section describes the steps to develop the WebSphere Message Broker resources. Message flow development is carried out in the WebSphere Message Broker Toolkit, which is used for the following tasks:

- Create a message set for the insurance quote message by importing the schema for it.
- Develop the message flow itself.

Create a message set for the insurance quote message

Perform the following steps:

1. Ensure that the WebSphere Message Broker Toolkit 7.0 is started.
2. Create a new message set by clicking **File** → **New** → **Message Set**, as shown in Figure 11-49. Click **Finish.** Click **OK** when the pop-up message displays.


New Message Set

Create a new message set

Enter a name for the new message set

Message set name: jrules-messaging-integration-messageset

Message set project name: jrules-messaging-integration-messageset

Project location

[✓] Use default

Directory: E:\redbook\work\workspaceWmb\jrules-messaging-integration-me

Copy message set contents from another message set

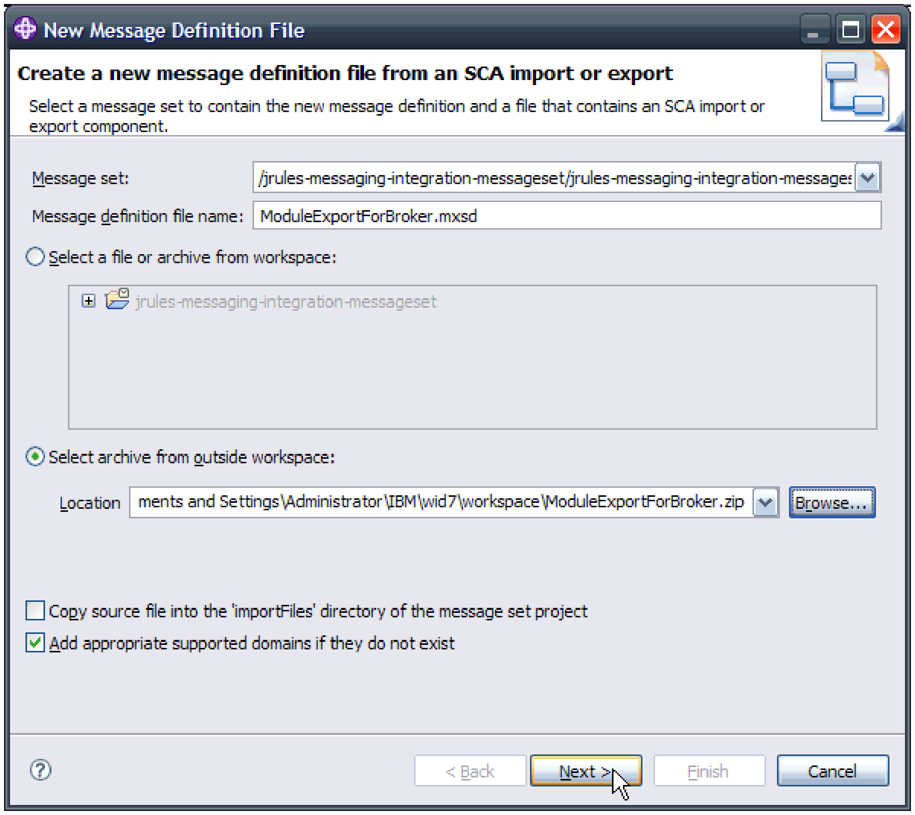
Message set: <create message set with no message defintions>

Figure 11-49 New Message Set wizard: Step 1

3. Create your message set from an SCA import. Right-click the message and click **Set** → **New** → **Message Definition File From** → **SCA Import or Export.**

4. Select an archive from outside workspace and select your WebSphere Integration Developer export, as Figure 11-50 shows:

   workdir/`workspaceWid/ModuleExportForBroker.zip`


New Message Definition File

Create a new message definition file from an SCA import or export

Select a message set to contain the new message definition and a file that contains an SCA import or export component.

