

Making Better Decisions Using IBM WebSphere Operational Decision Management

Business rules and events in solution applications and processes

Decision management lifecycle and governance guidelines

Best practices for automating decisions



Duncan Clark
Pierre Berlandier



International Technical Support Organization

**Making Better Decisions Using IBM WebSphere
Operational Decision Management**

April 2012

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

First Edition (April 2012)

This edition applies to Version 7.5 of IBM WebSphere Operational Decision Management.

© Copyright International Business Machines Corporation 2012. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	v
Trademarks	vi
Preface	vii
The team who wrote this paper	viii
Now you can become a published author, too!	viii
Comments welcome	ix
Stay connected to IBM Redbooks	ix
Chapter 1. Decision management overview	1
1.1 Decision management technology	2
1.2 WebSphere Operational Decision Management	3
1.3 Decision management solution development process	4
1.4 Insurance demonstration scenario	4
1.5 WODM Insurance solution overview	5
1.6 Website: Eligibility and pricing decisions	5
1.7 Website: Situational decisions	6
1.8 Call center	6
1.9 Underwriting approval process	7
1.10 Decision management and governance	8
Chapter 2. Development process overview	9
2.1 Roles and responsibilities	10
2.2 Decision development lifecycles	10
2.2.1 Service lifecycle	12
2.2.2 Decision lifecycle	13
2.3 Process development	14
2.4 Decision point development	14
2.5 Situation identification and response development	14
2.6 Decision information models and vocabulary	15
2.6.1 Business object definitions	15
2.6.2 Business object field primitives	17
2.6.3 Domains and enumerations	18
2.6.4 Variables	18
Chapter 3. Decision design	21
3.1 Identifying business decisions	22
3.2 Defining the domain of discourse for the rules	24
3.2.1 Tailoring the model for the business users	26
3.2.2 Sharing a common model across decisions	28
3.3 Establishing the decision signature	30
3.4 Decomposing the decision and orchestrating the subtasks	31
3.5 Authoring business rules	32
3.6 Decision integration	33
3.7 Roles, responsibilities, and decision development lifecycle	34
Chapter 4. Situation identification and response development	37
4.1 Situations and events	38
4.2 Defining situations of interest	38

4.3	Identifying business objects and the correlation context	39
4.4	Identifying touchpoint system events and actions	41
4.5	Mapping event data to business objects	45
4.6	Using event rules to invoke actions in response to patterns of events	48
4.7	Mapping business objects to action data and executing the action	50
4.8	Identifying situations using temporal event rules	52
4.9	Identifying situations using contextual event rules	55
4.10	Reacting to identified situations using event rule groups	56
Chapter 5.	Detect-Decide-Respond pattern design	59
5.1	Overview	60
5.2	Using context variables to accumulate situation state	61
5.3	Using data stores to enrich a situation state	64
5.4	Making a decision in response to a situation	66
5.4.1	Creating events and actions from a decision service	66
5.4.2	Triggering the decision from event rules	69
5.5	Responding to a decision	71
5.6	Raising events using web services	73
5.7	Raising events from within rules	77
5.7.1	Creating a JMS event publishing client	78
5.7.2	Exposing the JMS client in the rules business object model	80
5.7.3	Verbalizing the means to publish an event	80
Chapter 6.	Business process design with rules and events	85
6.1	Overview	86
6.2	Identifying the underwriting approval process activities	87
6.3	Defining decision services in a BPM process	88
6.4	Initiating a BPM process from an event	92
6.5	Triggering a BPM process from a WebSphere ODM action	95
6.6	Providing the information for the manual review	101
6.7	Raising an event from within a BPM process	105
Chapter 7.	Managing decision changes across rules, events, and processes	109
7.1	Support for managing and governing change in WebSphere ODM	110
7.1.1	Access control and permission mechanisms	110
7.1.2	Rule and event status	110
7.1.3	Decision test suites	111
7.1.4	Decision simulations	111
7.1.5	Event testing and simulation	112
7.1.6	Deployment baselines	114
7.1.7	Branch management	115
7.2	Rule lifecycle and governance	116
7.3	Decision lifecycle and governance	117
7.3.1	Define phase	117
7.3.2	Deploy phase	119
7.3.3	Monitor and Measure phases	119
7.3.4	Update phase	120
Related publications	121
Other publications	121
Online resources	121
Help from IBM	121

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

IBM®
ILOG®
Redbooks®

Redpapers™
Redbooks (logo) ®
Tivoli®

WebSphere®

The following terms are trademarks of other companies:

Microsoft, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Preface

Decision management is emerging as an important capability for delivering agile business solutions. Decision management is not a solution in its own right, but must be integrated into the solutions or business processes that it supports.

In this IBM® Redpapers™ publication, we describe the recommended best practices and integration concepts that use the business events, business rules, and other capabilities of IBM WebSphere® Operational Decision Management V7.5 (WebSphere ODM) to provide better decision making in those solutions and business processes.

To illustrate these recommended best practices and integration concepts, we use an insurance scenario that is implemented by a number of Web Applications and a business process implemented with IBM Business Process Manager V7.5. The scenario provides the opportunity to show:

- ▶ How decisions can be used by Custom Applications.
- ▶ How decisions can be used in business processes.
- ▶ How to identify situations of interest based on events that occur across the applications and process.
- ▶ How to decide on the action to take when situations are identified.
- ▶ How to manage and govern the design and change of these decisions to optimize the business.

Chapter 1, “Decision management overview” on page 1 provides an introduction to decision management and the WODM Insurance demonstration scenario and solution that is used as an example.

This solution is also available in the IBM SOA Sandbox, allowing readers to explore aspects of how the solution works in more detail. Although the scenario and concepts described here are based on this exercise, there might be implementation differences in the recommended best practices described in this document.

For this reason, it is not our intention to include detailed scenario screen capture or implementation details in this document, but to concentrate on the best practices and patterns for realizing the business values of decision management into service-oriented solution scenarios.

The team who wrote this paper

This paper was produced by a team of specialists from around the world working with the International Technical Support Organization.

Duncan Clark is a Product Manager for WebSphere Operational Decision Management with responsibilities for integration between WebSphere ODM and other IBM Products. He is based in IBM Hursley Labs. Over the last year, he has been responsible for demonstrations and integrations of IBM products, including WebSphere ODM, BPM, and IBM Tivoli® products. He managed the integration of the IBM ILOG® embedded rules into BPM V7.5 and the development of support pack LA71, which provides integration tools and development patterns for using WebSphere ODM with Business Process Manager. Before this assignment, Duncan led the Governance and Policy architecture for IBM WebSphere Service Registry and Repository.

Pierre Berlandier is a Senior Technical Staff Member with IBM Software Services for WebSphere. As part of the ILOG service team for 15 years, Pierre has worked on the different incarnations of rules programming paradigm, from expert systems to real-time intelligent agents, and business rules management systems. As part the Service Engineering group of IBM Software Services for WebSphere, he develops service delivery processes, mentors practitioners, and captures and organizes best practices in the business process and decision management space. Pierre has a Ph.D. in Computer Science from INRIA in France, where he started working on the synergy between the rules and the constraint programming paradigms.

We thank others who contributed to this paper, in the form of written content, advice, and project support. The team acknowledges help from our sponsors and reviewers, including:

Margaret Thorpe, WebSphere ODM Product Management lead
Jerome Boyer, STSM - Software Services for WebSphere
Brett Stineman, WebSphere ODM Product Marketing
Stephen J. Lyons, STSM - Lead Architect WebSphere ODM Business Events
Jonathan Bond, Decision Server Requirements Manager (Events & Runtime)
Dave Wakeman, WebSphere POC Team

Mary Comianos, Publications Management
Stephen Smith, Technical Writer
International Technical Support Organization

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 papers to be as helpful as possible. Send us your comments about this paper 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 email 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>



Decision management overview

Decision management is a business discipline that empowers organizations to automate, optimize, and govern repeatable business decisions. It is an extension of the decisions business people would make if they had unlimited time to make those decisions.

Decision management technology provides the means for these decisions to be automated and called in real time by processes, applications, and other business solutions.

To realize this automation, the technology must also make it possible to capture, change, and govern the important decision logic in a controlled, scalable, and rapid manner. It is only by effectively managing the decision logic that the wanted business agility can be delivered.

1.1 Decision management technology

There are two interrelated forms of decision management technology (Figure 1-1).

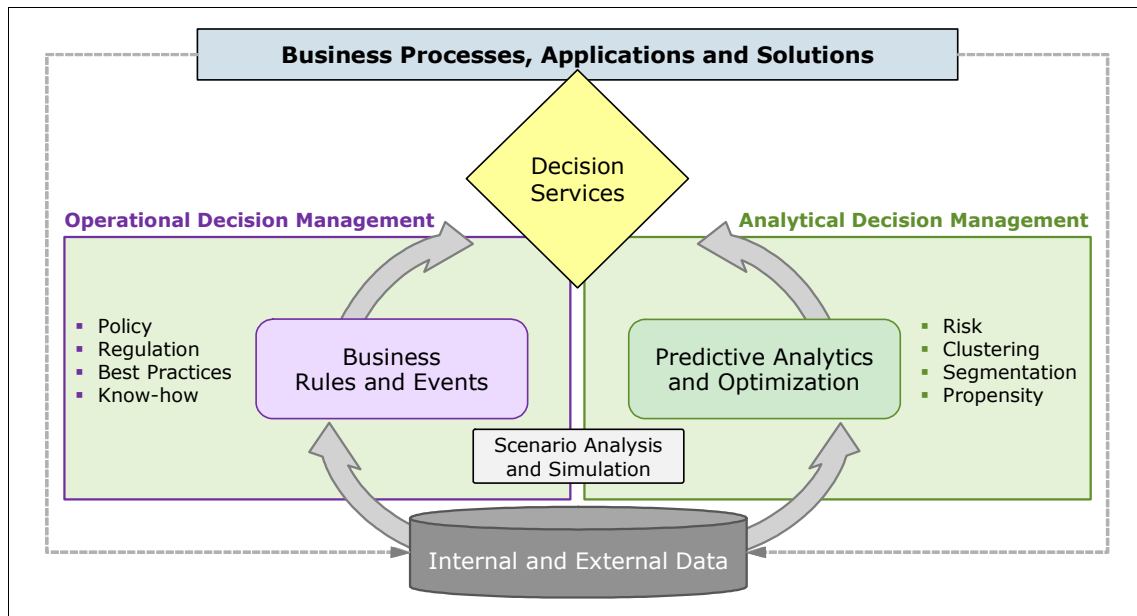


Figure 1-1 What is decision management

In both cases, the business processes and Applications make calls to the Decision Services that encapsulate the behavior of the decision being automated.

In *analytical decision management*, the history of decisions is analyzed to build a model that can be used to predict the best decision response for the future.

In *operational decision management*, policy, best practices, and business experience is used to write rules that describe how to make those decisions or identify situations that need to be reacted to. In many cases, the models that result from analytical decision management can also be used in the operational decision management automation.

There are situations where all the decision automation needed for a solution can initially be provided by one of these forms of decision management. However, as decision making become more complex, the need for synergistic use of both forms of technology becomes greater. When starting from analytical decision management, the models that define best practices or probability can be encapsulated and applied in a broader systems context through operational decision management. When starting from operational decision management, the decision logic that defines best practices can benefit from the data mining, segmentation, and insight provided by analytical decision management.

In both cases, there needs to be some measure of how good the decisions are. These *key performance indicators* (KPIs) relate to the overall goals of the business and, when used with Scenario Analysis and Simulation, allow for a real-world method of assessing how decision changes affect the behavior of business systems.

1.2 WebSphere Operational Decision Management

This paper focuses on WebSphere Operational Decision Management and its capabilities (Figure 1-2).

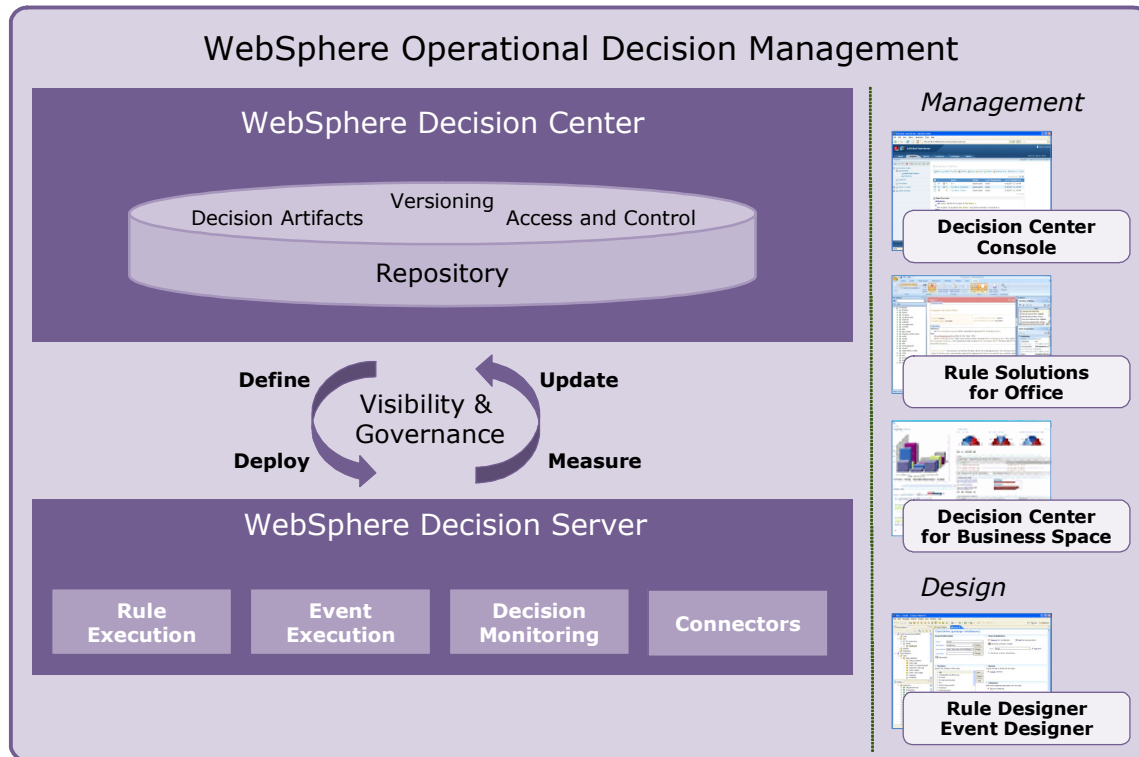


Figure 1-2 WebSphere Operational Decision Management components

IBM WebSphere Decision Center provides a convenient working environment for business people to collaborate, share, and manage their business policies in a secured environment. Operational decision projects are then stored in a centralized repository where releases can be saved as versions and accessed in a secure way. Decision Center also comes with various user interfaces:

- ▶ The web console providing a full set of management capabilities, including testing and simulations
- ▶ Plug-ins for maintaining rules using Microsoft Office documents
- ▶ Through widgets deployed in IBM Business Space

IBM WebSphere Decision Server is the environment designed for IT users to design, implement, integrate, prepare, and deploy operational business decision-based applications. The Decision Server integrated rules and events designers, both being specific Eclipse plug-ins, provide dedicated design perspectives for business rules and business events. Decision Server is also the execution environment for the business rules and business events policies.

This packaging means that WebSphere Decision Server provides a complete stand-alone capability for developers to design, deploy, and execute both business rules and business events in a SOA solution. WebSphere Decision Center provides capabilities to govern the changes to the policies or rules that define how those services behave.

1.3 Decision management solution development process

The goal of this paper is to provide guidance and best practices for realizing effective decision management solutions using WebSphere Operational Decision Management. Before going into the details of the example scenario and solution, it is necessary to consider the important steps that must be undertaken to define a solution.

The goal of decision management is to empower the business decision makers to rapidly understand and manage the way automated decisions are made within the solution. Decision management enhances their understanding, increases their throughput, and provides more precise fine-grained decisions that are appropriate to the situation.

However, the integration of decision management into the solution requires a deep understanding of the IT infrastructure and architecture. It is therefore important to understand the roles and tasks that are typically undertaken when developing a decision management solution. This topic is described in Chapter 2, “Development process overview” on page 9, which sets the scene for the more detailed design practices chapters in the rest of the paper.

1.4 Insurance demonstration scenario

In this paper, we use a fictional automobile insurance organization to demonstrate how the decision management capabilities of WebSphere ODM can be used in a realistic solution.

The automobile insurance organization (WODM Insurance) currently has two channels for marketing:

- ▶ A self-service website
- ▶ A call center

Using either channel, customers can apply for insurance quotes by providing the driver characteristics, vehicle characteristics, and type of coverage required.

Based on these characteristics, underwriting decisions are made as to the risk of providing insurance and, if approved, a price is calculated and a quote offer made to the customer.

WODM Insurance wants to automate the underwriting decisions and pricing rules to reduce risk, improve profitability, and ensure that they can react quickly to marketplace trends.

They are also concerned about the amount of market share they have and need to attract a higher proportion of profitable customers. They intend to accomplish this task by identifying situations where there is an opportunity to retain a desirable customer by offering personalized promotions.

WODM Insurance identified a number of situations where it is not possible to safely automate an underwriting decision. In this case, they require underwriter approval before offering a quote. They intend to use a business process to automate the approvals, ensuring that the underwriters can quickly decide which of these situations should be approved and which should be rejected. Automation of the approval process should also improve the turnaround time of the quote and improve customer satisfaction and retention.

1.5 WODM Insurance solution overview

The WODM Insurance solution illustrating the role of the WebSphere Operational Decision Management components is shown in Figure 1-3.

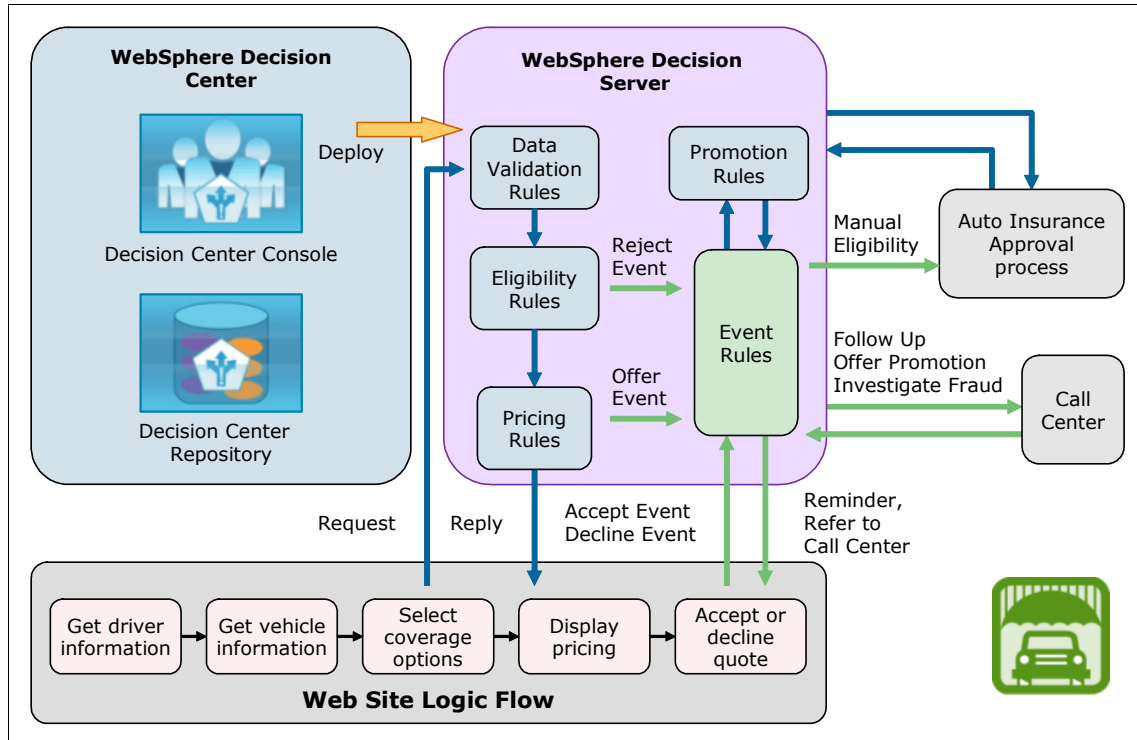


Figure 1-3 WODM Insurance solution

Each of the main components of the solution is now described, relating the component to later chapters in this paper where the design of that component is described in more detail.

1.6 Website: Eligibility and pricing decisions

The website is the main channel for customers to obtain quick quotations. On the website, the customer is asked for details about the driver, the vehicle, and the type of insurance coverage required.

The website application first validates the information provided and then starts WebSphere Decision Server to check the underwriting rules and assess if this driver is eligible for insurance. The Eligibility decision assesses the risk and identifies the quote as eligible, ineligible, or requiring manual approval.

If the decision is ineligible or manual, the quote is rejected and a reason for rejection passed to the customer. If the quote is eligible, the Pricing decision is made and a quote is offered to the customer.

The design of these decisions is described in Chapter 3, “Decision design” on page 21.

1.7 Website: Situational decisions

When a quote is offered to the customer, they have the option of either accepting or declining the quote. As with any web application, they can also close the browser and leave the site.

The website application also integrates with the WebSphere Decision Server Event run time. Events are generated whenever a quote is offered or rejected by Decision Server and also when a customer accepts or declines a quote.

These events are used in the Customer Acquisition Situations project (shown as Event Rules in Figure 1-3 on page 5) to determine customer behavior and determine what actions to take to close a sale. Two actions that influence the customer are possible through the website:

- ▶ Reminder Messages can be sent to the customer about quote offers, encouraging them to accept the quote.
- ▶ The customer can also be referred to the call center where they can obtain quotes that cannot be provided through the website.

In the situation when a manual eligibility approval is required, an action is raised to start the Approval process and the customer is referred to the call center to obtain the quote.

The design of stand-alone situational decisions is described in Chapter 4, “Situation identification and response development” on page 37. This design includes how to use Event rules to correlate and detect the responses from an individual customer across the website and call center.

If a promotional situation is identified, the Customer Acquisition Promotions (shown as Promotion rules in Figure 1-3 on page 5) decision is made to determine the most effective promotion. This decision uses business rules associated with events in a Detect - Decide - Respond pattern; it is described in Chapter 5, “Detect-Decide-Respond pattern design” on page 59.

1.8 Call center

The call center has an interface that is similar to the website, but it is used by a call center operator, who enters the customer details based on information received on the phone. After the details are provided, the Eligibility decision is made, and if the customer is eligible, the Pricing decision is made and a quote is obtained. The call center operator can also see the result of any promotion decisions and accept the quote on behalf of the customer or reject the quote. This situation illustrates the consistent reuse of the Eligibility and Pricing decisions described in Chapter 3, “Decision design” on page 21.

In addition, the call center maintains a list of referrals from the website with guidance about how to respond to those particular customers. These referrals include:

- ▶ Following up a customer that is unable to obtain a quote through the website.
- ▶ Providing a promotion to a desirable customer to close a sale.
- ▶ Alerting about potentially fraudulent quote applications where, for example, several quotes are requested for the same vehicle by different drivers in different states.

This capability uses the situation design principles described in Chapter 4, “Situation identification and response development” on page 37 and the Detect - Decide - Respond patterns described in Chapter 5, “Detect-Decide-Respond pattern design” on page 59.

1.9 Underwriting approval process

The underwriting approval process is implemented using IBM Business Process Manager (Figure 1-4).

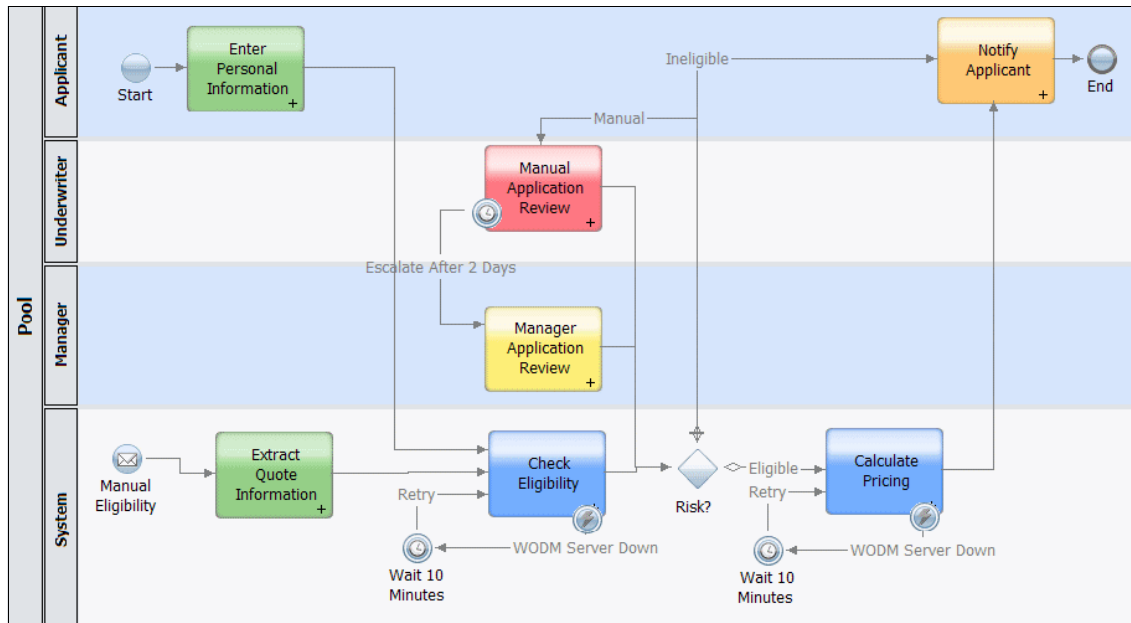


Figure 1-4 Underwriting approval process

The approval process can be initiated in two ways:

- ▶ The driver, vehicle, and coverage requirements can be implemented using a coach. This situation occurs when WODM Insurance receives an email or paper application form where the quote details must be manually entered into the system. This path through the process shows how WebSphere ODM business decisions can be used within a process to automate and improve consistency in the decision making.
- ▶ Alternatively, if a quote from the website needs manual approval, the quote characteristics already obtained from the website or call center can be retrieved and passed into the process.

As with the website and call center, the process first starts the Decision Server Eligibility decision. In the case of a manual Eligibility quote, the Eligibility decision is repeated to provide the rationale to the Underwriter, even though it is already made by the website.

The process flow gateway (called “Risk?” in Figure 1-4) uses the Eligibility decision response to route the process as follows:

- ▶ An eligible response routes the process to perform the Pricing decision and notification to the customer of a quote offer.
- ▶ An ineligible response routes the process to notify the customer of an unsuccessful application.
- ▶ A manual response routes the process to Manual Application Approval.

The Manual Application Approval provides a coach that displays the characteristics of the driver, vehicle, and cover to the underwriter together with the reason for the manual review. Based on this information, the underwriter can approve or reject the quote application together with a rationale for rejection.

If the underwriter tasked with the review does not respond quickly enough, the approval is escalated to the underwriter's manager, ensuring a fast turnaround in quote approvals. Based on the underwriter's response, the customer is notified of their unsuccessful application or the quote is priced and an offer is made to them.

The key design concepts of a business process and its integration with decision management are described in Chapter 6, "Business process design with rules and events" on page 85.

1.10 Decision management and governance

The final part of the WODM Insurance solution is to use Decision Center to provide the business with the capability to quickly change, simulate, validate, and deploy changes to the decisions and situations of interest.

This capability is achieved by making and managing the following decisions and situation projects:

- ▶ The Data Validation rules used by the website and call center to check the fields entered by the user.
- ▶ The Eligibility decision defining the acceptable underwriting conditions and those conditions that need manual approval.
- ▶ The Pricing decision defining the pricing rules for the different insurance cover, driver, and vehicle characteristics. This decision also shows how these pricing decisions might be managed at a regional level, allowing different regional managers to define the pricing rules needed for their regions while still complying with a global pricing policy.
- ▶ The Customer Acquisition Situations project that detects situations where customers might need promotions or other actions to obtain their business.
- ▶ The Customer Acquisition Promotions decision that determines a personalized promotion offer matched to each customer quote request.

The Decision lifecycle of authoring, simulating, validating, and deploying, together with the roles and governance features to support versioning and change management, are described in Chapter 7, "Managing decision changes across rules, events, and processes" on page 109.



Development process overview

This chapter presents the high-level steps needed to develop a solution involving a collection of business processes, with decision points implemented by business rules, and where situations are detected and acted upon through business events.

2.1 Roles and responsibilities

Service-oriented architectures (SOAs) and Business Agility promise organizations the ability to quickly adapt and optimize the way their IT solutions behave in response to changing market conditions and thus remain competitive in the marketplace. However, the responsibilities for defining and developing these solutions are spread across the organization.

The responsibility for specifying and managing the business must be considered from the point of view of different roles:

- ▶ Business Analysts are responsible for specifying how the business should behave, identifying key performance indicators (KPIs) that reflect how well the business is doing, and defining the processes and decision points needed to manage the business.
- ▶ Line of Business Users are responsible for the day to day management of the business using the solutions. They are responsible for monitoring the KPIs and modifying the way decisions are made to optimize the business. In a decision management solution, these roles have responsibility for optimizing decisions to meet the business need.
- ▶ End Users, such as call center operators (and to some extent, customers) are responsible for using the solution and need to be considered from a consumability and process efficiency perspective. In many cases, the End User role might be the subject of KPIs that the solution is designed to support, for example, customer satisfaction or call center response times.

The responsibility for the delivery and maintenance of these systems is usually an IT department. These IT solutions must be developed and tested in a rigorous manner to ensure that they are reliable, available, and meet the requirements of the business. Within this IT department, a number of key roles can be identified:

- ▶ Operational Roles have responsibility for maintaining the solutions and ensuring the availability and performance.
- ▶ Architectural Roles have responsibility for the design of the solution and ensuring that it meets the business requirements. This responsibility includes the requirements to be able to respond rapidly to certain changing market or business conditions.
- ▶ Development and Integration Roles have responsibility for configuring the products and building deployable solutions for operations to manage according to the architectural designs.

2.2 Decision development lifecycles

The service development lifecycle is often used by IT departments to undertake the delivery and maintenance of service-oriented solutions. While the details and products used to support this lifecycle vary considerably across organizations, the key activities undertaken by the IT roles are similar. The introduction of decision or policy management into these solutions allows an alternative lifecycle that interacts with the service development lifecycle. This alternative lifecycle provides the business roles with a faster means of changing the way decisions are made and thus react quickly to the emerging market changes.

These intersecting lifecycles are shown in Figure 2-1.

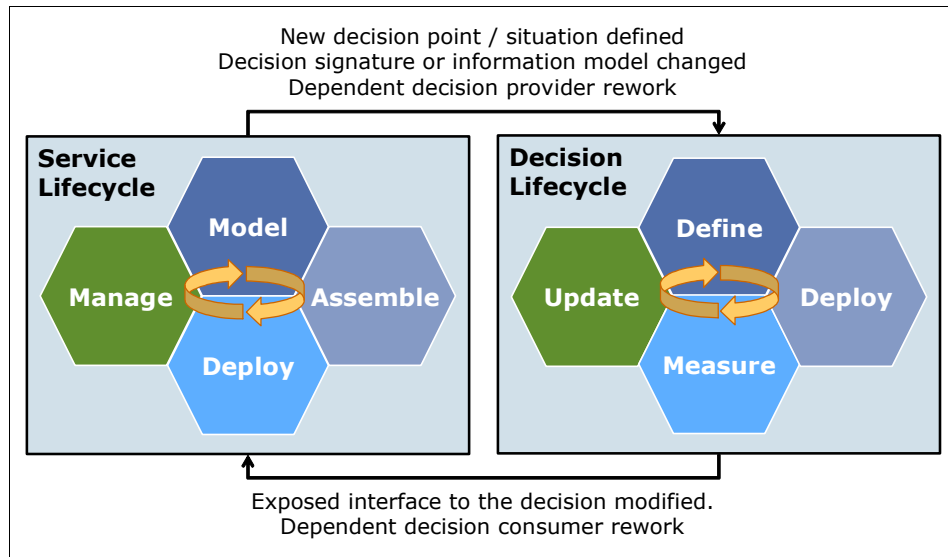


Figure 2-1 Intersecting lifecycles for service development and business decision management

When considering the design and development of solutions that use decision management capabilities, such as those provided in WebSphere ODM, consider both of these intersecting development lifecycles and the conditions under which changes in one lifecycle imply changes to artifacts managed in the other lifecycle.

The *service lifecycle* defines how decisions are exposed as decision services and integrated into the solution environment. This decision service definition then provides an initial definition of the decisions that can be managed by the business. Often, this initial design is undertaken by both Business and IT resulting in a set of deployed decision services and an initial set of rules that implement the decisions to be managed.

This initial service lifecycle is undertaken mostly by IT and is usually a relatively slow lifecycle to ensure that all the software quality testing is undertaken. Providing changes to or new decisions for the solution usually implies that the decision rules and vocabulary that realize that decision need to be revised.

The *decision lifecycle* defines how the business defines and manages the decisions that implement those decision services. After the decision service and associated vocabulary is defined, changes to the way the decision is made can be made faster (and ideally) by Line of Business Users. In this lifecycle, the Business Users can rapidly change the rules, simulate them, deploy them and thus deliver the business agility needed.

However, there are some cases when the changes to the decisions cannot be accommodated simply by changing the rules. A new decision might be needed or the vocabulary might need to be extended. In this case, a new decision service must be undertaken, resulting in a new version of the decision services being deployed and a new set of decisions available for the Line of Business Users to manage. This situation implies that the longer IT driven service lifecycle must be undertaken before those decision changes can be deployed.

Having summarized the interactions between these lifecycles, the phases and main activities in the lifecycles are described in the following sections.

2.2.1 Service lifecycle

The service lifecycle describes how the interfaces to the decisions are designed and exposed to the solution applications and processes. The emphasis is on how these decisions are exposed as services whose signatures are governed and managed as part of the software development lifecycle.

These interfaces represent the decision services and thus the identification of decisions, the input and output parameters, and the associated vocabulary. Relating these items to a centralized SOA Governance capability ensures that control over the purpose of these decisions and their role in the overall business solution is maintained. It also provides a common mechanism for discovering and sharing these decision services across the organization.

The versioning strategy for decisions in regard to the decision interface and endpoints available externally must be considered and aligned with any solution BPM governance and snapshot strategy. This situation ensures that the decision service definitions are used consistently in the business processes and solutions.

The phases and activities undertaken by the various roles are summarized here:

- **Model**

The Business Analyst and Architect work together to identify the processes and their activities, the decision points, and their role in the solution and the situations of interest to the business. This phase is then further decomposed to identify the business information needed to make any decisions together with how that decision is used in the solution.

The situation characteristics and initial responses are elaborated and the KPIs used to assess how well the decisions are performing. Simulation scenarios are defined based on the expected business requirements and the KPIs evaluated. These simulations can be used to optimize the information needed and the basic rules needed to meet the initial requirements.

- **Assemble**

The Architect works with Rule and Integration developers to develop the decision service implementations and integrate them into the solutions. This situation includes elaborating the vocabularies and business objects used in the rules and providing an initial set of rules to realize the decisions. The decisions and situations are integrated into the solution according to the decision service signatures agreed and the events and actions available in the solution.

This phase might be iterative until a satisfactory solution is achieved. Test cases are developed to ensure that the decisions and situation rules can be modified safely without breaking the integration and thus their role in the solution.

- **Deploy**

In this phase, the IT development organization hands over to the operational department and the processes, decisions, and situations are deployed as part of the solution. Although the development organization deployed the services to development environments, this deployment phase deploys to environments such as QA and Staging before deploying to the final production environment.

- **Manage**

In this phase, the processes, services, decisions, and situations are all operating as part of the solution and the KPIs agreed are being monitored by the LOB Users in dashboards. The LOB roles with responsibility for decision management examine the KPIs and the ways the decisions are performing to ensure that the wanted business goals are being met. If the business is not operating optimally, they use the Decision lifecycle to simply modify the way the decision is made. It is only when significant changes are required that they need to instruct IT to modify the decisions and perform another iteration of the decision service lifecycle. This iteration should be undertaken using the existing SOA Governance practices.

2.2.2 Decision lifecycle

The decision lifecycle describes how the business can change the way decisions are made, the way situations are identified, or the actions that are taken in response to situations being identified. The decision services and vocabulary is fixed within this lifecycle, as is the way each decision is used in the solution.

The business can extend the rules within the boundaries of the existing vocabulary and change the way the information and patterns are interpreted and responded to. As with the Decision Service lifecycle, a number of phases are described. These phases are summarized in the following list, and the activities are described in more detail in Chapter 7, “Managing decision changes across rules, events, and processes” on page 109:

- **Define**

This phase is undertaken between the Business Analyst and the Architect when the decision definition overlaps with the decision service lifecycle. In the case of a policy change or decision lifecycle iteration, the extent of the changes required need to be evaluated to determine whether the change can be accommodated simply by changing the decision.

This phase also includes simulation to ensure that the changes deliver the improvement in KPIs that are required. If the decision interface and vocabulary need to be modified, this action has an impact on the solution and another cycle of the service lifecycle is required.

- **Deploy**

This phase is undertaken by the Operational department and should include checking the validity of rules changed in the decisions and running the test suites to make sure that the new rules do not introduce any unpredicted behavior. After this validation is performed, the decision implementations are deployed through the various environments (for example, QA and Staging) to ensure validity before deploying to the Production environment.

It is important to point out that this deployment is for changes to the rules in response to the business and does not need to involve development. The initial version of the rules is deployed by development with the decision services and part of the decision service lifecycle.

- **Measure**

This phase is undertaken by the LOB Users observing the decisions made and the KPIs through dashboards and ensuring that the business is running optimally. Usage of the decision warehouse can help you understand what is actually happening in the business and identify those decisions that are working well and those decisions that need to be improved.

- Update

This phase is undertaken by the Business Analyst or LOB User who needs to have sufficient understanding of the way the decisions are made to be able to modify them to improve the performance. The output of this phase is a set of current KPI measurements and target KPI goals for the next iteration, version, or baseline of the decision lifecycle.

Further details of the decision lifecycle are described in Chapter 7, “Managing decision changes across rules, events, and processes” on page 109.

The remainder of this chapter describes key aspects of the decision service lifecycle that need to be considered to realize decisions.

2.3 Process development

In many cases, the solution design can be obtained by an analysis of the current activities and processes being undertaken by the business. This analysis leads to identification of decision points, situations of interest (exceptions and escalations), and other important pieces of information. Process analysis is not the subject of this paper, but considerations for integrating decision management into business processes are described in Chapter 6, “Business process design with rules and events” on page 85, including:

- Making decisions from within processes
- Initiating a process as a result of an action or in response to a situation
- Raising an event to be used in the assessment of a situation across processes or systems

2.4 Decision point development

Identification of key decision points across the solution is crucial to the whole decision development. This topic is described in Chapter 3, “Decision design” on page 21 and is applicable to decision use across solutions, processes, or situations (events). A key part of this development is the identification of the information used in the decisions. This analysis is similar but different for processes, decisions, and situations, so a comparison of the information modeling is described in 2.6, “Decision information models and vocabulary” on page 15.