Message set: /jrules-messaging-integration-messageset/jrules-messaging-integration-message:

Message definition file name: ModuleExportForBroker.mxsd

○ Select a file or archive from workspace:

jrules-messaging-integration-messageset

● Select archive from outside workspace:

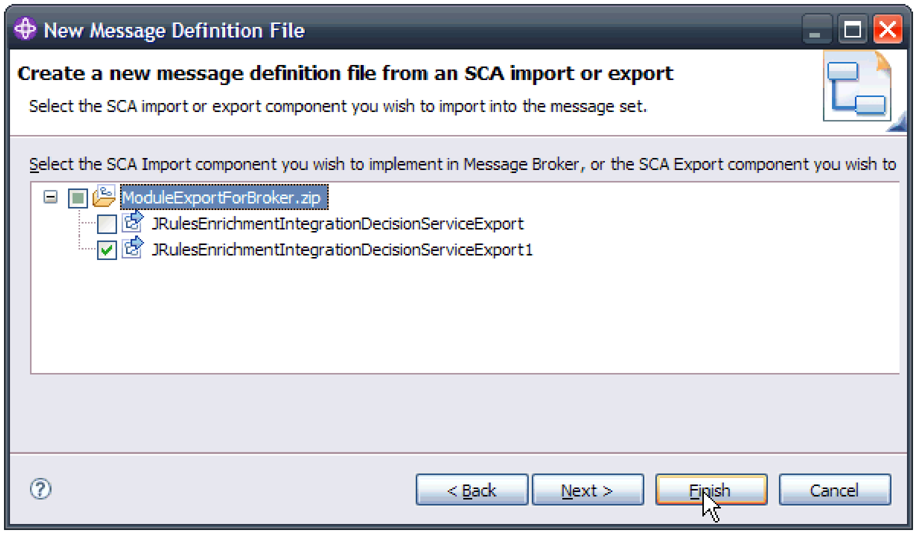
Location: ments and Settings\Administrator\IBM\wid7\workspace\ModuleExportForBroker.zip

☐ Copy source file into the 'importFiles' directory of the message set project

☑ Add appropriate supported domains if they do not exist

Figure 11-50 New Message Set wizard: Step 2

5. The SCA export contains the SCA export and the web service export. Select the web service export. See Figure 11-51.
6. Click **Next** and then click **Finish**.


New Message Definition File

Create a new message definition file from an SCA import or export

Select the SCA import or export component you wish to import into the message set.

Select the SCA Import component you wish to implement in Message Broker, or the SCA Export component you wish to

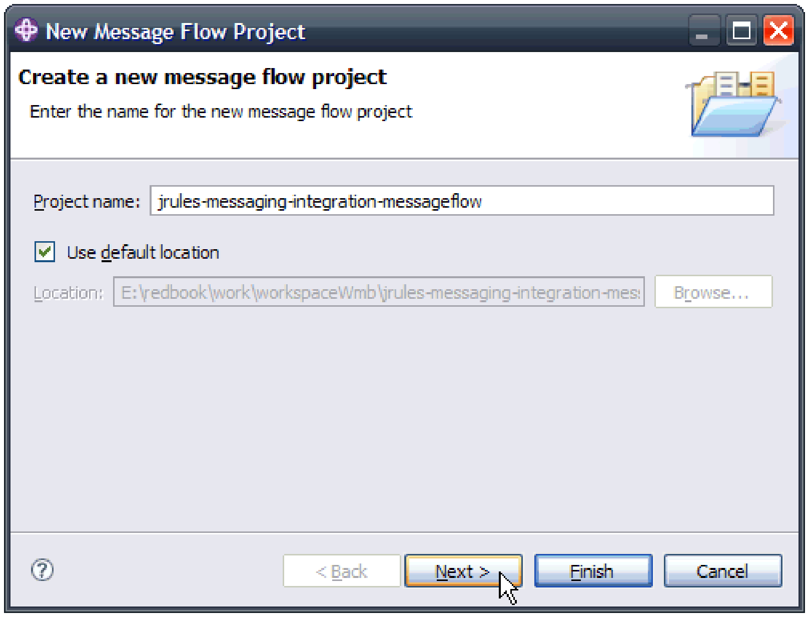
- ModuleExportForBroker.zip
  - [ ] JRulesEnrichmentIntegrationDecisionServiceExport
  - [✓] JRulesEnrichmentIntegrationDecisionServiceExport1

Figure 11-51 New Message Definition File wizard

Develop the message flow

Perform the following steps:

1. Click **File** → **New** → **Message Flow Project**. Click **Next**, as Figure 11-52 shows.



New Message Flow Project

Create a new message flow project

Enter the name for the new message flow project

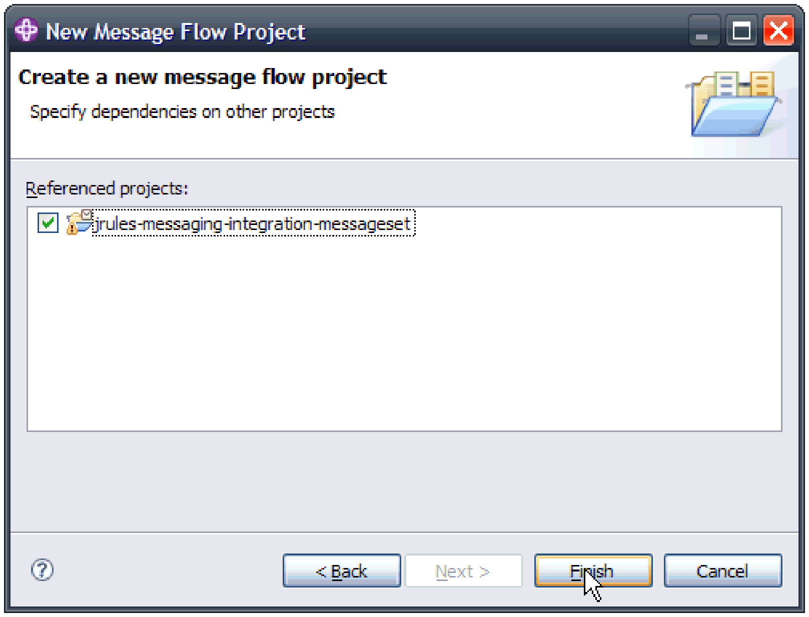
Project name: jrules-messaging-integration-messageflow

[✓] Use default location

Location: E:\redbook\work\workspaceWmb\jrules-messaging-integration-mes

Figure 11-52 New Message Flow Project wizard - Step 1

2. Select your Message Set and click **Finish**, as shown in Figure 11-53 on page 349.


New Message Flow Project

Create a new message flow project

Specify dependencies on other projects

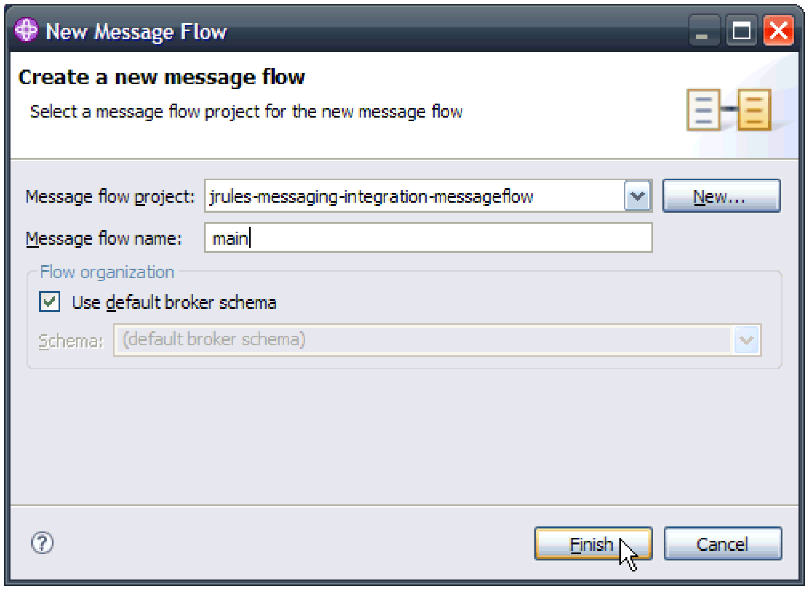
Referenced projects:

- [✓] jrules-messaging-integration-messageset

Figure 11-53 New Message Flow Project wizard - Step 2

3. Create a message flow to test the SCA connection. Right-click the flow and select **Project** → **New** → **Message Flow**.