2.5 Situation identification and response development

The activities are still emerging for the identification of situations and how to respond to them. There are many similarities with identification of decisions, so we do not define the situation development activities in detail. Instead, we identify some of the key integration patterns for situation identification and response in Chapter 4, “Situation identification and response development” on page 37 and then describe how to combine situations and decisions into a Detect - Decide - Respond paradigm in Chapter 5, “Detect-Decide-Respond pattern design” on page 59.

2.6 Decision information models and vocabulary

Before any form of decision making can be undertaken, the information about which the decision is to be based must be identified. This identification is an iterative process and involves a good understanding of the solution and the information being processed by that solution.

A starting point is to identify the concepts or types of object that are processed by the solution. In the WODM Insurance solution, the concepts of Driver, Vehicle, and Coverage Request can immediately be seen as useful concepts. Each of these object types has a set of characteristics that is used by the solution or decision making, although different parts of the solution might have different perspectives of these concepts and require different characteristics to be defined, such as:

- ▶ The solution needs a driver's address to send out quote policies
- ▶ The decision making component is interested in a driver's age
- ▶ The event processing component is interested in other quotes that the driver requested
- ▶ The approval process needs to know a driver's identifier to ensure that approvals are applied to the correct quotes.

It is therefore important to identify a common business terminology that can be used across the solution, Rules, Events, and Process tools. This terminology includes the naming, verbalization, and characterization of decisions, situations, and actions, and aligning business object types across rules events and processes. This situation does not imply that a single representation must be used for these information models. However, you have better interoperability and achieve a common understanding between business and IT (and the different tools that support them) if you follow the recommended practices described in this section.

2.6.1 Business object definitions

This section describes the characteristics of business objects and how they are represented across the Rules, Events, and Processes. When identifying the information needed for these decisions, it is the business objects and the fields that describe them that are important. This capability is available across the different tools, but the manner in which it is implemented is different.

Table 2-1 shows the different characteristics of the information model and how it may be represented in the different tools.

Table 2-1 Information model characteristics

Characteristic	Rules	Events	PD	IBM Integration Designer / Schema
Class representing a type of object	Business Class with natural language verbalizations and descriptions	Only available as shared objects currently	Data -> Business Object representing a type of business object	xsd:ComplexType

Characteristic	Rules	Events	PD	IBM Integration Designer / Schema
Object or instance of a class representing a named concept of relevance to the business	Variable or Decision parameter of type Business Class	A Business Object	Variable	xsd:Element
Fields	Attributes referencing simple types or Business Classes	Field available in Business Objects. Can reference only simple types	Parameters referencing simple types or Business Objects	xsd:Element
Scope allowing the same names to be resolved within a single solution	Package	Package	None	namespace/prefix
Name (nls) - allowing the business to refer to the characteristic in a business like form	Verbalization of classes, variables, and attributes	Verbalization of business objects and their fields		No
Description - allowing the semantic meaning of the characteristic to be understood	Available for Classes and attributes and can be defined for multiple languages	Available for objects and fields in a single language	Available for business objects, and parameters in the language of definition	Available through the use of annotations
Field cardinality defining how collections of objects or primitives can be handled	Field cardinality constraints and verbalization for handling collections of field values	Single cardinality at a field level although objects can be collections	Field cardinality supported	Complex type cardinality constraints on field elements

Although it might be convenient to have a single representation, it is important to recognize that the different types of processing actually require different sorts of information models.

Decisions usually use rich information models based on deep hierarchies of classes or types of objects meant to provide a detailed factual context for the decision. The more refined the decision is, the more information it requires to be established. Decisions are provided with the full context that they need to render the decision and do not retrieve additional information after they are started (for a more detailed explanation, see 3.6, “Decision integration” on page 33). The decision response can also require a complex model. For example, a complex eligibility decision might return a list of alternative coverage quotes, potentially upselling the prospect with additional coverage, or other types of insurance, such as home, life, and so on.

Events consider patterns of objects and usually need an array of objects of the same structure. The fields are a subset that is used to determine the pattern, which means that they should not be complex types (classes) or collections of values. Key fields should be added that either:

- ▶ Allow the objects to be correlated (for example, a Vehicle Identification Number allowing different quotes for the same vehicle to be associated)
- ▶ Allow other processing to augment the concept fields with information not provided in the event (for example, having a quote ID that can retrieve information from a database).

Often, the keys from different objects need to be brought together to identify the patterns, so a quote might include a driver's full name, a Vehicle Identification Number, and quote ID fields which allows identification to be based on different contexts. The nature of event processing is that many objects must be held in memory at the same time, so wherever possible, the size of the objects (number of fields) should be minimized.

Processes tend to have little emphasis on information, but focus on the sequence of activities to be performed. Information that is added by those activities may be stored in databases or in some cases stored in process state variables. Similarly, information presented to users participating in activities are retrieved from databases or other services based on some key process state variables.

For example, when processing a quote approval, it might be adequate to simply maintain the quote ID of the quote. When an activity is undertaken, the subset of fields appropriate to that activity can be obtained from the database, the activity undertaken, and the results stored back to the database.

When integrating systems using this approach, the requirement is then about mapping or augmenting the fields that are required from the fields that are provided. This capability can easily be provided by ESB mediations or is often provided as a core capability of individual products. In WebSphere ODM event processing, event object fields can be mapped to business object fields directly or by using JavaScript functions. Process Designer web service integrations have mapping tables that allow the web service parameters to be mapped into process variables for the duration of an activity.

These different perspectives mean that attempts to unify the information model often result in failure. The approaches often used within SOA start with a data dictionary that defines the concepts and ways that they are expressed. They also describe the fields or characteristics to be associated with each concept. This situation does not mandate that every representation uses every field, but it does provide a basis for using common terminology when creating the system-specific concepts.

2.6.2 Business object field primitives

Software technologies often provide an extensive number of ways of representing information, especially the simple fields. In practice, there are a few primitive types that can support most of the business requirements.

Table 2-2 describes the preferred primitive types in use and the recommended subset to avoid the extensive mapping required for the individual fields.

Table 2-2 Information model field primitives

Type	Rules	Events	PD	IBM Integration Designer / Schema
String	String	String	String	xsd:string
Integer	Integer	Integer	Integer	xsd:int
Real	Double	Double	Decimal	xsd:double
Boolean	Boolean	Boolean	Boolean	xsd:boolean
Date	Date	DateTime	Date	xsd:date
Time	Time	DateTime	Time	xsd:dateTime

By using these field primitives, mapping of field values can often be undertaken automatically by the products. This situation means that the mapping can concentrate on selecting the fields that need to be included in the particular representation.

2.6.3 Domains and enumerations

Many information models can be built on the class and field constructs described earlier. However, many information models require that only a limited range of values be used in a field. This set of values is often termed an enumeration or *domain*. In the WODM Insurance scenario, domains are used to represent many different fields, including the type of vehicle (Sedan, Coupe, Pickup, and so on) or the type of coverage (comprehensive, liability, and so on). Although the values in these domains might be fairly static, other domains might be much more likely to change. Examples of this change could include vehicle makes and models or even product classification systems.

These domain values usually have representations used in the solution and definitions for what the value actually means. This meaning is important when making decisions, so individual values often have a verbalization, allowing rules to compare the field value to detect the condition intended by the rule author.

Domains normally reflect a range of possible values that may be used in the solution, which usually has checks to ensure that only the values in the domain are handled correctly. As new domains values are added, the solution provides the means to make sure that the solution can handle those values.

Decisions that are made based on those values now need to be modified to reflect the responses for the new values. This management of domain updates in decision management is an important feature that needs to be considered in the information model. WebSphere ODM does support various techniques for updating domain values, but it is often difficult to maintain domain values across different system boundaries.

2.6.4 Variables

In the previous sections, we described the general structure of an information model and how the characteristics of its concepts can be described. This description does not provide information about the context or role of the concept in the solution or decision being made.

This context is defined by giving a concept a name or by defining a variable that describes that particular concept.

In the WODM Insurance solution, a quote could refer to multiple drivers. The context must distinguish between:

- ▶ The driver applying for the quote
- ▶ The main driver for a vehicle
- ▶ The set of drivers for a vehicle

Providing named variables similar to these items makes the context far clearer when writing rules for decisions.

Variables are used in a similar manner for correlating the patterns of events. In this case, a variable holds a sequence of objects (of the same type, such as Quote) associated with Events from the same context (for example, the main Driver full name).

When considering the analysis of decisions and the solution business performance, it is important to define KPIs. These KPIs must have a context that defines what they are a measure of, so they are examples of the use of variables in the information model.

Putting these recommendations together:

- ▶ Variables should be used to identify concepts of interest in the solution or decision making, such as the main driver.
- ▶ The variables should represent a certain type of object, such as a driver.
- ▶ The fields associated with the type of object should be defined in a data dictionary to ensure consistent interpretation.
- ▶ Each system representation of a variable should use the fields needed.
- ▶ System variable fields should avoid complex types and arrays where possible.
- ▶ Care should be taken when considering the use of domains for field values to ensure that the values are interpreted correctly across the systems and can be extended when the domains are updated.



Decision design

This chapter describes how to use the business rule modeling and execution capabilities of WebSphere ODM to build complex decision services. These decision services look at the context of information submitted by an application and determine what should be the response to that context according to an established business policy.

The approach is to create a model of the business context that gathers all of the necessary pieces of information that are needed to:

- ▶ Make the decision.
- ▶ Decompose the overall decision process into a set or organized substeps.
- ▶ Define each substep as a set of business rules that, when applicable in the context of information, enriches that context with new information that contributes to the final decision.

In WebSphere ODM, *decisions* are implemented by rule sets that are deployed to the Rule Execution Server. *Decision services* can then be constructed that request the execution of the decision on the Rule Execution Server.

3.1 Identifying business decisions

Decisions that can be implemented through business rules emerge from the application requirements or the business process maps at the time labels or high-level descriptions are applied to the different activities of the solution. Descriptions that use an action verb, such as *determine*, *check*, *calculate*, *evaluate*, *decide*, and so on, that are applied to business objects typically indicate the potential for a decision point.

Examples from the WODM Insurance scenario are the *Validate quote request data*, *Determine driver eligibility*, and *Compute quote pricing* descriptions that appear in the high-level design of either the Insurance website or the call center application.

Rule-based decisions are made using a stateless context of information. This context of information should be complete before the decision is made and represents a snapshot of a business situation at a certain time. This decision contrasts with event-based decisions that yield their outcome from observing the evolution of context-based information over time.

Decisions interface with the outside world through a set of input and output parameters. Decisions are invoked synchronously, with the caller waiting for the decision outcome before proceeding (for more information, see 3.6, “Decision integration” on page 33). You are free to process requests asynchronously (for example, using a message driven bean) to efficiently perform tasks such as batch processing.

The outcome of a decision is a combination of flags and computed values, which result in a set of actions taken on the side of the decision maker:

- ▶ A flag returned by a decision influences the flow of the application code or the business process. In the latter case, the activity that makes the decision is directly followed by a gateway in the process map. The Data Validation and Eligibility determination fall in this category (see the Check Eligibility activity in Figure 3-1).
- ▶ A value computed by a decision is assigned to an attribute of one of the application or process business objects. In the context of a business process, a decision that computes a value may not be followed by a gateway, but the returned value directly participates in the result of the business transaction. The Pricing determination of the WODM Insurance scenario falls in this category (see the Calculate Pricing activity in Figure 3-1).

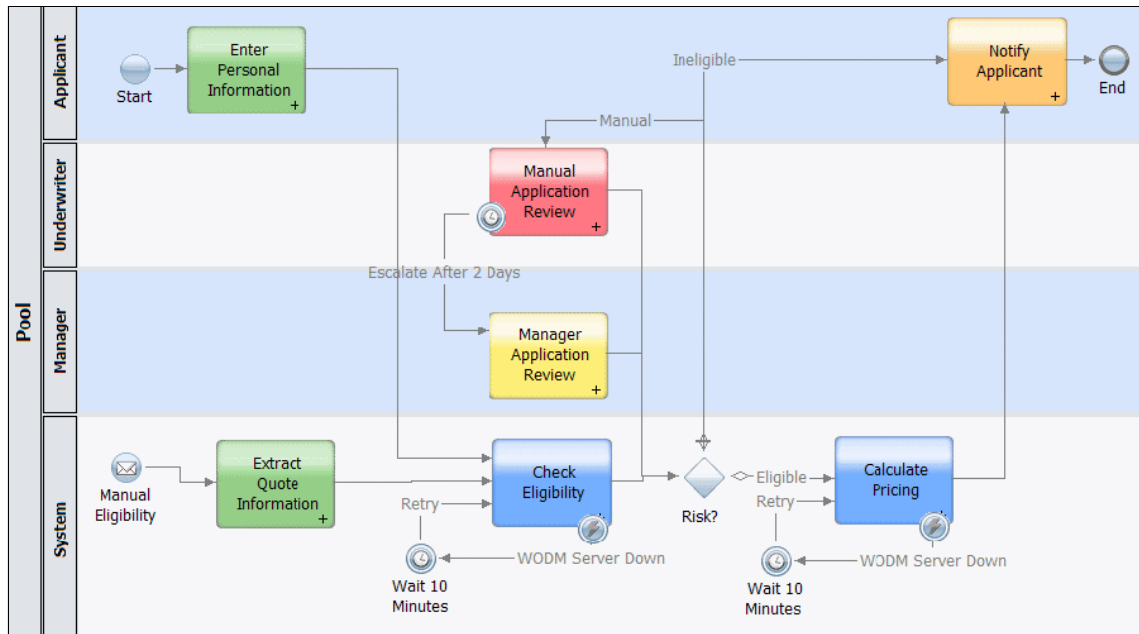


Figure 3-1 Decision points in the business process

Decisions commonly involve knowledge that belongs to business subject matter experts (SMEs). Looking for human tasks that are followed by a gateway in a business process is a good way to identify decision points. These decisions can be split into a straight-through processing component that defers to business rules, and an exception component that handles complex cases involving human expertise. As the decision point is better understood and the set of rules is extended, the proportion of cases addressed by the rules versus the one that should be addressed manually is growing.

Decisions are characterized by:

- ▶ Their complexity: A valuable business decision represents a complex business operation. This operation commonly involving hundreds of rules organized in a number of functional subtasks that might involve the participation of several business departments and cover multiple business dimensions, such as geographies, channels, products, and so on.

- ▶ The large amount of context data they need to execute:
 - As a corollary of the decision complexity and because it needs to make fine-grained inferences based on many business dimensions (the channels, geography, and so on), the decision rules need extensive information about the business situation it is examining. The more context information about the borrower's credit history or the line items of the insurance claim is provided to the decision maker, the more accurate and powerful the decision can be.
 - Reference data: Decisions involving the validation of a business artifact often need a catalog of prototypes and constraints to perform the validation process (for example, the list of phones or calling plan available for a zip code).
- ▶ The volatility of their definition: The definition of business policies changes often, motivated by competitive or regulatory factors coming from outside the enterprise, or by the needs to address new business situations (extend the decision footprint) or existing situations (improve the decision quality).

Based on the first two characteristics, the business rules components embedded in BPM platforms are not suited to define and manage complex business decisions. The data that is passed through the activities of a business process is not detailed enough to make complex decisions.

Also, externalizing decisions to a decision management system allows users to:

- ▶ Agree on common terminology and vocabularies that can be used for decision making across applications and business processes. This situation makes the decision rules understandable and accessible across the organization.
- ▶ Make the decisions reusable across applications and business processes. As more business processes are automated and applications are modernized, common decision services are identified and can be shared through a decision management system. This situation is the case in the Web Quote application, where the Eligibility and Pricing operations are shared by the website and the call center. Decisions are commonly encapsulated and exposed as web services, which allows easy reusability and integration.
- ▶ Clearly decouple the lifecycle and the governance of the decisions from the lifecycle of the application or business process. Indeed, the motivations and the stakeholders that drive changes in the decisions are different from the aims of processes. Also, decision changes (business policy, market shifts, new regulatory requirements, and so on) tend to be more frequent than application or process changes.

During the decision point identification process, although the details of the data that are needed as input to the decision are often unclear and incomplete, the outcome of the decision should be well-defined, as it directly influences the continuation of the application or business process execution. The precise input needed by the decision is determined while harvesting the business rules and building the business object model, as described in 3.2, "Defining the domain of discourse for the rules" on page 24.

3.2 Defining the domain of discourse for the rules

After the set of decision points needed by the solution is decided, the Rule Projects that provide the implementation of the different decisions can be created. The starting point for the development of a Rule Project is to identify the conceptual object model that supports the representation of the context data for the decision and also captures the decision response.

The discovery and organization of the conceptual object model is an activity that is concomitant with the harvesting of some initial business rules that apply to it. Harvesting rules prompts the definition of the business terms and facts that are needed to structure the object model. Conversely, structuring the terms and facts in an object model has an impact on how the rules are eventually written.

It is a best practice to define the conceptual object model as a UML class model. This action facilitates the iterative design that is bound to occur during the early phases of the model construction, and also makes the design task more approachable and manageable by a business analyst.

In the WODM Insurance application, the three decisions rely on a common object model because all the business rules revolve around the context of an AutoQuoteRequest. The model is organized hierarchically, based on the concepts illustrated in Figure 3-2.

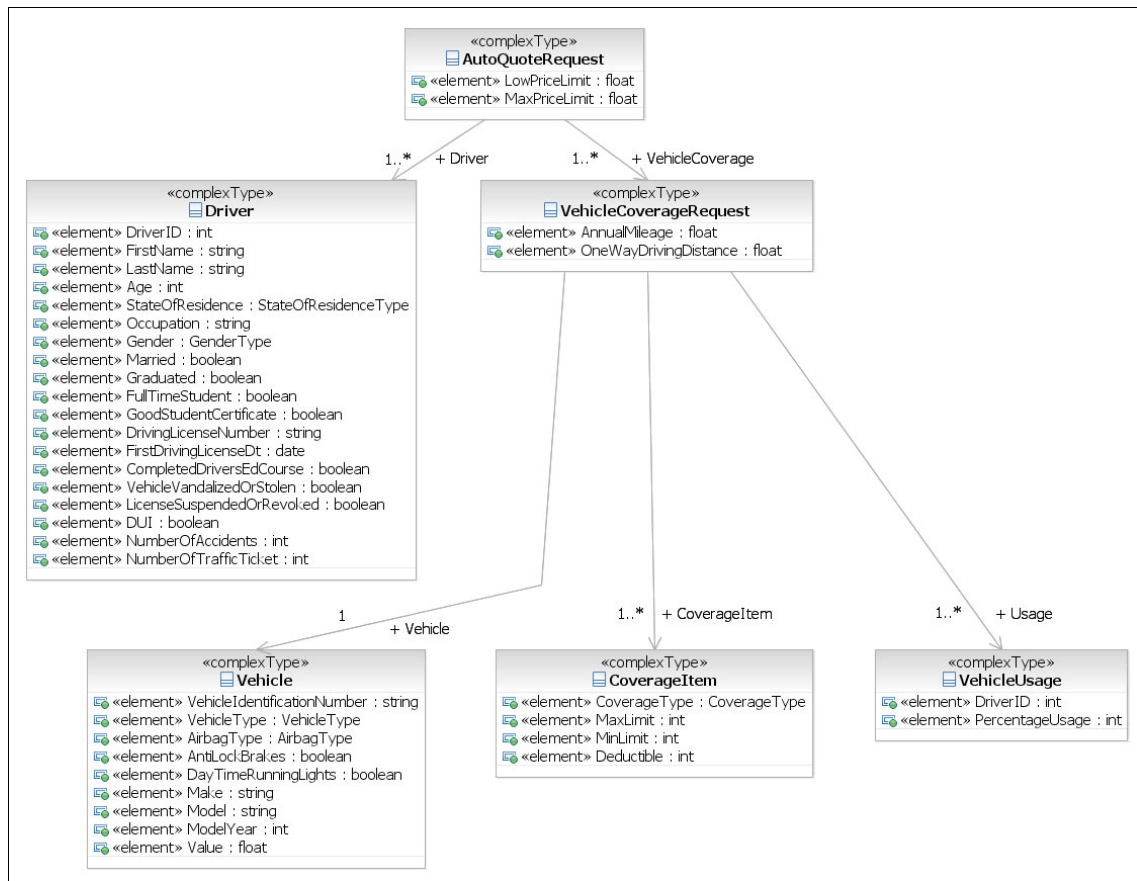


Figure 3-2 AutoQuoteRequest model

The conceptual object model should be designed to accommodate two (often conflicting) characteristics: Cover all the details needed to make the decision while staying understandable to the business users. Therefore, the model can end up being large and complex, and it must be instrumented and adapted for the business users.

In most cases, the object model acted upon by a rule-based decision is much larger and demands more details than the business objects that are managed by the business process or the events.

Conversely, the object model is much less complex than industry standard models such as ACORD and MISMO, which tend to be all encompassing and are better suited to exchange information across systems or enterprise boundaries.

3.2.1 Tailoring the model for the business users

After a first workable version of the conceptual model is available, it can be transformed into an eXecution Object Model (XOM), which is a set of Java classes or an XML Schema. If a UML tool is used to design the conceptual object model, this transformation is straightforward. The XOM represents the model of the data that is passed to the decision when it is executed. The rules are applied to this model.

To make the rule writing activity intuitive for the Business Users, a Business Object Model (BOM) is built on top of the XOM. The BOM allows you to associate verbalizations with classes, attributes, and methods from the XOM. It also allows you to mask elements from the XOM that are technical and are not directly relevant to rule authoring, and it allows you to add some business terms to the BOM that are derived from the combination of attributes and operations from the XOM.

Using in Rule Designer, A BOM in a Rule Project is defined by someone in an IT role or a business analyst with enough technical background and experience in object-oriented modeling.

In the AutoQuoteResponse BOM class example shown in Figure 3-3, the GlobalAdjustmentsList and the MessagesList attributes come from the associated AutoQuoteResponse XSD complex type, which carries back the outcome of the response to the decision.

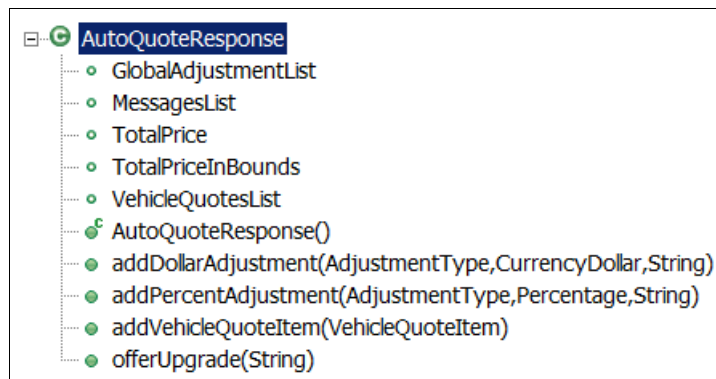


Figure 3-3 AutoQuoteResponse BOM class structure

However, these attributes are not directly relevant to the business person who writes the pricing rule. The business person wants to be able to “add a dollar adjustment” or “add a percent adjustment” to the quote while adding a justification for doing so.

The BOM editor of Rule Designer empowers you to:

- ▶ Make placeholders that are not directly relevant to the business (for example, GlobalAdjustmentList and MessageList) invisible by not verbalizing them (Figure 3-4). Attributes that do not need to be used in rule project artifacts should not be part of the BOM. Attributes that are in the BOM, but not verbalized, can be used in technical artifacts, such as functions or technical rules, which are written against the structure of the BOM classes.
- ▶ Create business level actions (addDollarAdjustment and addPercentAdjustment) that use underlying XOM class placeholders (Figure 3-5).

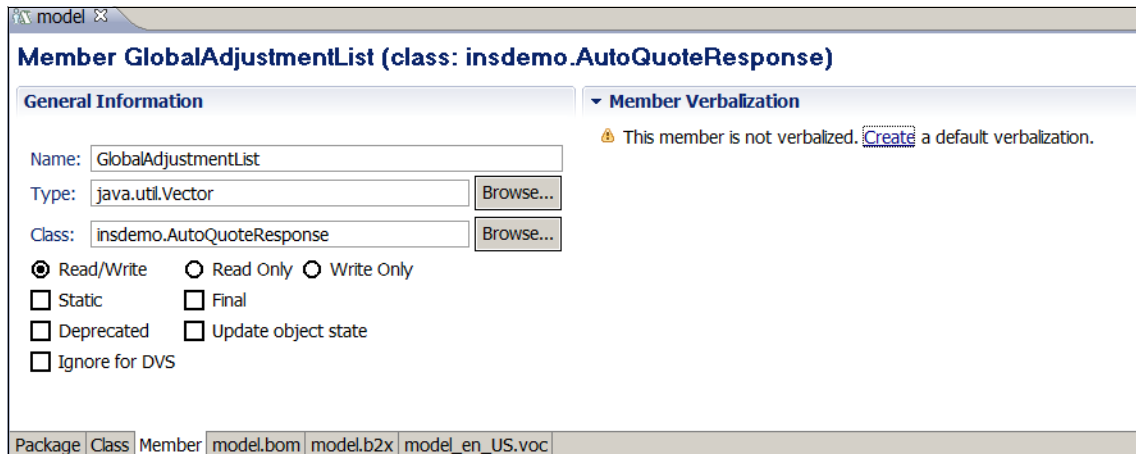


Figure 3-4 Not verbalizing attributes that are not needed by the business rule authors

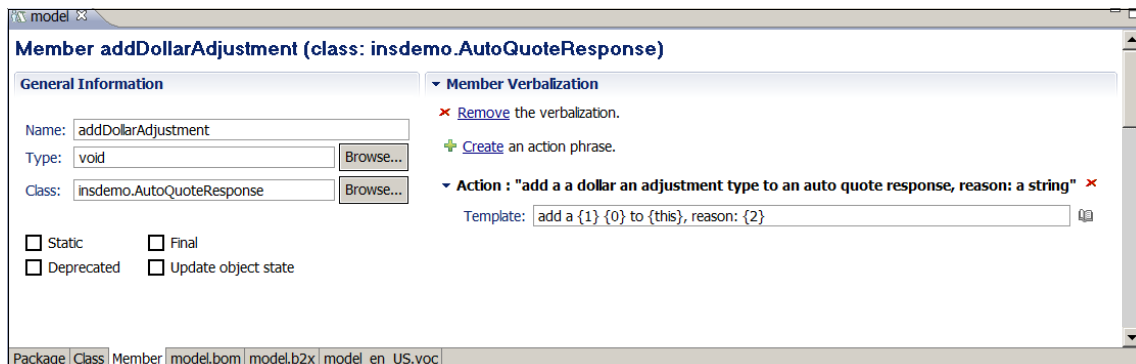


Figure 3-5 Creating business level operations in the BOM

Deep hierarchies of classes in the XOM may lead to complex expressions being needed in business rules. One role of the BOM is to simplify terminology. For example, a driver could have an Address type field with a State field. This situation means that rules that make decisions based on the state of residence would need to be expressed as:

If the state of the address of the driver is "CA" then...

This situation can be confusing, so in the BOM representation of the driver, you could include a state of residence virtual field that leads to more understandable rules, such as:

If the driver resides in "CA" then...

3.2.2 Sharing a common model across decisions

Because all the decisions in the WODM Insurance application are using the same BOM, a common rule project named `AutoInsuranceQuotingBOM` is dedicated to the BOM definition, and does not contain any rules. See Figure 3-6.

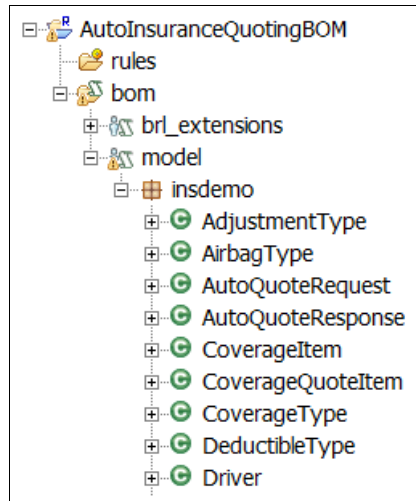


Figure 3-6 Common insurance BOM isolated in a rule project

The three rule projects that correspond to our three decisions reference this BOM project so that they can use it in the definition of the rule artifacts. Figure 3-7 shows the reference from the Eligibility rule project to the `AutoInsuranceQuotingBOM` project.

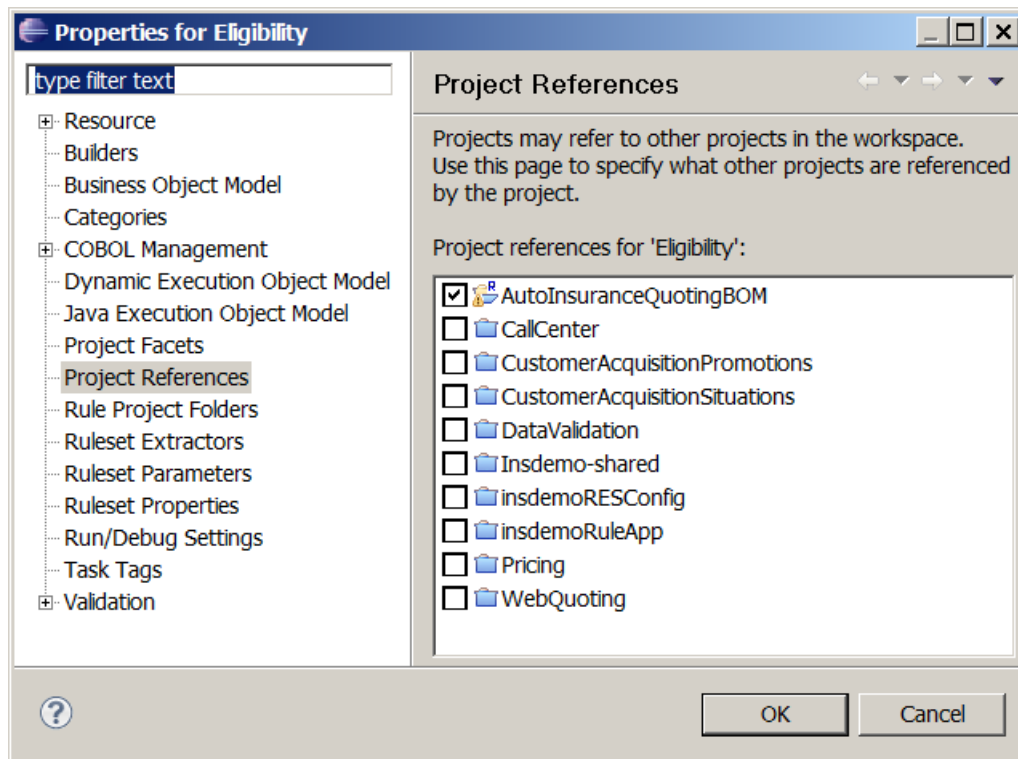


Figure 3-7 Sharing the common BOM through a project reference

The project organization is shown in Figure 3-8.

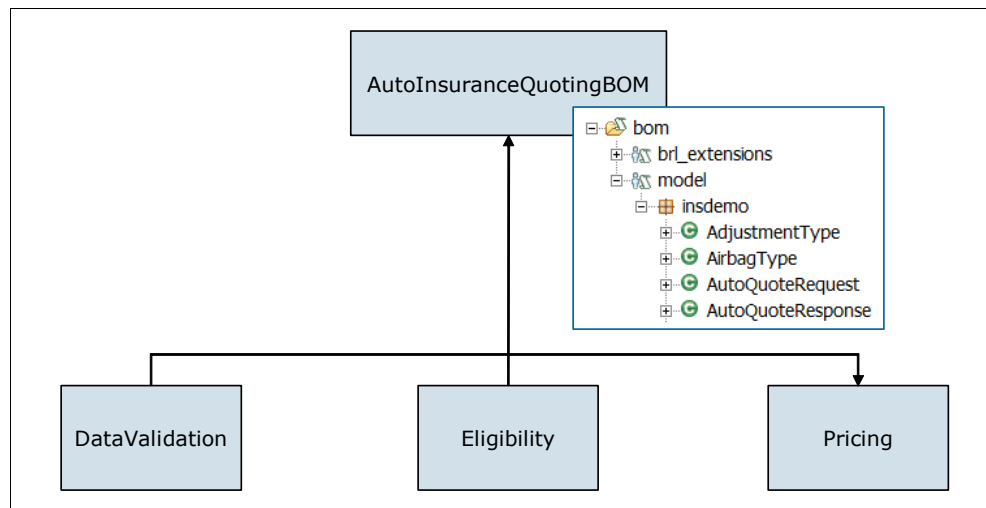


Figure 3-8 Shared BOM

In the WODM Insurance application, the common BOM includes the model for the outcome of the three decisions. The class `ValidationResponse` is associated with the `DataValidation` decision, the class `EligibilityResponse` with `Eligibility`, and the class `AutoQuoteResponse` with `Pricing`.

The BOM can be composed of multiple BOM entries, and you could have:

- ▶ Excluded the classes `ValidationResponse`, `EligibilityResponse`, and `AutoQuoteResponse` from the BOM defined by the `AutoInsuranceQuotingBOM` project.
- ▶ Defined a BOM entry in each of the decision project, containing only the response class relevant to that decision.

The goal of this practice is obviously to place exactly the set of concept and operations where they belong and the benefit is to not bother the business user with concepts and operations that are irrelevant to a specific decision.

The resulting design is shown in Figure 3-9.

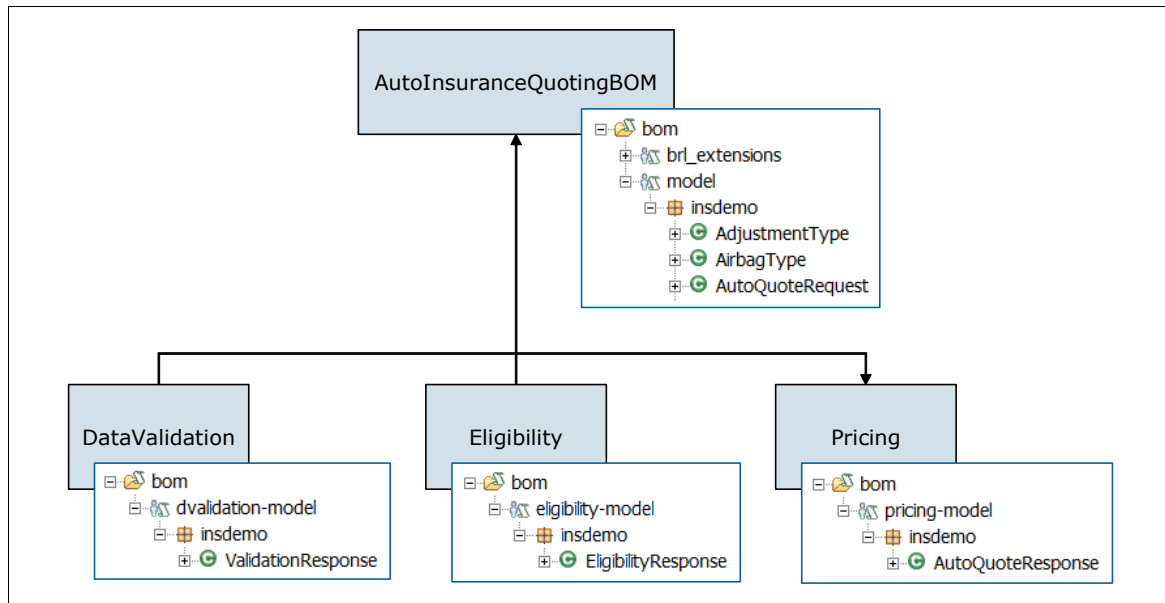


Figure 3-9 Shared BOM with local BOM entries extensions

3.3 Establishing the decision signature

After the Business Object Model is defined, the decision signature is established by defining the interface of the decision to the outside world. That is, what data does the decision need as input and what data it returns as the output of the decision. This interface is defined by the ruleset parameters associated with the rule project (Figure 3-10).

The type of the ruleset parameters is drawn from the set of classes defined in the BOM. Therefore, part of the XOM and BOM design activities is to plan for classes that are used as input and output parameters.

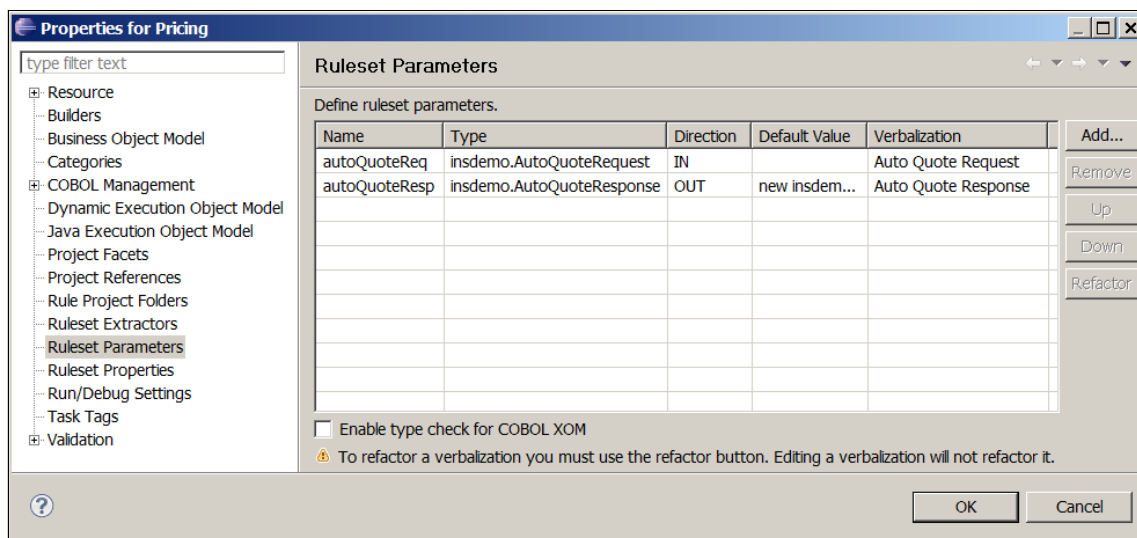


Figure 3-10 Pricing project ruleset parameters

As described in 3.1, “Identifying business decisions” on page 22, the outcome of a decision is usually a collection of flags or computed values, often completed by a collection of messages that gives insight into the reasons for the decision. It is then the responsibility of the invoker to process the flags and use the computed values as needed.

For example, the `DataValidation` and `Eligibility` decisions focus is on determining a flag value, which is reflected in their output parameters, the `Validated` and the `Eligible` flags. See Figure 3-11 and Figure 3-12.