4. Complete the name and click **Finish**, as Figure 11-54 shows.


New Message Flow

Create a new message flow

Select a message flow project for the new message flow

Message flow project: jrules-messaging-integration-messageflow

Message flow name: main

Flow organization

☑ Use default broker schema

Schema: (default broker schema)

Figure 11-54 New Message Flow wizard

5. In the message flow editor, add the following information:
   - An MQ Input node (name it `SCA.IN`, message domain: XMLNSC and the created message set).
   - An MQ Output node (name it `SCA.OUT`).
   - Optional (for an additional routing node): an MQ Output node (name it `SCA.OTHER.OUT`).

See Figure 11-55.

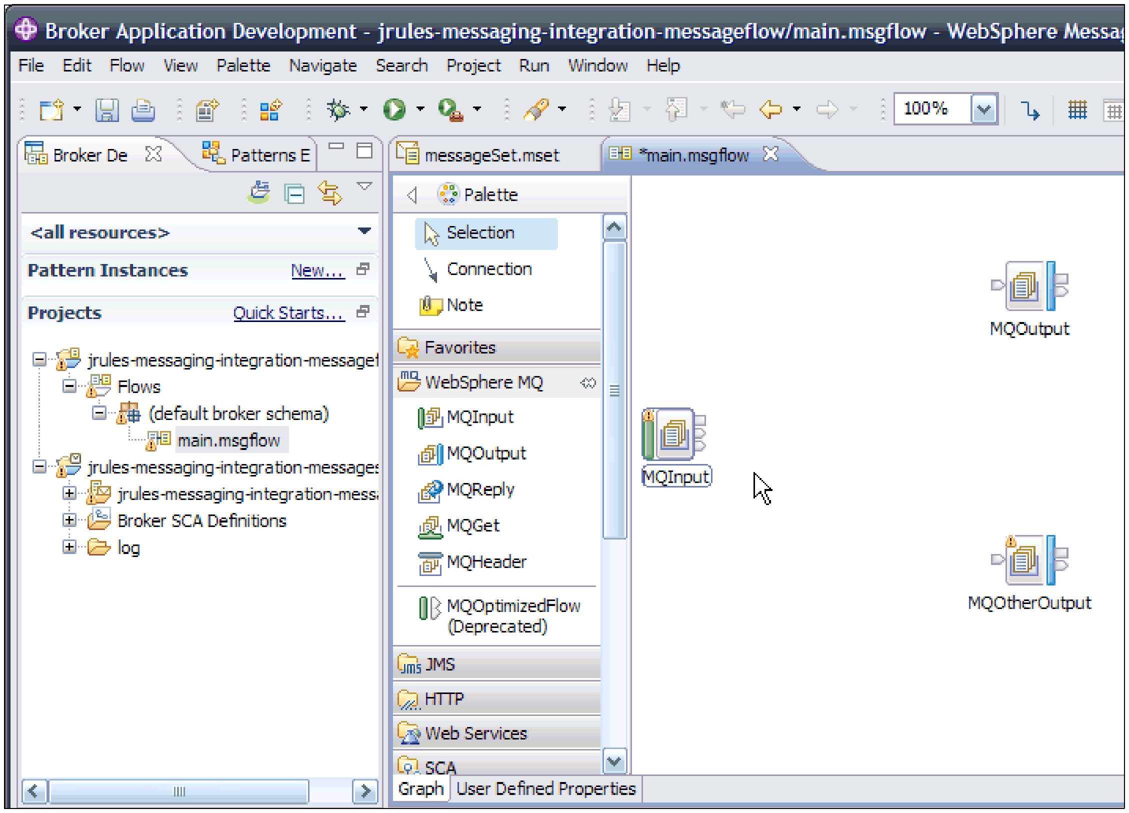


Figure 11-55   Message Flow Definition: Step 1

6. Drag your SCA export to the main.msgflow, shown in Figure 11-56.

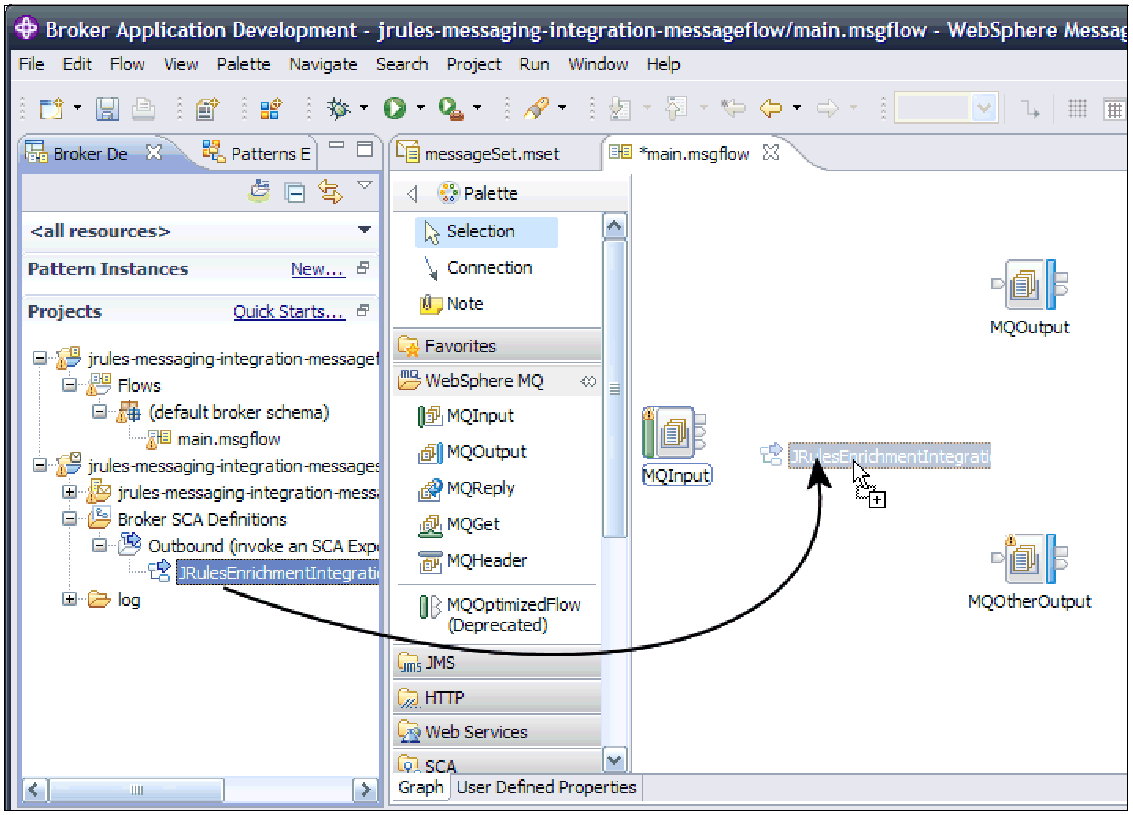
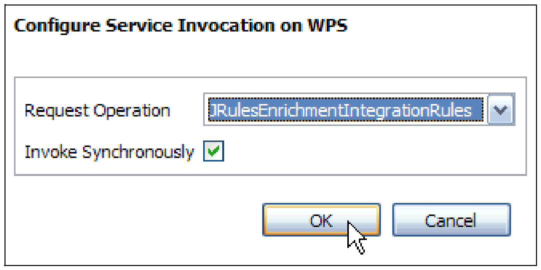

Broker Application Development - jrules-messaging-integration-messageflow/main.msgflow - WebSphere Messag
File Edit Flow View Palette Navigate Search Project Run Window Help
Broker De
Patterns E
<all resources>
Pattern Instances
New...
Projects
Quick Starts...
jrules-messaging-integration-messagef
Flows
(default broker schema)
main.msgflow
jrules-messaging-integration-messages
jrules-messaging-integration-mess
Broker SCA Definitions
Outbound (invoke an SCA Expo
JRulesEnrichmentIntegrati
log
messageSet.mset
*main.msgflow
Palette
Selection
Connection
Note
Favorites
WebSphere MQ
MQInput
MQOutput
MQReply
MQGet
MQHeader
MQOptimizedFlow (Deprecated)
JMS
HTTP
Web Services
SCA
Graph
User Defined Properties
MQOutput
MQInput
JRulesEnrichmentIntegrati
MQOtherOutput

Figure 11-56 Message Flow Definition: Step 2

7. Select your request operation and click **OK**, as shown in Figure 11-57.



Request Operation: JRulesEnrichmentIntegrationRules
Invoke Synchronously: ✓

Figure 11-57 Message Flow Definition - Step 3

8. Add an additional compute node between the MQ input and the SCA node to force the encoding of your message to UTF-8 instead of the default encoding

(probably not supported by JAX-RPC). We change the name to `ChangeCharSet` and double-click the node to edit the code part. Uncomment copy parts and add the following additional code, as shown in Figure 11-58.

```
SET OutputRoot.MQMD.CodedCharSetId = '1208'; (for UTF-8)
```

```
CREATE COMPUTE MODULE main_ChangeCharSet
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyMessageHeaders();
        CALL CopyEntireMessage();
        SET OutputRoot.MQMD.CodedCharSetId = '1208';
        RETURN TRUE;
    END;

    CREATE PROCEDURE CopyMessageHeaders() BEGIN
        DECLARE I INTEGER 1;
        DECLARE J INTEGER;
        SET J = CARDINALITY(InputRoot.*[]);
        WHILE I < J DO
            SET OutputRoot.*[I] = InputRoot.*[I];
            SET I = I + 1;
        END WHILE;
    END;

    CREATE PROCEDURE CopyEntireMessage() BEGIN
        SET OutputRoot = InputRoot;
    END;
END MODULE;
```

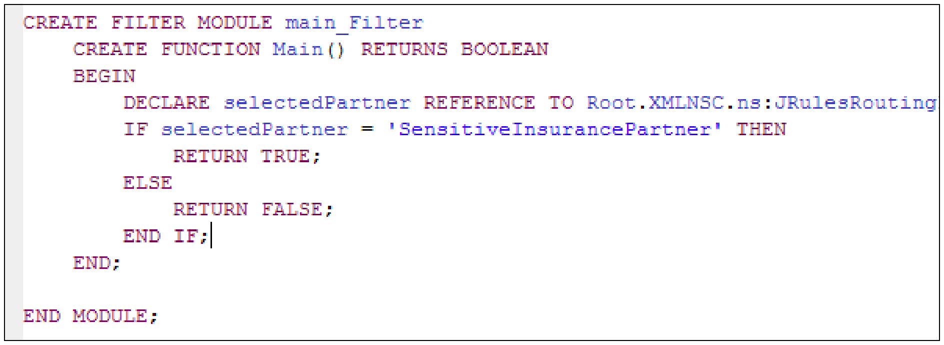
Figure 11-58 Change the CodeCharSetId

You may perform any code transformation in this node if needed.

In our case, we pass the input XML as a request in our WebSphere ILOG JRules RES call.

Additional SCA node to replace old calculated node

Add an additional SCA node (same job as enrichment) to replace old calculated node with a Decision Service call and then a filter node with a condition, as shown in Figure 11-59.



```
CREATE FILTER MODULE main_Filter
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        DECLARE selectedPartner REFERENCE TO Root.XMLNSC.ns:JRulesRouting
        IF selectedPartner = 'SensitiveInsurancePartner' THEN
            RETURN TRUE;
        ELSE
            RETURN FALSE;
        END IF;
    END;

END MODULE;
```

Figure 11-59 Routing with a filter node

A message flow with enrichment and routing node using Decision Service, based on WebSphere ILOG JRules is shown in Figure 11-60. Be sure to proceed to all your message transformations between each node.