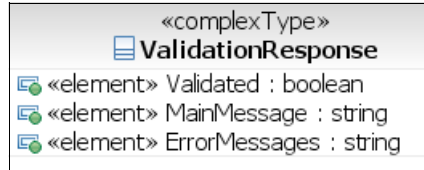


Figure 3-11 *DataValidation* output parameter

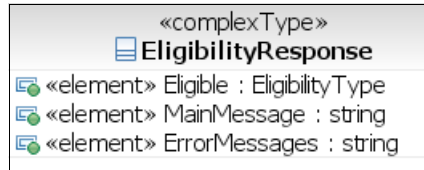


Figure 3-12 *Eligibility* output parameter

3.4 Decomposing the decision and orchestrating the subtasks

A complex decision is composed of multiple substeps. So, before you create individual rules, you need to model the decomposition of the decision into substeps and organize the sequencing of the substeps through a flow.

Each substep is usually a group of rules that revolve around a common business criterion. For example, the `Eligibility` decision performs a group of validations based on the driver’s age, one based on the driver’s perceived risk, and one based on the concept of profile, which is the aggregation of multiple characteristics of the driver. The resulting groups are defined as rule packages (Figure 3-13).

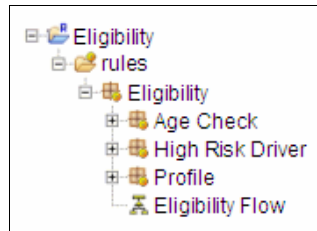


Figure 3-13 *Rule packages decomposition*

The next step is to place the functional groups of rules into a sequence using a rule flow.

For decisions that perform a validation such as Eligibility, the common rule flow pattern (Figure 3-14) is a cascade of rule tasks associated with a rule package, in which, after each task, the value of the output flag is tested. If the task detects an invalid state, the execution terminates; otherwise, it continues to the next group of validations.

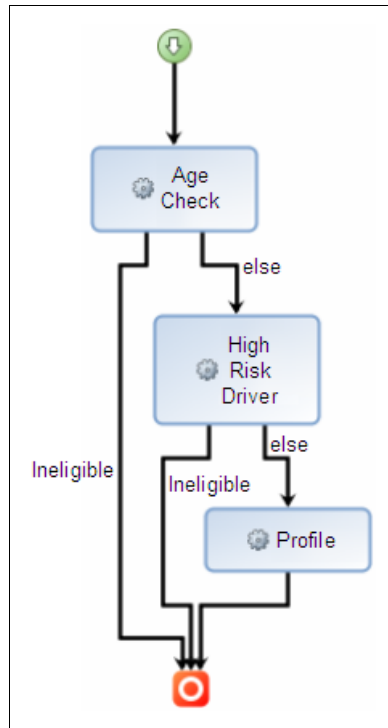


Figure 3-14 Eligibility rule flow

For decisions that perform a computation, such as Pricing, the rule flow is decomposed into a set of tasks that represent the different steps of the computation (Base Premium, Local Adjustments, and Global Adjustments for Pricing). There is also some branching that determines which of the steps are needed for the request context.

3.5 Authoring business rules

With the decomposition and orchestration of the different steps of the decision in place, the individual business rules can now be specified in the rule package they belong to. Rule authoring is performed by the Business users using the Decision Center.

The business rules are written to recognize a pattern in the context of information provided to the decision. When the pattern is recognized by the rule conditions, a set of actions is applied by the rule.

Although the actions can correspond to an arbitrary snippet of code, they usually assign or update the value of an attribute that contributes to the decision response. The goal of the Eligibility decision, for example, decides whether the insurance application should be accepted or rejected. By default, the application's eligibility flag is set to "Eligible" and the goal of the rules is to recognize the conditions under which the application is "Ineligible" (or should go through a manual review process) and mark it as such.

The simplest metaphor to express a rule is an Action Rule, which is a simple if-then entity. Figure 3-15 shows an example of such as rule in the context of the Eligibility decision, which rejects the application for a driver who is too young. This rule belongs to the Age Check rule task defined in the rule flow.

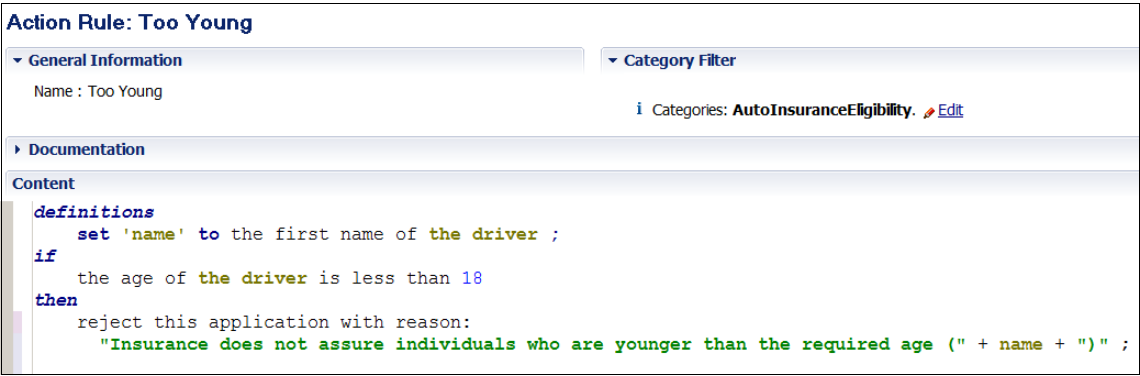


Figure 3-15 Action rule definition

The rule action is designed to set the eligibility flag of the application to Ineligible and record an explanation message. Its verbalization, defined through the Business Object Model, makes its intent clear to the Business user.

When multiple rules share a template of conditions and actions, they can be grouped in a more concise metaphor, which is the *decision table*, such as the one shown in Figure 3-16. This decision table belongs to the Profile rule task defined in the rule flow.

Gender	Age		State of Residence	Set Eligibility	
				Eligible?	Message
female	<div><div>< 16</div><div>[1625[</div><div>≥ 25</div></div>	<div><div>CA</div><div>Otherwise</div></div>	Ineligible	Sorry, you are too young to qu...	
			Manual	Females between 16 and 25 in...	
			Eligible	Congratulations! Your applicati...	
			Eligible	Congratulations! Your applicati...	
male	<div><div>< 18</div><div>[1821[</div><div>≥ 21</div></div>	<div><div>CA</div><div>Otherwise</div></div>	Ineligible	Sorry, you are too young to qu...	
			Manual	Males between 18 and 21 in C...	
			Eligible	Congratulations! Your applicati...	
			Eligible	Congratulations! Your applicati...	

Figure 3-16 Decision table definition

3.6 Decision integration

A ruleset encapsulates the rule artifacts that compose a decision. A ruleset is extracted from a rule project and its dependencies, and is associated with a RuleApp, which can group multiple rulesets. In the WODM Insurance example, a single RuleApp groups all the decisions used in the solution, that is, the decisions used for underwriting (Data Validation, Eligibility, and Pricing) and decisions made in reaction to identified situations (CustomerAcquisitionPromotions). After the RuleApp entity is deployed to the Rule Execution Server, the decisions are available for invocation.

A common way to expose a decision to potential clients is through a decision web service. A complex business decision usually requires a significant amount of context information to be properly executed and automated. There is often a relationship between the amount of information that is made available to the decision rules and the accuracy of the decision, and the extent of the business situations it can address.

For example, in a first approach to the decision implementation, you can decide that having only the number of accidents and tickets of a driver available for the rules is enough to decide the driver's eligibility for a quote.

This situation is the case of the rules for the Eligibility decision of the WODM Insurance application when assessing whether a driver presents high risks. However, you might realize that this approach yields a crude decision that ends up routing many quote requests to the manual review process.

You can later decide to refine the decision and use a more precise snapshot of the driver risk, such as a Motor Vehicle Report (MVR), which provides a detailed history of the driver's tickets, with dates and severity. This refinement empowers you to write more discriminating business rules. These actions are a part of the iterative and incremental approach to the development of a business decision, which starts with a simplified version and incrementally adds the handling of more cases.

One direct consequence of this incremental approach is from the decision web service design point of view. Although the definition of the interface is stable over time, the definitions of the ruleset parameters evolve as the footprint and the complexity of the decision increases. We must enrich the data that is received by the decision web service to execute the rules.

This enrichment cannot be performed from within the ruleset execution, as best practices of rule execution dictate that the rule engine should not perform any outside system call during its execution. Using this best practice avoids having a rule engine instance idling while waiting for the external system call to complete.

Enrichment should thus be performed before the invocation of the ruleset execution, through for example, call to data services within the implementation of a custom web-service. As a result, the WSDL of the custom web service that provides the decision operations is stable, referencing mostly business object keys and light context information, while the data needed for the ruleset parameters is gathered by the service operation.

We can initially use the Hosted Transparent Decision Service (HTDS) deployment facility of the Rule Execution Server to quickly make the decision services available to their consumers. The HTDS can then be replaced by a custom web service that is enriching the data provided by the service input before performing the ruleset invocation.

3.7 Roles, responsibilities, and decision development lifecycle

The initial development of a business rules based decision is the shared responsibility of three main roles:

- ▶ The Business Policy Manager is the source for the business rules and the business terms. This person is responsible for describing the intent of the business policy, the different business dimensions involved, the KPIs against which the decision can be measured, and how the policy is decomposed down to the rules level. This person is also responsible for providing concrete scenarios that can be run through the rules.

- ▶ The Business Policy Analyst is responsible for establishing and organizing the business terms into a usable business object model, and harvesting, analyzing, and classifying the rules. This person plays the central role in defining the BOM, capturing the rule flow and a characteristic sample of rules that goes in each task of the flow.
- ▶ The IT Developer is responsible for establishing the infrastructure of the rule project and enriching the XOM or BOM with utility functions to support business operation. This person is also in charge of preparing the decision for integration, implementing, for example, the decision services that are exposed.

The Business Policy Analyst has a pivotal role in the development of the decision. This person works with the Business Process Analyst for the identification of the decision point. After the decision points and their interfaces are agreed upon, the Business Policy Analyst can work with minimal need for synchronization with the business process development cycle.

The Business Policy Analysts provide the IT Developer with the requirements for the rule project instrumentation (for example, changes to the rule model, dynamic domains, vocabulary extensions, and so on) to best support rule authoring from a business point of view.

The overall development of the decision follows an agile and iterative approach. The best such approach is captured by the Agile Business Rules Development process described in detail in *Agile Business Rule Development: Process, Architecture, and JRules Examples* by Boyer, et al.

A high-level view of this process is presented in Figure 3-17.

Figure 3-17 Agile Business Rules Development (ABRD) process



Situation identification and response development

This chapter describes how to use the event processing capabilities of WebSphere ODM to identify specific situations of interest based on events from a broad range of applications and sources. It also describes how to create actions that make the solution respond to those situations in the wanted manner.

This decision logic is defined in Event Projects, which are initially created by IT Architects and developers. After the events, actions, data connector, and business objects are defined and verbalized, Business Users can write event rules and event rule groups that define the situations of interest and the appropriate actions to take in those situations.

4.1 Situations and events

Before describing the design approach, it is worth describing the relationship between a *situation* and an *event*. An event is usually a measurable physical occurrence that can be represented as a message in the solution. In the WODM Insurance solution, we use events to define discrete system or customer occurrences, such as offering, accepting, or rejecting a quote.

Conversely, a situation is not directly measurable. It defines a set of circumstances that might arise in the solution. It is this defined pattern that makes it useful to the business, as the business can use rules to define both:

- ▶ When that situation is deemed to occur
- ▶ The appropriate action to take when it does occur

The approach taken in WebSphere ODM is to correlate related events to build up a stateful context over time and then use event rules to make a decision based on that stateful context to determine the course of action to take. This Detect - Respond paradigm is the basis for defining event rules described in this section.

In some cases, it is useful to split this paradigm into two parts. The first part correlates related events to build up a stateful context over time and then uses event rules to simply identify a situation. The situation is represented as a special action that is immediately injected back into the event processing as a *synthetic event*. Whenever a situation is identified, there is an event that occurs that can be used to trigger the response that is required. After the situation is identified in this way, another set of event rules may determine the course of action to take whenever the situation occurs.

This chapter describes a recommended set of steps to design an event project that delivers this situational decision making behavior.

4.2 Defining situations of interest

A recommended starting point for an Event Project is for the business users to identify the initial business situations that are of interest in the solutions. This action determines the events that need to be considered and the actions that are required to respond to those situations. The definition and integration of the situations as synthetic events is undertaken by an IT Architect or Developer role working in Event Designer to meet the specifications provided by the Business User.

In the WODM Insurance scenario, there are two major situations of interest to the business:

- ▶ *Multichannel customers* represents those customers that seem keen to buy insurance and have looked for the best price on both the website and call center. If these customers can be encouraged to use a new multichannel insurance policy, support costs would be lower and a discount can be offered to encourage them to buy.
- ▶ *Customer acquisition* represents those situations where a good customer prospect (low risk with a potential revenue stream) seems to be leaving or rejecting the quotes. In these situations, WODM Insurance can offer a personalized promotion that is tailored to the customer quote request to encourage the customer to close the deal.

Each situation of interest can be captured as a synthetic event in Event Designer, if an action that can be invoked to identify that the situation occurred and an event can be used to trigger the response to the situation. Figure 4-1 shows the initial representation of these situations in Event Designer.

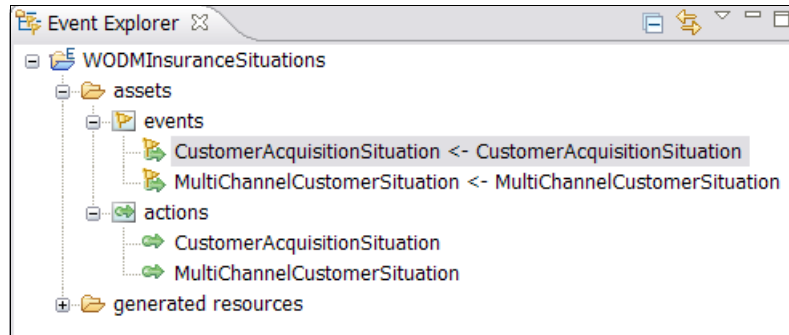


Figure 4-1 Situation definition in Event Designer as synthetic events

To allow the situation to be used easily by a Business User in their event rules, a natural language verbalization is provided (Figure 4-2).

Figure 4-2 Situation verbalization

At this stage, the situations are conceptual and the design needs to concentrate on the information needed to represent and identify the situation. This process is undertaken through business objects.

4.3 Identifying business objects and the correlation context

The basis for identifying situations is to correlate related but different events and recognize the patterns that are appearing over time. Correlation must be based on a particular piece of information provided in an event. To hide the details of the event implementations from the business user, Information is mapped out of the events into business objects whose fields can be used to define the basis for event correlation and filtering when identifying the situations.

When identifying business objects, it is likely that these same objects appear in the events that are used to populate them and the actions that are populated in response to a situation. Therefore, there are three sorts of objects:

- *Business* objects are verbalized and available to the business user. They can hold the correlated state over multiple events and represent the context in which the interaction takes place.

- ▶ *Event* objects represent the information available in an event. They must be mapped into the business objects on receipt of each event, allowing the events contribution to the context to be taken into account. Event objects are part of the integration and are not visible to the business user.
- ▶ *Action* objects are populated from the business objects when an action occurs. They allow the action to pass information that describes the state of the situation that is identified. Action objects are part of the integration and are not visible to the business user.

The mapping of fields between objects can be undertaken using four techniques:

- ▶ **Field Mapping:** The value from one field object is copied into the mapped field. For this technique to occur, the two fields must be of the same type.
- ▶ **Constant values:** The field is always initialized with a fixed value.
- ▶ **JavaScript:** The field uses a JavaScript expression to determine the value of the field.
- ▶ **Database enrichment:** If the field is needed, a value is obtained by looking up the information from a database. The keys for the lookup are defined in the data store definition.

In the WODM Insurance scenario, we consider four business objects that are meaningful to the business user:

- ▶ **Driver** represents the identity and characteristics of the main driver and customer applying for insurance. Events that relate to the same driver as determined by their full name can be correlated to understand what the customer is doing and thus the customer's likelihood to buy.
- ▶ **Vehicle** represents the identity and characteristics of the vehicle being insured. Correlating events based on the vehicle identification number (VIN) would allow situations to be detected where quotes were being requested for the same vehicle but with different drivers or in different states for fraudulent intent.
- ▶ **Quote** represents a particular quote that is requested or obtained. Correlating based on the quote ID allows a quote case history to be obtained and thus determine quote takeup rates. Because the whole scenario is about obtaining quotes, this business object can also be used to summarize the important correlation information in one place.
- ▶ **CoverageRequest** represents the type of cover requested. In general, although these fields might be used to filter events, they are not suitable as correlation keys.

These business objects can be defined directly in Event Designer or based on existing events or actions.

The Architect or Developer then defines and verbalizes these business objects so that the Business User can refer to them in their event rules when identifying the situations (Figure 4-3).

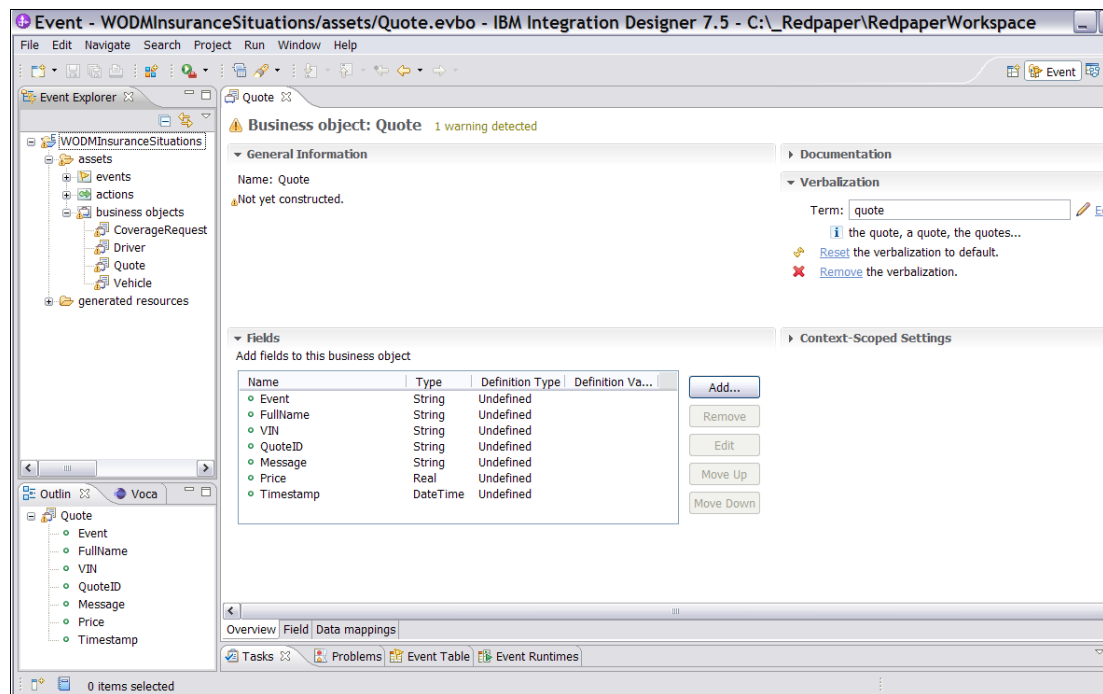


Figure 4-3 Business object definition in Rule Designer

In general, it is a best practice to minimize the fields in the Business Objects, as each field might represent additional information that must be stored in memory to represent the situation state, potentially increasing computing resource requirements. In addition, each field is exposed to the Business User role, adding complexity to the vocabulary available to them when writing the event rules. This latter issue can be resolved by removing the verbalization for the fields that do not need to be exposed to the business user, but might still be needed when populating an action.

4.4 Identifying touchpoint system events and actions

The situations and the information needed to detect the situations are defined. Before you can write the rules to identify the situations, you must integrate the solution events used to identify the situations, and find a means to populate any business object information that is not available directly from those events.

The first step in identifying the business events (and actions) is to identify the different systems that need to be integrated. These systems are referred to as *touchpoint* systems and are represented as folders containing the events and actions for that system.

In the WODM Insurance scenario, you can initially identify two main systems and summarize the events available from those systems (Table 4-1).

Table 4-1 WODM Insurance solution touchpoint systems and events

Touchpoint system	Events	Description
Website	WebQuoteOffered WebQuoteRejected AcceptWebQuote DeclineWebQuote	<ul style="list-style-type: none"> ▶ Priced quote offered to customer. ▶ Customer is ineligible for insurance. ▶ Customer accepts an offered quote. ▶ Customer declines an offered quote.
Call center	RequestCallCenterQuote AcceptCallCenterQuote DeclineCallCenterQuote	<ul style="list-style-type: none"> ▶ Customer requests a quote from the call center. ▶ Customer accepts an offered quote. ▶ Customer declines an offered quote.

While examining these touchpoint systems, identify the actions that could be triggered from a situation response (Table 4-2).

Table 4-2 WODM Insurance solution touchpoint systems and actions

Touchpoint system	Actions	Description
Website	<ul style="list-style-type: none"> ▶ CustomerReminder ▶ ReferToCallCenter 	<ul style="list-style-type: none"> ▶ Display a reminder alert to customer. ▶ Refer the customer to the call center.
Call center	<ul style="list-style-type: none"> ▶ FollowUp ▶ OfferPromotion ▶ InvestigateFraud 	<ul style="list-style-type: none"> ▶ Instructs a call center operator to follow up on a quote. ▶ Instructs an operator to offer a quote promotion. ▶ Warns an operator of potentially fraudulent quotes.

Each of these touchpoints can be captured in the Event project together with the events and actions that are integrated (Figure 4-4).

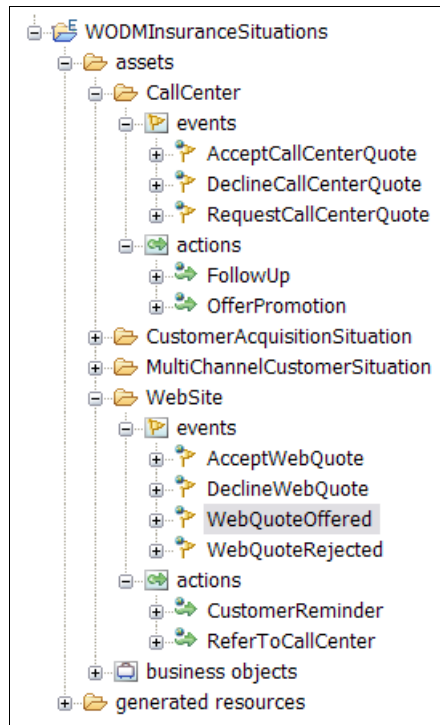


Figure 4-4 Touchpoint systems, events, and actions

Each event can be verbalized (and documented) to provide a clear means by which the Business User can refer to the event when developing the rules for situation identification (Figure 4-5).

A screenshot of the 'WebQuoteOffered' event configuration interface. It shows a 'General Information' tab with the name 'WebQuoteOffered'. A 'Documentation' section has a text area with 'Priced quote offered to customer'. A 'Verbalization' section has a text field with 'web quote offered' and buttons for 'Reset' and 'Remove'.

Figure 4-5 Event verbalization and documentation

The definition of the event can also describe information passed within the event in terms of event objects (Figure 4-6).

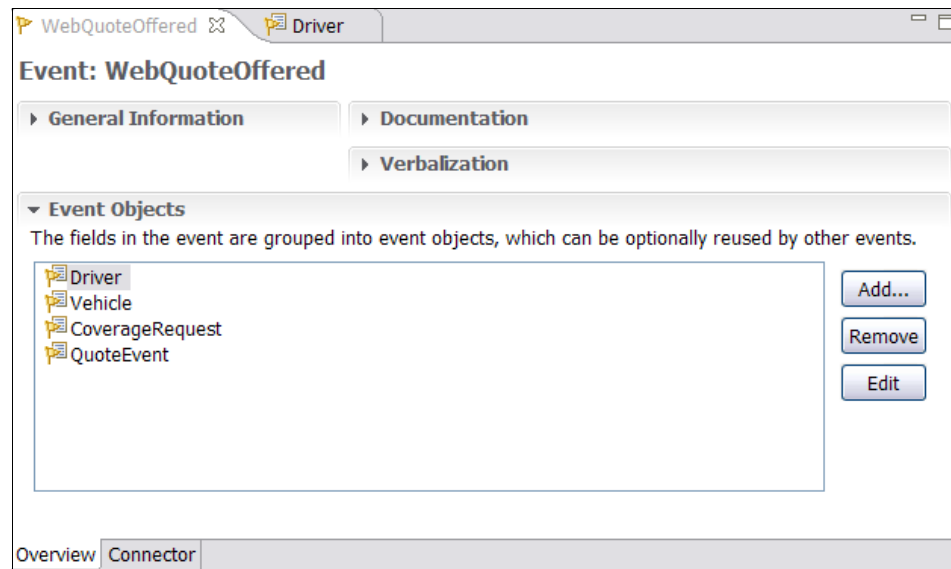


Figure 4-6 Event object definition

The final stage of configuration is to define the connector that the touchpoint system uses to signal that the event occurred and pass in the event data objects. These connectors can be selected by the developer from drop-down menus in the Connector tab (Figure 4-7).

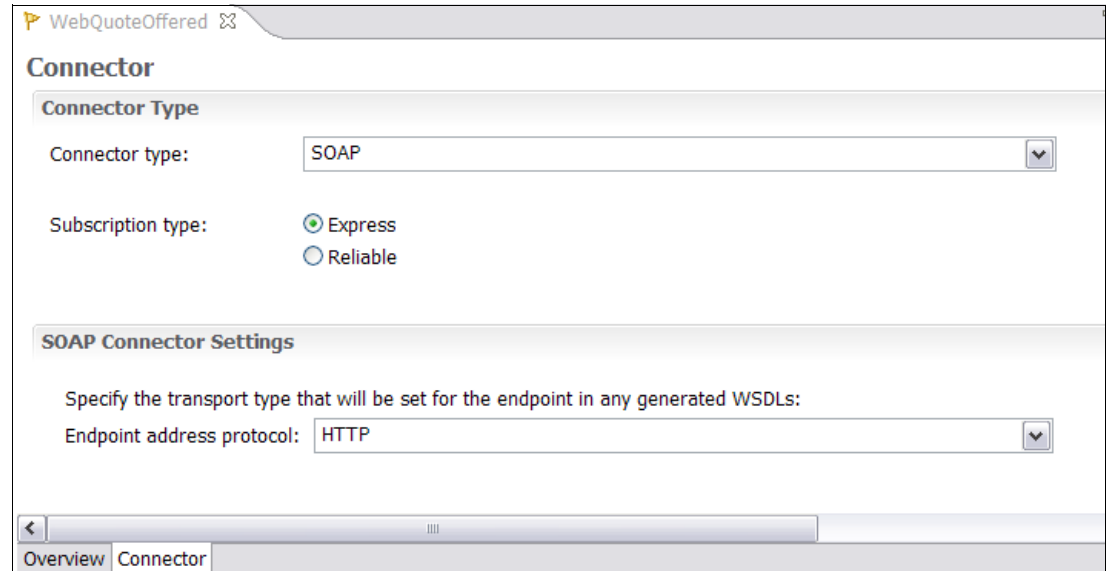


Figure 4-7 Event connector definition

Connectors are provided for the following protocols:

- ▶ Email
- ▶ File
- ▶ FTP
- ▶ HTTP
- ▶ JDBC

- JMS
- SOAP

After the connector type is selected, settings appropriate to that connector can also be set up.

More details about the technology connectors available from WebSphere ODM are available in the Information Center at the following address:

<http://publib.boulder.ibm.com/infocenter/dmanager/v7r5/topic/com.ibm.wbe.appdev.doc/doc/whataretechnologyconnectors.html>

The connector allows any events in the solution to be referenced by the rules that the business user requires.

4.5 Mapping event data to business objects

Each event object is defined in terms of a number of primitive fields in the same way as the business objects are defined. These objects can be defined manually in Event Designer or imported automatically from schemas used to describe the events. Although different connectors are often available for the events (JMS, SOAP, and HTTP), the common underlying definition format is usually an XML schema. This situation results in an event object definition similar to the one shown in Figure 4-8.

Event object: Driver

General Information
Name: Driver

Fields
Add fields to this event object

Name	Type	Mapped
DriverID	Integer	True
FirstName	String	True
LastName	String	True
Age	Integer	True
StateOfResidence	String	True
Occupation	String	True
Gender	String	True
Married	Boolean	True
Graduated	Boolean	True
FullTimeStudent	Boolean	True
GoodStudentCertificate	Boolean	True
DrivingLicenseNumber	String	True
FirstDrivingLicenseDt	DateTime	True
CompletedDriversEdCourse	Boolean	True
VehicleVandalizedOrStolen	Boolean	True
LicenseSuspendedOrRev...	Boolean	True
DUI	Boolean	True
NumberOfAccidents	Integer	True
NumberOfTrafficTicket	Integer	True

Number of Occurrences
Specify the number of instances of this event object that should appear in the event.

☒ Unlimited

0 Minimum number of occurrences

1 Maximum number of occurrences

Overview | Field | Field Constructor

Figure 4-8 Event object definitions

The cardinality of the event object within the event can be set, which means that an event can contain multiple event objects. The fields within an event object are limited to simple types. There are no hierarchical types allowed inside an event object. Each field can contain only one value, and if the field is not present in an event object, a default can be used (Figure 4-9).

The screenshot shows a configuration window titled "Field: DriverID". At the top, there is a "Choose field:" dropdown menu with "DriverID" selected. Below this, there are two main sections: "General" and "Documentation".

In the "General" section, there are two input fields: "Name:" with the value "DriverID" and "Type:" with a dropdown menu showing "Integer".

In the "Documentation" section, there is a text area labeled "Enter a description of this field:" containing the text "(xs:int)".

Below these sections is a "Default Value" section. It contains a text area labeled "Default value:" and a note: "When the event object field is empty, the default value is used." Below the text area is a label "A fixed value to be used for the field." with an empty input field.

At the bottom of the window, there is a tabbed interface with three tabs: "Overview", "Field", and "Field Constructor". The "Field" tab is currently selected.

Figure 4-9 Event Object field definitions

The final stage in the event information mapping is to define which business objects are created and how the business object fields are initialized as a result of an event arriving.

In Event Designer, each event object can have its fields mapped to business object fields. Each event object may construct fields in any number of business objects, but each business object can be constructed only from one event object within an event. You cannot have a single business object with fields populated from different event objects.

In Figure 4-10, the Quote and Quote Event business objects are both initialized from the QuoteEvent event object, but they could not have fields initialized from the Driver event object in the event.

QuoteEvent

Event object: QuoteEvent

General Information

Name: QuoteEvent

Documentation

Fields

Add fields to this event object

Name	Type	Mapped
Event	String	True
FullName	String	True
VIN	String	True
QuoteID	String	True
Message	String	True
Price	Real	True

Add...

Remove

Edit

Move Up

Move Down

Number of Occurrences

Specify the number of instances of this event object that should appear in the event.

☒ Unlimited

0 Minimum number of occurrences

1 Maximum number of occurrences

Field Constructors

Add business objects that will receive data from the fields in this event object:

Business Object	Field	Data Type	Definition Type	Definition Value
Quote	QuoteID	String	Field	QuoteID
Quote	Price	Real	Field	Price
Quote	FullName	String	Field	FullName
Quote	Event	String	Field	Event
Quote	VIN	String	Field	VIN
QuoteEvent	Timestamp	DateTime	Javascript	_Global._FireTime;
QuoteEvent	Message	String	Field	Message
QuoteEvent	QuoteID	String	Field	QuoteID

Add...

Edit...

Remove...

Overview

Field

Field Constructor

Figure 4-10 Constructing business objects from event objects

In many cases, the definition of each business object field is simply obtained from the mapped event object field. For more complex mapping combining fields, JavaScript expressions can be used to map the field values, and in simple situations, the Business Object field can be populated with a fixed value.

This mapping allows the Business Object fields required for situation identification to be populated in a consistent manner from the information provided in the event.

4.6 Using event rules to invoke actions in response to patterns of events

The simplest form of event processing decision is to use an event rule to define the actions to take in response to a particular pattern of events. Event rules are tested on receipt of a specific event. Events can be correlated with previous events based on a Context ID that refers to a business object field. This situation means that the rule then has access to all business objects, events, and actions that are related by the same Context ID. In the WODM Insurance scenario, there is a requirement to remind a customer if they have not responded to a quote. This action can be undertaken by using an event rule (Figure 4-11).

The screenshot shows a web-based configuration interface for an event rule named 'KeepCustomerInterested'. The interface has a tabbed structure with 'General Information' and 'Documentation' tabs. Under 'General Information', the 'Name' is 'KeepCustomerInterested'. Below this, the 'Event Rule Context and Event' section contains two input fields: 'Context ID' with the value 'the applicant name of the quote' and 'Event' with the value 'web quote offered'. There are buttons for 'Change Context...', 'Remove', and 'Change Event...'. The 'Content' section has a text area for defining the rule logic. The logic is as follows:

```
after 1 minutes
if
  past occurrences of customer accepts web quote within 1 minutes equals 0
  and past occurrences of customer declines web quote within 1 minutes is 0
then
  remind customer of offered quote ;
```

At the bottom, there is an 'Overview' tab.

Figure 4-11 Simple Event rule - mapping an event pattern to an action

In this case, the rule evaluates the situation a minute after the quote is offered and checks to see if the customer accepted or declined a quote in this time. If there is no response from the customer in this interval, an action is raised to remind the customer of the quote.

In an event rule then clause, it is not possible to set Business Object fields from the rules. This situation means that currently (unlike with business rules) business users cannot change the state of the business objects. This mapping must be undertaken at design time.

In current solutions, the business objects are used to define the correlation and aggregation needed for the rules and action objects. This process includes accumulating correlated results and maintaining arrays of correlated objects.

It is possible to define action parameters that are supplied by the rules. These action parameters are then mapped into the Action described in 4.7, “Mapping business objects to action data and executing the action” on page 50. This situation allows the business rule to set information in an action (Figure 4-12).

Event rule: KeepCustomerInterested

▼ General Information ► Documentation

Name: KeepCustomerInterested

▼ Event Rule Context and Event

Context ID: [Change Context...](#) [Remove](#)

Event: [Change Event...](#)