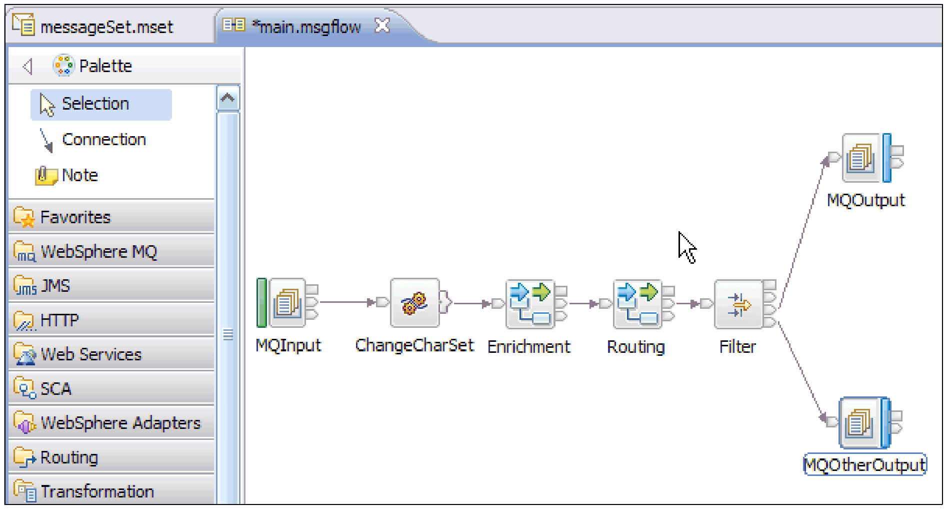


Figure 11-60 Sample message flow for routing

11.5.6 Deploy the WebSphere Message Broker message flows

Perform the following steps:

1. In the WebSphere Message Broker Toolkit, right-click the **jrules-messaging-integration-messageflow** project and select **New** and **Message Broker Archive**. Enter `'sca'` in the Name field and click **Finish**.
2. Select the *main* message flow, the **messageSet** message set, click **Build broker archive,** and then click **OK**.
3. Select the brokers pane and ensure that the BRKR broker and default execution group are running.
4. Expand the tree in the project explorer pane to show the *sca*`.bar` file, as shown in Figure 11-61.

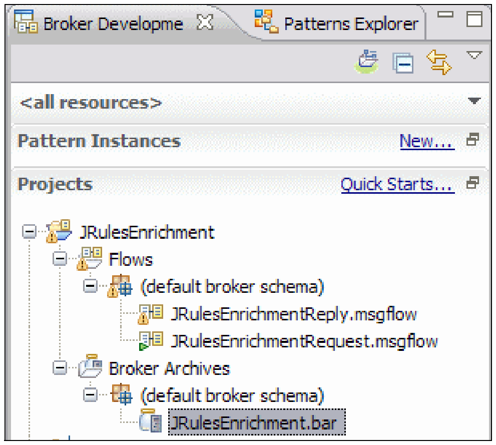


Figure 11-61 Deploy the message flow

5. Click and drag the `.bar` file to the default execution group.

11.5.7 Testing

Perform the following steps:

1. In your message flow editor, right-click an empty part of the editor and click **Test**.
2. In the message part of the test panel, right-click the grid to Add Message Part with the data in Example 11-1 on page 356.

Example 11-1 Add the data

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ins="http://www.ilog.com/insdemo">
 <soapenv:Header/>
 <soapenv:Body>
  <ins:JRulesEnrichmentIntegrationRules>
   <vehicleIn>
    <antiLockBrakes>true</antiLockBrakes>
    <dayTimeRunningLights>true</dayTimeRunningLights>
    <make>Make</make>
    <model>Model</model>
    <modelYear>2009</modelYear>
    <value>100</value>
    <vehicleIdentificationNumber>NN</vehicleIdentificationNumber>
    <vehicleType>
     <name>COUPE</name>
    </vehicleType>
   </vehicleIn>
   <driverIn>
    <age>20</age>
    <completedDriversEdCourse>true</completedDriversEdCourse>
    <DUI>true</DUI>
    <driverID>123</driverID>
    <drivingLicenseNumber>DrivingLicenseNumber</drivingLicenseNumber>
    <firstName>John</firstName>
    <fullTimeStudent>true</fullTimeStudent>
    <gender>
     <name>MALE</name>
    </gender>
    <goodStudentCertificate>true</goodStudentCertificate>
    <graduated>true</graduated>
    <lastName>Doe</lastName>
    <licenseSuspendedOrRevoked>true</licenseSuspendedOrRevoked>
    <married>false</married>
    <numberOfAccidents>0</numberOfAccidents>
    <numberOfTrafficTicket>0</numberOfTrafficTicket>
    <stateOfResidence>CA</stateOfResidence>
    <vehicleVandalizedOrStolen>true</vehicleVandalizedOrStolen>
   </driverIn>
  </ins:JRulesEnrichmentIntegrationRules>
 </soapenv:Body>
</soapenv:Envelope>
```

3. Click **Send Message** and **Finish**.
4. Select your Execution Group and click **Finish**.

You message is enriched, as shown in Example 11-2.

Example 11-2 Message is enriched