Content

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
after 1 minutes
if
    past occurrences of customer accepts web quote within 1 minutes equals 0
    and past occurrences of customer declines web quote within 1 minutes is 0
then
    remind customer of offered quote with message: "You have not responded to your quote"
```

Overview

Figure 4-12 Using action parameters in an event rule

4.7 Mapping business objects to action data and executing the action

After an action is triggered, the WebSphere ODM event run time populates the action objects with information provided in the business objects. Figure 4-13 shows the definition of the CustomerReminder Action, showing the different Action Objects and the verbalization of the action that was used in the preceding event rule.

The screenshot shows the 'CustomerReminder' action definition in a web interface. The title bar says 'CustomerReminder' with a close icon. The main heading is 'Action: CustomerReminder'. There are two tabs: 'General Information' and 'Documentation'. The 'General Information' tab is active, showing 'Name: CustomerReminder'. The 'Documentation' tab is also visible, showing a 'Verbalization' section with 'Navigation:', 'Phrase: remind customer of offered quote', 'Action:', and 'Template: remind customer of offered quote with message: {Message}'. Below the template, there are links to 'Reset the verbalization to default.' and 'Remove the verbalization.' The 'Action Objects' section is expanded, showing a list of objects: Driver, Vehicle, QuoteID, Message, and CoverageRequest. To the right of the list are buttons for 'Add...', 'Remove', and 'Edit'. At the bottom, there is a checkbox labeled 'Automatically generate multiple actions if the number of occurrences of an Action Object would exceed its maximum'.

CustomerReminder

Action: CustomerReminder

▼ General Information

Name: CustomerReminder

► Documentation

▼ Verbalization

Navigation:

Phrase: remind customer of offered quote

Action:

Template: remind customer of offered quote with message: {Message}

[Reset](#) the verbalization to default.

[Remove](#) the verbalization.

▼ Action Objects

The fields in the action are grouped into action objects, which can be optionally reused by other actions.

- Driver
- Vehicle
- QuoteID
- Message
- CoverageRequest

[Add...](#)

[Remove](#)

[Edit](#)

☐ Automatically generate multiple actions if the number of occurrences of an Action Object would exceed its maximum

Figure 4-13 Action object definitions

The action objects and their fields are populated from the business objects in the same way as for the event objects. This action is shown in Figure 4-14 for the message action object, which in this case is initialized from the action parameter.

Action object: Message

▼ General Information
Name: Message

▼ Documentation
Enter a description of this asset:
Message

▼ Fields
The fields in the action are grouped into action objects, which can be optionally reused by other actions.

Name	Type	Definition Type	Definition Va...
Message	String	Parameter	Message

Buttons: Add..., Remove, Edit, Move Up, Move Down

▼ Number of Occurrences
Specify the number of instances of this action object that should appear in the action.

☐ Unlimited

1 Minimum number of occurrences

1 Maximum number of occurrences

Overview | Field

Figure 4-14 Populating Action object fields from business objects

As for the event object field mapping, fields can be mapped to constants, and business object fields or JavaScript expressions can be used to calculate the value.

After the Action object data fields are populated, the action can then be sent to the touchpoint system using the appropriate technology connector (Figure 4-15) for the customer reminder that is sent to the website using the HTTP protocol.

CustomerReminder Message

Connector

Connector Type

Connector type: HTTP

Action packet version: 6.2

Subscription type: ☒ Express ☐ Reliable

HTTP Connector Settings

Action format: Connector packet

Enter the connection details for the server:

Protocol: HTTP

Host name: localhost

Port: 9080

Directory: /InsdemoWebQuoting/event/

File name pattern: CustomerReminderAction

Preview: http://localhost:9080//InsdemoWebQuoting/event//CustomerReminderAction

Overview Connector

Figure 4-15 Using Action technology connectors to execute actions in the solution

Having followed the complete cycle of responding to an event pattern by executing an action, you can now look at how this process is applied to business situations.

4.8 Identifying situations using temporal event rules

The events are defined and mapped to initialize the content of the business objects. You can now define the rules to identify the situations. This task can be undertaken by IT Architects (using Event Designer) or by Business Users using Decision Center.

It is worth considering creating folders to collect together all the event rules associated with a particular situation. This setup allows the rules to be managed consistently, where all the behaviors for a particular situation are in one place. This folder can also be used to contain the situation synthetic event (Figure 4-16).

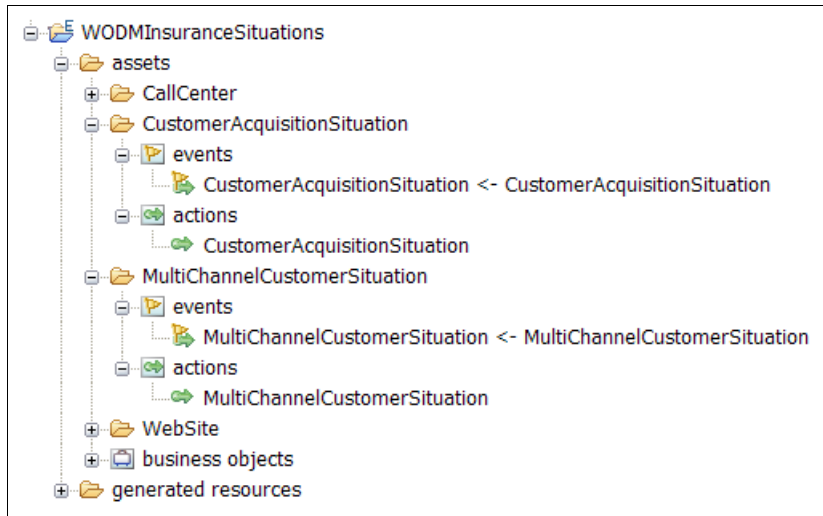


Figure 4-16 Using folders to organize situations

In the WODM Insurance scenario, you want to acknowledge that a MultiChannelCustomer situation occurred if a customer obtained at least two quotes from the website and one from the call center. You can accomplish this task by writing an event rule (Figure 4-17).

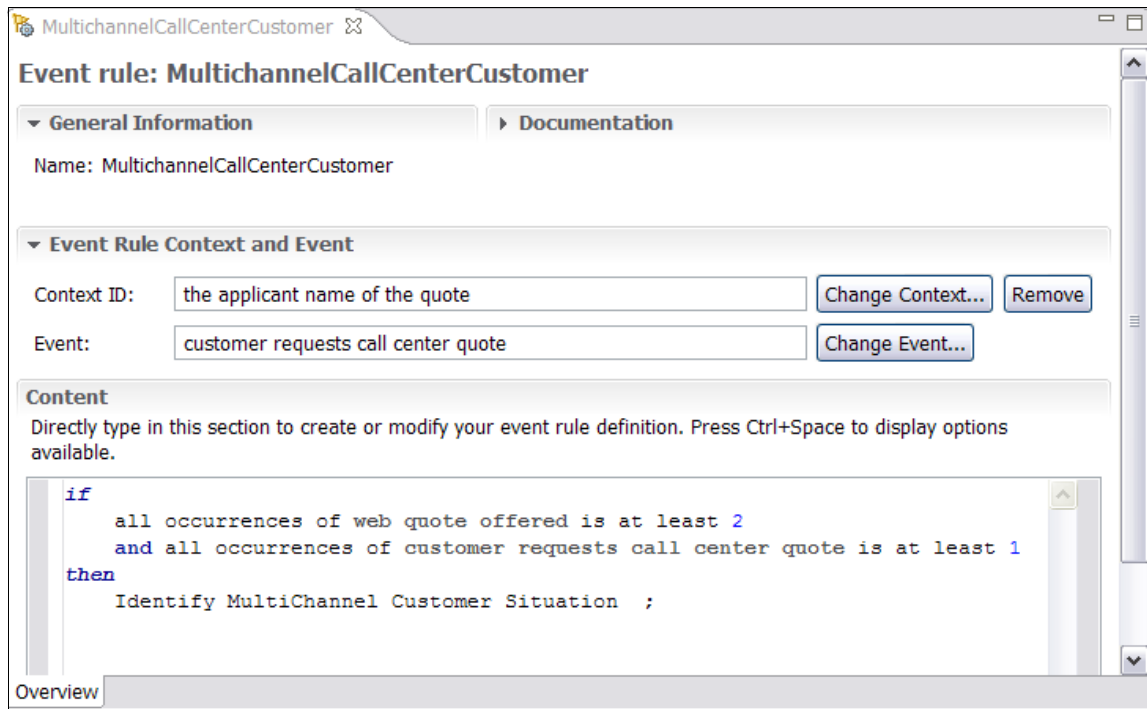


Figure 4-17 Event Designer situation identification event rule

This rule defines the patterns that indicate that the customer is using both website and call center to obtain a quote.

As this pattern can arise only in response to a customer obtaining a quote from the call center, other event rules need to be defined that indicate the pattern triggered by a customer obtaining a quote from the website. In this case, as there are likely to be many quote requests, the conditions under which the situation is identified might be more stringent (Figure 4-18).

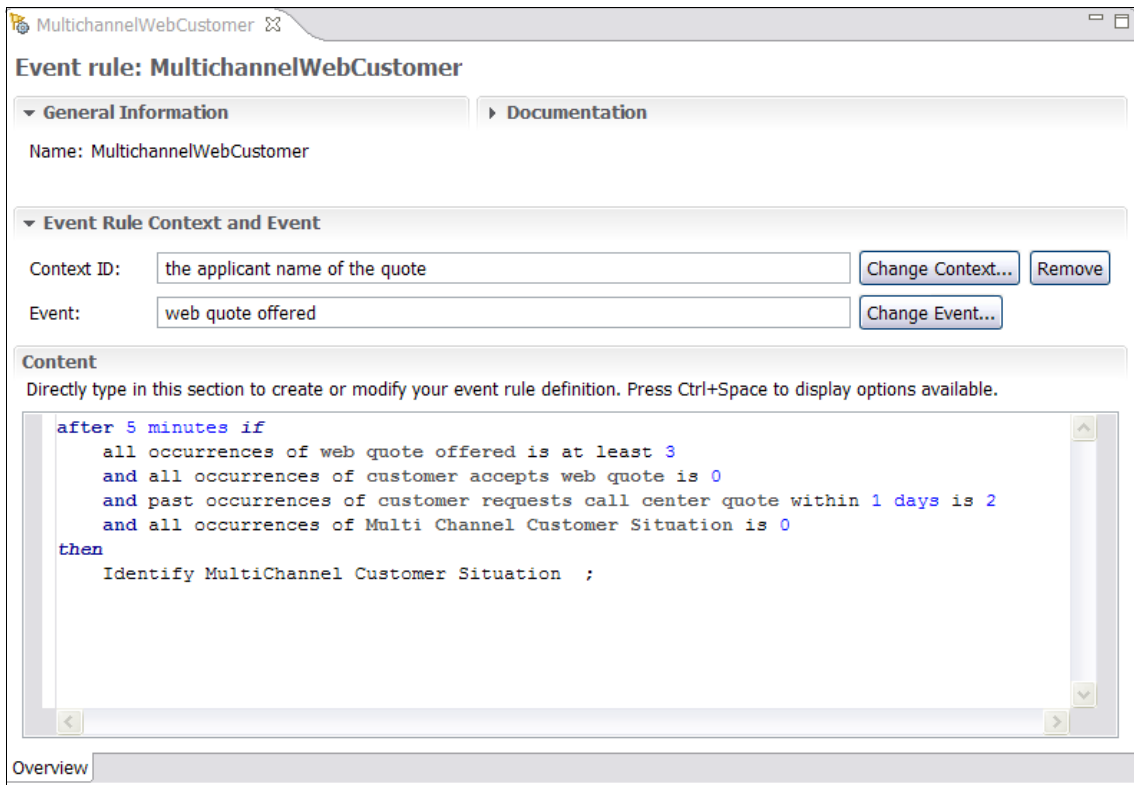


Figure 4-18 Event Designer situation identification event rule

By providing a number of these event rules, the conditions under which situations occur can be clearly expressed and managed by the business user. Each of these rules has a number of important clauses (Table 4-3).

Table 4-3 Event rules

Clause	Description
<Event>	This clause selects the event that triggers the evaluation of the rule. Rules are not evaluated on every event, only on the specific event identified. In this case, the event is whenever a web quote (or in the previous case a call center quote) is offered.
<Context ID>	This clause selects the business object field (populated as a result of the event) that is used to correlate other events and actions into the same context. In this case, it is the applicant's name that derived from the driver's full name.
<Delay Clause> "after 5 minutes"	This clause determines that the rule is not evaluated immediately in response to the event, but delays the evaluation to allow other events or actions to be inserted into the context.

Clause	Description
<Condition Clauses> "if" <condition> "and" <condition>	This clause allows a number of condition clauses to be combined by using Boolean operators. If all conditions are met, the <Action> clauses are invoked.

Examples of condition clauses include:

- ▶ “all” occurrences of web quote offered is at least 2": Counts up the total number of events or actions (web quotes offered in this case) in this context (this customer) and compares the count against a number.
- ▶ “past” occurrences of customer requests call center quote within 1 day is 2": Counts up the number of events or actions (customer requests call center quote, in this case) within a specific time interval. The count can then be compared against a number.
- ▶ <Action Clause "then" <action> ; <action>: This clause determines the different actions that are invoked if the condition clauses evaluate to true. This clause just specifies the name of the action to be undertaken.

4.9 Identifying situations using contextual event rules

Although some situations can be identified by looking at the temporal patterns between events, WebSphere ODM also supports conditions based on information in the Business Object fields. In the WODM Insurance scenario, the Customer Acquisition Situation needs to use this information to determine if the customer has the wanted characteristics.

To simplify the event rules, it is possible to combine conditions into filters that can be verbalized and used in the event rules. A filter that determines the characteristics of a desirable customer is shown in Figure 4-19.

Filter: Desirable customer

General Information
Name: Desirable customer

Documentation

Verbalization
Phrase: desirable customer
[Reset](#) the verbalization to default.
[Remove](#) the verbalization.

Content
Directly type in this section to create or modify your filter definition. Press Ctrl+Space to display options available.

```
the age of the driver is between 18 and 50
and the number of accidents the driver has been involved in is at most 1
and the driver has completed a drivers education course
and it is not true that the driver has had their license suspended or revoked
```

Overview

Figure 4-19 Using filters to simplify complex conditions

The grammar used in these rules is the same as in the business rules projects, but the vocabulary is drawn from the verbalizations of the business objects and their fields rather than a Business Object Model. This situation means that care must be taken to ensure consistent verbalization if information models are shared across Rules and Events.

After a filter is defined, it can be used in an event rule in combination with temporal conditions to identify the situation illustrated in Figure 4-20 for the Customer Acquisition Situation.

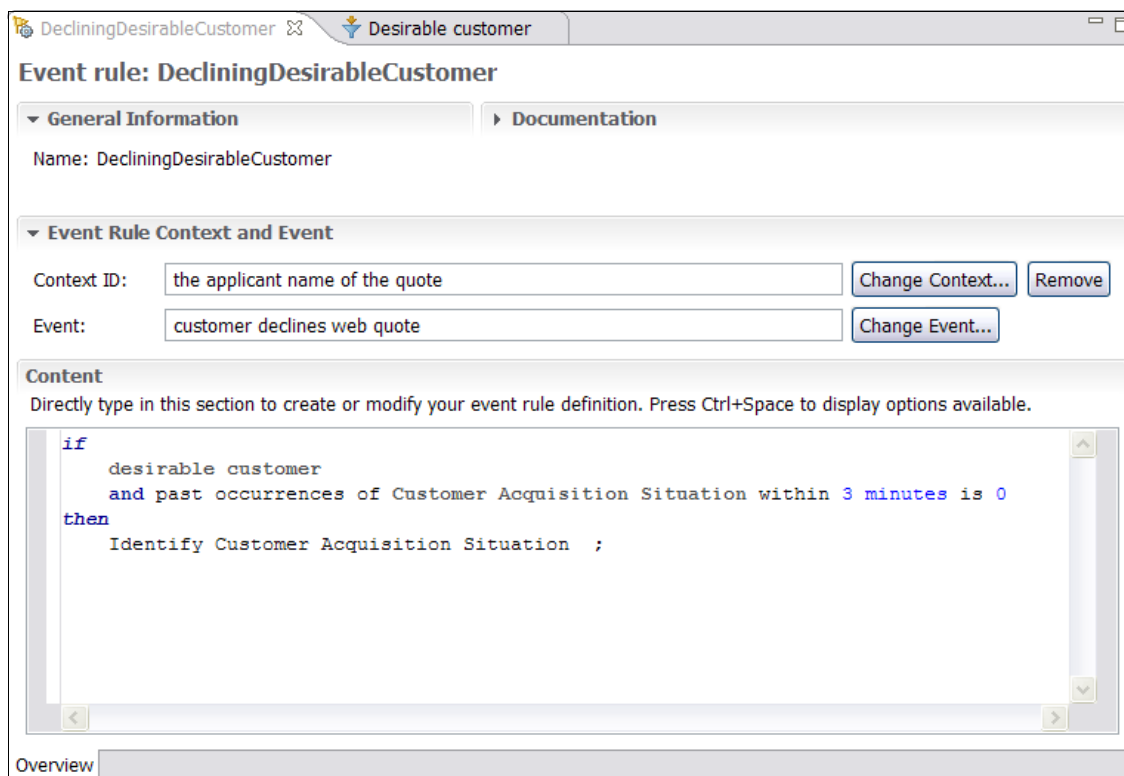


Figure 4-20 Using Filters in event rules to enhance understanding

This combination of temporal and contextual rules provides a powerful way of clearly defining the situations that are important to the business.

4.10 Reacting to identified situations using event rule groups

When a situation is identified, the objective of decision management is to decide how best to respond to that situation. In many cases, the response is not a simple action execution but a sequence of related actions.

In the WODM Insurance scenario, the objective of identifying the multichannel customer situation is to encourage the customer to accept an offered quote. This action is undertaken by offering a promotional discount on their last quote. The promotional discounts are only available through the call center, so we need to direct the customer to the call center if their promotion discount is identified as a result of the website.

This action can be undertaken using an event group, which provides a collection of responses to the same situation, as shown in Figure 4-21.

Event rule group: MultichannelSituationResponse

General Information Documentation

Name: MultichannelSituationResponse

Event Rules

Context ID: the applicant name of the quote Change Context... Remove

Event: Multichannel Customer Situation Change Event...

Expand All Collapse All

then offer customer a promotion ;

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
if the event of the quote contains "Web"
then
refer customer to call center ;
```

Add Remove Move Up Move Down Add the Otherwise Event Rule

Overview Full View

Figure 4-21 Using Event Rule Groups to define multiple actions

The mapping from the business objects to the actions is performed in the same way as described 4.7, “Mapping business objects to action data and executing the action” on page 50. When the action is executed, an instruction is sent to the appropriate system in the solution to act accordingly.



Detect-Decide-Respond pattern design

In this chapter, we describe a method for designing a Situation and Decision to work in a Detect- Decide - Respond pattern. After this foundation is designed and implemented by your IT department, the Business Users can use WebSphere ODM Decision Center to quickly define the best behavior for the Situation Identification, Decision, and Action.

The WODM Insurance scenario uses the Customer Acquisition Situation to identify a situation where a customer might be about to be lost. Business rules are used to decide about a personalized promotion to offer to that customer to get them to buy the insurance. Having decided what promotional offer to make based on the customer's history, the event processing must then be used to turn the decision into actions, and close the deal with the customer.

Applying this design to the solution means that business users can use Decision Center to rapidly define and test:

- ▶ Situation Identification Rules: When do we think we have a customer that we want to retain?
- ▶ Promotion Decision Rules: What promotion should we give to this customer to make them buy?
- ▶ Promotion Action Rules: How do we engage the customer to offer the promotion and close the deal?

5.1 Overview

The Situation Design described in Chapter 4, “Situation identification and response development” on page 37 showed how to use the event processing capability of WebSphere ODM to detect situations and act in response to those decisions. It is possible to use filters and event rules to make simple contextual decisions about the situation in determining the response, but a much more powerful pattern can be achieved by using the contextual decisions of the business rules engine.

In this pattern, you use the Situation identification concepts described in Chapter 4, “Situation identification and response development” on page 37 to detect that a situation occurred and to decide what to do. You then accumulate the information you know about the situation and pass that information to a decision service to make an informed appraisal of what to do in this particular situation case. The response from the decision can then be used in the event rules to decide which action to take.

This pattern is summarized in Figure 5-1 and is described in more detail in the following sections.

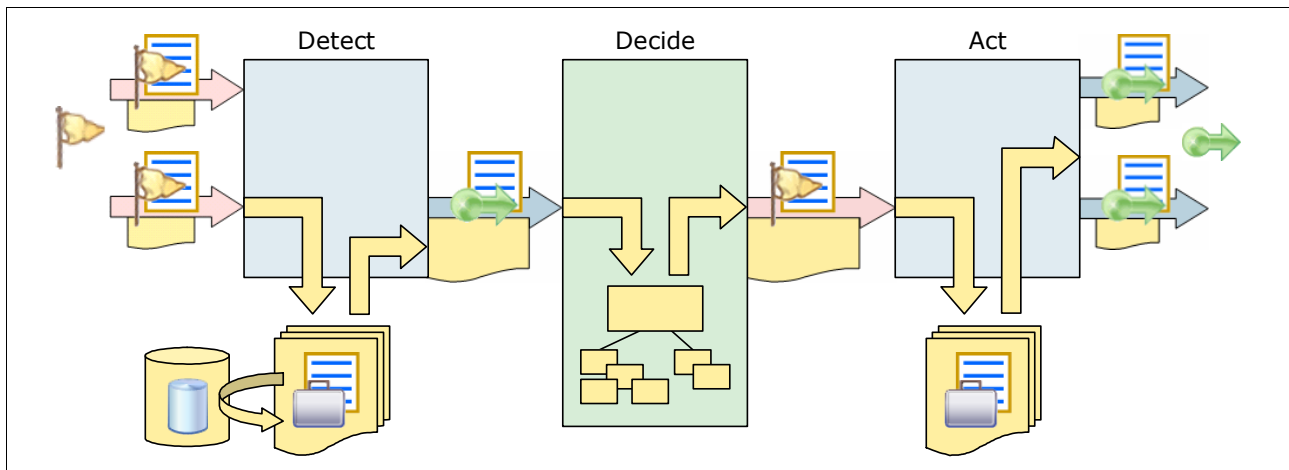


Figure 5-1 Detect-Decide-Respond pattern combining event and rules-based decisions

In Figure 5-1:

1. The Detect stage uses Event rules to identify a situation when the decision needs to be made based on an event pattern, as described in Chapter 4, “Situation identification and response development” on page 37.
2. During this processing, information about the situation is accumulated, as described in 5.2, “Using context variables to accumulate situation state” on page 61.
3. In many cases, the events might not hold enough information, so Situation information can be enriched using JavaScript or data sources, as described in 5.3, “Using data stores to enrich a situation state” on page 64.
4. The information provided to the situation can be mapped to a decision service and the decision made, as described in 5.4, “Making a decision in response to a situation” on page 66.
5. The response from the Decision is then used to generate appropriate actions, as described in 5.5, “Responding to a decision” on page 71.

5.2 Using context variables to accumulate situation state

Chapter 4, “Situation identification and response development” on page 37 described how to detect business situations in response to a pattern of events by correlating events with the same context. It also showed how the information contained in the events could be mapped into the actions that result from the event rules (Figure 5-2).

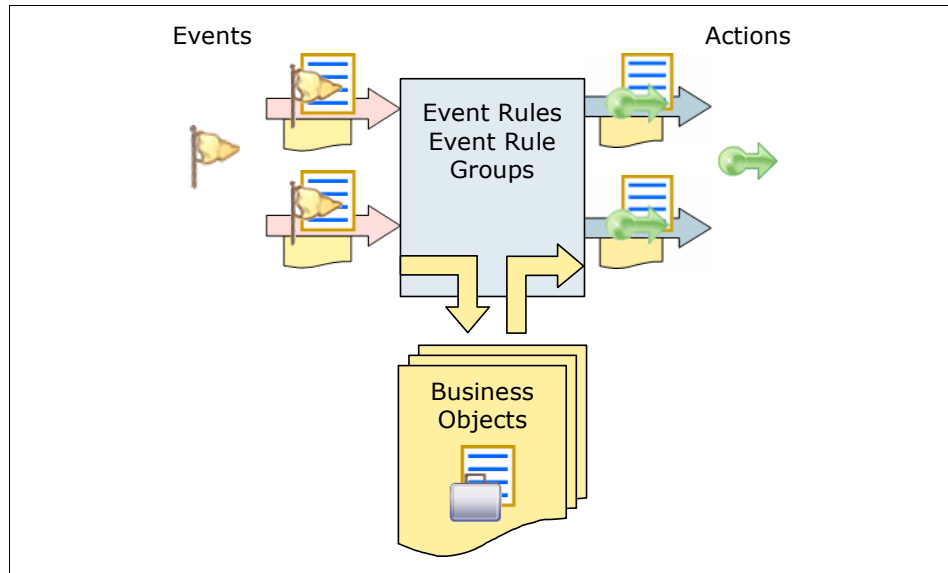


Figure 5-2 Mapping of information through business objects in event processing

Business objects can also be used to hold arrays of information based on successive events that are correlated with the same context.