```
[snip]
   <JRulesEnrichmentIntegrationRulesExecutionResult>
    <vehicleIn>
     <airbagTypes xsi:nil="true"/>
     <antiLockBrakes>true</antiLockBrakes>
     <dayTimeRunningLights>true</dayTimeRunningLights>
     <make>Make</make>
     <model>Model</model>
     <modelYear>2009</modelYear>
     <value>1.1E+2</value>
     <vehicleIdentificationNumber>NN</vehicleIdentificationNumber>
     <vehicleType>
      <name>COUPE</name>
     </vehicleType>
    </vehicleIn>
   </JRulesEnrichmentIntegrationRulesExecutionResult>
  </in:JRulesEnrichmentIntegrationRulesResponse>
[snip]
```

The value of the vehicle was enhanced because the driver lives in the state of California (CA).

5. Try to change the driver state of residence (to something such as Illinois, IL) and you get a different result, as shown in Example 11-3.

Example 11-3 Different result

```
[snip]
   <JRulesEnrichmentIntegrationRulesExecutionResult>
    <vehicleIn>
     <airbagTypes xsi:nil="true"/>
     <antiLockBrakes>true</antiLockBrakes>
     <dayTimeRunningLights>true</dayTimeRunningLights>
     <make>Make</make>
     <model>Model</model>
     <modelYear>2009</modelYear>
     <value>1E+2</value>
     <vehicleIdentificationNumber>NN</vehicleIdentificationNumber>
     <vehicleType>
      <name>COUPE</name>
     </vehicleType>
    </vehicleIn>
   </JRulesEnrichmentIntegrationRulesExecutionResult>
  </in:JRulesEnrichmentIntegrationRulesResponse>
[snip]
```

11.6 Summary

As with other ESB products, WebSphere Messsage Broker offers a choice of integration patterns and technologies for integrating with JRules. Although each project has its own set of requirements, to choose the correct integration pattern, we make the following general points:

- Web services (hosted transparent decision services) integration is the simplest integration solution we found for XML XOM-based rule sets.
- JMS integration is the best pattern for asynchronous invocation of business rules.
- SCA integration is useful in scenarios where rules have been developed for, or are invoked by, BPM products.

We also note the following information regarding our testing:

- Web services (monitored transparent decision services) integration was also tested with success, although it is available only for Tomcat and JBoss.
- We also successfully tested a JRules Java SE Engine embedded in a JavaCompute node.

A

Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG24Source`

You may also go to the Additional materials website at and search for the folder that corresponds with this IBM Redbooks form number:

`http://www.redbooks.ibm.com/Redbooks.nsf/pages/addmats`

Alternatively, you can go to the IBM Redbooks web site at:

`ibm.com``/redbooks`

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, and look for the `Source.zip` file.

Using the web material

The additional web material that accompanies this book includes the following files:

File name	*Description*
Source.zip	Compressed code samples

Create a subdirectory (folder) on your workstation, and extract the contents of the web material `.zip` file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

Online resources

These web sites are also relevant as further information sources:

- Announcement of EJBs as an integration pattern for CICS

  http://www.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&supplier=897&letternum=ENUS209-135
- Eclipse shell as WebSphere Integration Developer

  http://www.ibm.com/developerworks/data/library/techarticle/dm-0811khatri/index.html?ca=drs-
- WebSphere MQ 7.0.1 + APAR IC68305

  http://www.ibm.com/support/docview.wss?uid=swg1IC68305

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, and also order hardcopy Redbooks publications, at this web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Patterns: Integrating WebSphere ILOG JRules with IBM Software

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Patterns: Integrating WebSphere ILOG JRules with IBM Software

WebSphere ILOG JRules overview and solutions

Patterns, existing applications, BPM, and connectivity

Scenarios with JRules, CICS, WebSphere components

This IBM Redbooks publication describes how the IBM WebSphere ILOG JRules product can be used in association with other IBM middleware products to deliver better solutions.

This book can help architects position a business rule management system (BRMS) in their existing infrastructures to deliver the value propositions that the business needs.

This book can also help developers design and integrate JRules with those middleware products (focussing on WebSphere Process Server, WebSphere Message Broker and IBM CICS) and help to illustrate common integration patterns and practices for these products.

SG24-7881-00 ISBN 073843518X