In the WODM Insurance scenario, this situation means that if a customer requested a number of quotes (possibly from different channels or systems), the quote details, driver’s details, vehicle details, and coverage requests could be stored in separate business object arrays. This situation allows comparisons across successive quotes.

Figure 5-3 shows an example of setting up a QuoteHistory Business object to hold summary information about quoting events for a context. The definition and verbalization can be set as for any other business object.

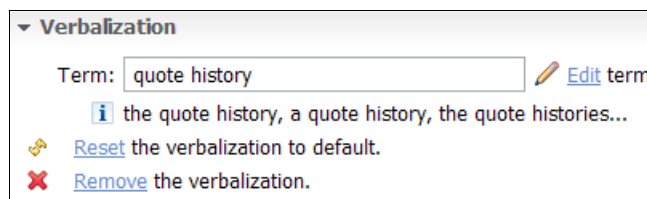


Figure 5-3 Accumulating array verbalization

The fields are defined based on the event object that populates each element of the array (Figure 5-4).

Name	Type	Definition Type	Definition Va...
Event	String	Undefined	
FullName	String	Undefined	
VIN	String	Undefined	
QuoteID	String	Undefined	
Message	String	Undefined	
Price	Real	Undefined	

Figure 5-4 Accumulating array fields

There might be situations where an event does not initialize a business object field. In this case, the Definition Type setting can be used. This setting provides a number of different ways to define the business object field value:

- ▶ Constant values: The field is initialized with a fixed Definition Value or a named constant.
- ▶ Filters: The field is initialized with the Boolean result of a filter. This tool allows the event rules to set binary flags in the business objects.
- ▶ Javascript: The field uses a JavaScript expression to determine the value of the field.
- ▶ Database enrichment: If the field is needed, a value can be obtained by looking up the information from a database. The keys and values for the lookup are defined in the data store definition. A Data mappings tab for the business object then defines how each field in the business object relates to the lookup. It defines which fields form the keys (and therefore must be present) and which fields are populated as a result of a lookup.

There are three types of business object context settings supported by the event run time:

- ▶ Single Event: This type of context populates the business object based on an individual event. Each successive event overwrites the variable. These variables might be an array if each event provides multiple objects.
- ▶ Summary: This type of context uses JavaScript to combine the results from each event with results from previous events with the same context. This context can be useful for determining statistical patterns and trends in sequences of events.
- ▶ Multiple Events: This type of context provides an array of values that is kept in a rolling buffer. As new events arrive in the same context, they are added to the top of the list. Objects are removed from the end of the list either when the list becomes a certain size or after a certain time interval. This issue is important to understand in an operational system, as it implies that all this information for every event is persisted for this time. This form of context variable is useful for maintaining event histories and providing the information to make a complex situational decision.

These context- scoped settings for business objects are shown in Figure 5-5.

▼ Context-Scoped Settings

Configure this business object to store values received across multiple events within a context.

☐ Contains data received from a single event.

Specify how many instances of the business object exist at run time in response to the incoming event.

☐ Always treat as a single instance (scalar)

☐ Always treat as multiple instances (array)

☐ Can be either

☐ Maintains a single set of values based on a calculation of values from multiple events (summary instance).

☒ Maintains an array of values from multiple events (accumulating array).

☐ Specify the number of events over which to retain values in the array.

Maximum number of occurrences

☒ Specify the period of time over which to retain values in the array.

Days:

Hours:

Minutes:

Seconds:

Figure 5-5 Business object context scoped settings

The top radio button shows the default settings that we used in Chapter 4, “Situation identification and response development” on page 37, where each business object contains data from a single event. The illustrated setting provides a Quote object instance for each event received. In this case, data is held in the business object array for 5 minutes.

The settings also highlight another way of accumulating information through a summary instance. This way allows event fields to be averaged or counted and reflect a single instance that describes the sequence of data.

All business objects can be mapped to action objects with the accumulating array providing a complete list of the business objects instances reflecting the quote history over the retention period. This situation means that when we detect a situation, the situation context can now contain a set of objects reflecting all the correlated events (Figure 5-6). This set of objects can be added to the situation action and is displayed as the event data for the situation event.

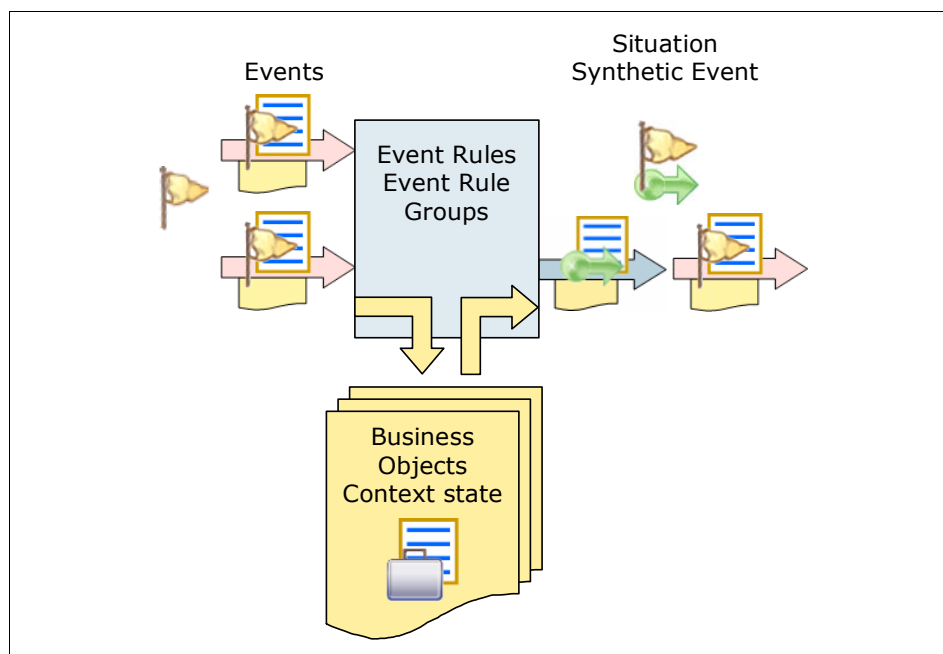


Figure 5-6 Situation context

The information available when the situation is identified can reflect the whole of the event history if required. The amount of information recorded this way has a significant impact on performance, so it is important to be precise about the information stored and how it is used when deciding what to do in the next steps.

5.3 Using data stores to enrich a situation state

This section describes how to use the event database connections to provide information not available in the situation state, but needed to make a decision.

It is a best practice in many event processing or messaging solutions to minimize the information being sent between systems. If information is not used or not needed, then sending it just in case causes a network bandwidth impact. As we saw in 5.2, “Using context variables to accumulate situation state” on page 61, maintaining a large numbers of fields in an object can also have performance implications for event processing. Relying on all required information being available in an event also has implications for change management, as the slightest change in information content needed means that all event sources and sinks must be modified to handle the new event structure.

In practice, most detailed records are stored in a database and can be retrieved by using a unique key to those records. When we look at the objects that we are using in the WODM Insurance scenario, we can see that we already identified these keys and are using them in the solution:

- ▶ The customer or driver uses a key of the driver's full name.
- ▶ The vehicle uses a key of the vehicle identification number
- ▶ The decision warehouse stores the quotes and decisions according to a quote or decision ID.

This situation means that it is possible to minimize the information sent around the event processing but still be able to retrieve the full information needed to make a decision.

WebSphere ODM provides a database connector that can be used during the event processing to augment Business Object fields (Figure 5-7).

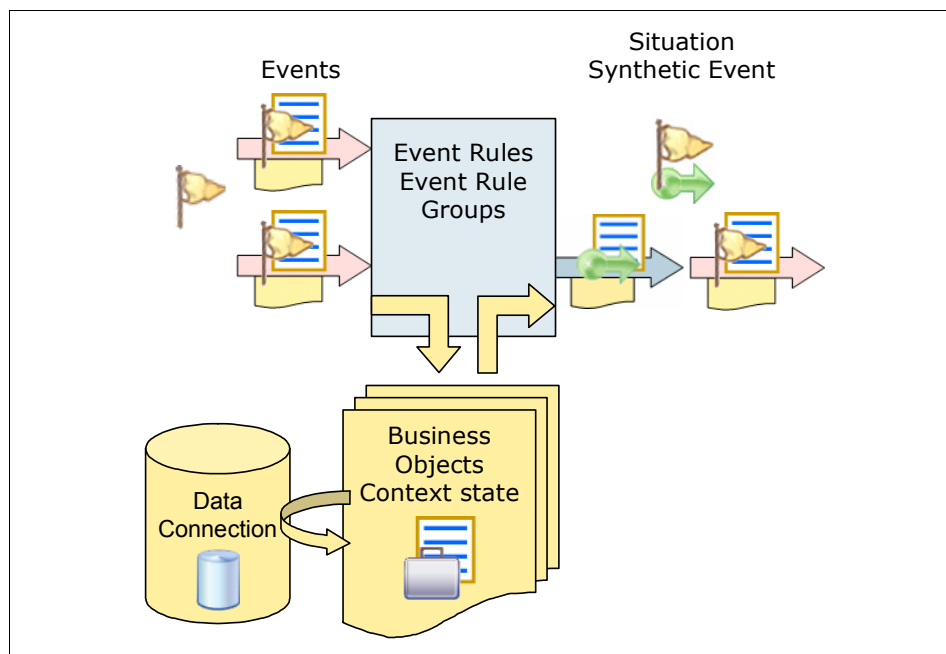


Figure 5-7 Usage of a data store to enrich the situation context

For example, a Customer Business Object could be available from a customer database. This object would include static characteristics, such as the phone number, email, and address, which are not normally used in any of the decision making, but might be needed to take appropriate action. Other fields such as Date of Birth and StateOfResidence might be used to cross-check with information provided on the quote.

Access to the Customer Records is keyed to the FullName of the driver, which is also the correlation context for the situation.

If the synthetic event action references the Customer Business Object, the run time looks up the missing fields in the Customer Business Object from the database connector. This action completes the information available in the customer object, allowing it to be included in the Situation action objects.

5.4 Making a decision in response to a situation

Section 5.3, “Using data stores to enrich a situation state” on page 64, described how to populate a situation with all the information needed to make a decision. In this section, you see how we can map that situational information onto a decision service call and get the response back in the form of an event.

In the WODM Insurance scenario, in response to the Customer Acquisition Situation situation, you are going to call the Customer Acquisition Promotions decision to decide what promotion to offer the customer.

The Decision is implemented as a service and can be started from the event run time by using the SOAP web services connector (Figure 5-8).

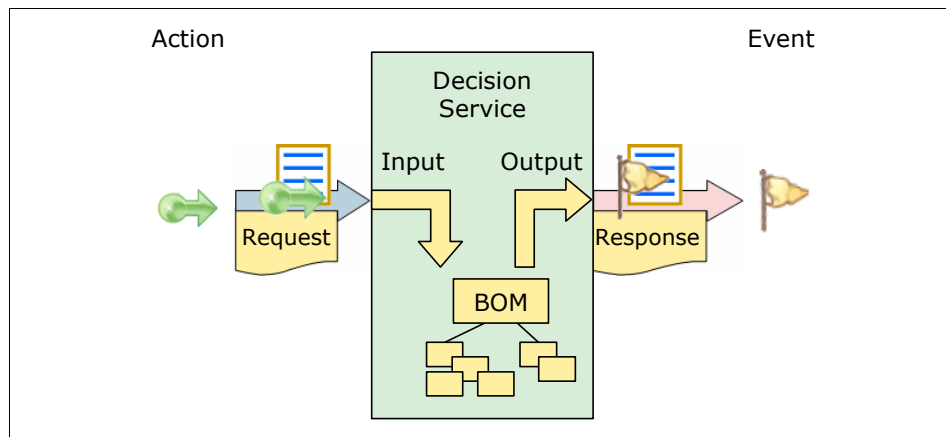


Figure 5-8 Starting a decision service from the event run time

The information needed to make the decision is passed as an input parameter to the web service. This parameter can be mapped onto the action objects using the SOAP connector.

When the decision returns, the response is displayed as a new event with the event objects corresponding to the output parameters of the service.

In the WODM Insurance scenario, you input in the driver, vehicle, coverage request, quote, and quote history. The Customer acquisition promotions look at the current quote (driver, vehicle, coverage request, and quote) and might look at the quote history to determine what the customer is trying to achieve and therefore what promotion should be offered. In practice, looking at the coverage request history might be more meaningful to determine what the customer is trying to achieve. If you are looking for fraudulent behavior, you might need to look at the driver and vehicle histories. As this action necessitates storing more information in the situation context, a balance must be made when considering the information to pass between the situation and decision.

5.4.1 Creating events and actions from a decision service

The simplest way of performing this integration is to obtain the WSDL describing the decision service and import it into the Event Project in Event Designer.

The WSDL declares define operation signatures that are based on the Ruleset Parameters of the decision (Figure 5-9).

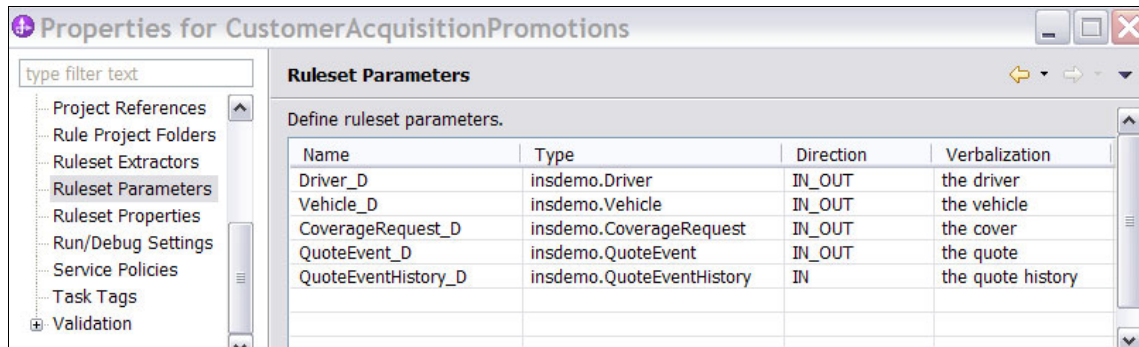


Figure 5-9 CustomerAcquisitionPromotions decision service parameters

When this web service is imported, the Service request is mapped onto an action (CustomerRequestPromotions) and the response onto an event (CustomerAcquisitionPromotionResponse). An additional event is generated for each fault in the WSDL (CustomerAcquisitionPromotionsSoapFault). The input and output parameters are then mapped onto the request action and response event (Figure 5-10).

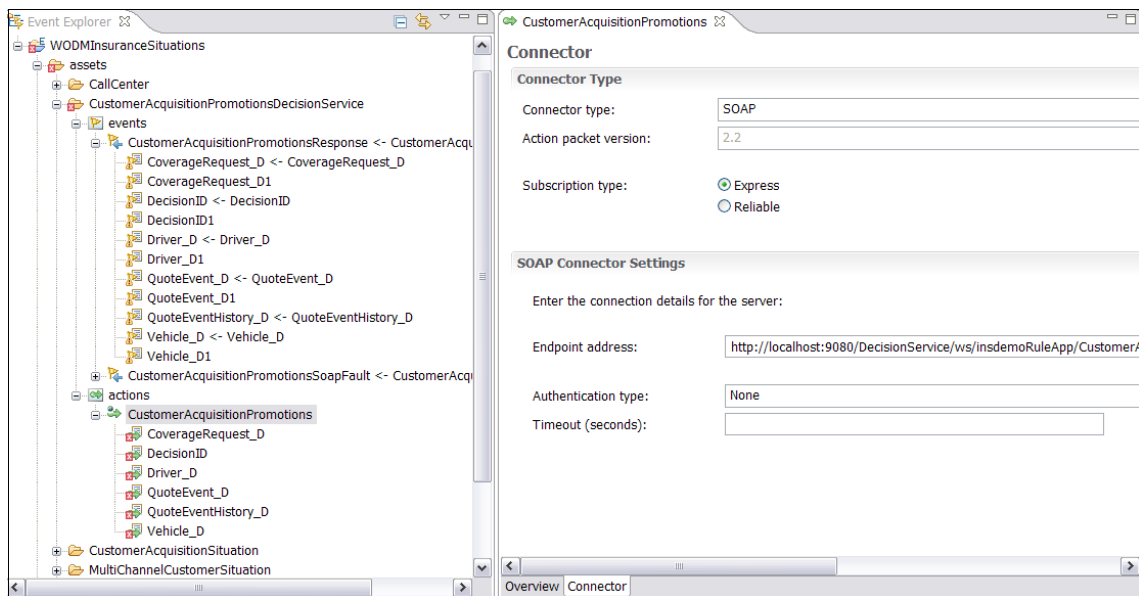


Figure 5-10 Actions, Events, and Connector settings after importing a decision service

At run time, a decision is requested by populating the CustomerAcquisitionPromotions action objects with the information to define the driver, vehicle, coverage request, and current quote (action object QuoteEvent_D) and then sending the action.

The decision response is returned as an event called CustomerAcquisitionPromotionResponse containing a modified quote in event object QuoteEvent_D1. The response event also contains a copy of the request action objects (QuoteEvent_D <- QuoteEvent_D) so that the original request is available to the response event processing.

There are a number of issues to be aware of when using this form of integration.

- ▶ The event and action object fields must *not* contain complex types or lists. This mapping limits the complexity of the decision service parameter types. If the decision vocabulary requires hierarchical types, this mapping must take place inside the ruleset.
- ▶ If a parameter is to contain a repeated object, it must contain a single member that is a Vector of the actual type required. In this case, the QuoteEventHistory type is a Vector of QuoteEvent. The representation of this member in the CustomerAcquisitionPromotions BOM is shown in Figure 5-11.

Member QuoteEventList (class: insdemo.QuoteEventHistory)

General Information

Name:

Type:

Class:

☒ Read/Write ☐ Read Only ☐ Write Only

☐ Static ☐ Final

☐ Deprecated ☐ Update object state

☐ Ignore for DVS

Member Verbalization

✗ [Remove](#) the verbalization.

+ [Create](#) a navigation phrase.

+ [Create](#) an action phrase.

+ [Edit](#) the subject used in phrases.

Navigation : "the quote events of a quote event history" ✗

Template:

Arguments

Edit the arguments of this member.

Name	Type	Domain

Domain

Create and edit a domain for this member.

+ [Edit](#) the domain.

✗ [Remove](#) the domain.

Domain type: Collection

Element type: insdemo.QuoteEvent

Cardinality: 1, *

Package | Class | Member | model.bom | model.b2x | model_en_US.voc

Figure 5-11 Representation of arrays of objects in a parameter

This integration approach allows the Situation Context state, including any information added from the database connection augmentation, to be mapped directly into the decision service request and thus into the decision vocabulary.

5.4.2 Triggering the decision from event rules

There are two options for triggering the decision service request. The first uses the Situation Synthetic Event to trigger an event rule, which then starts the decision service action (Figure 5-12).

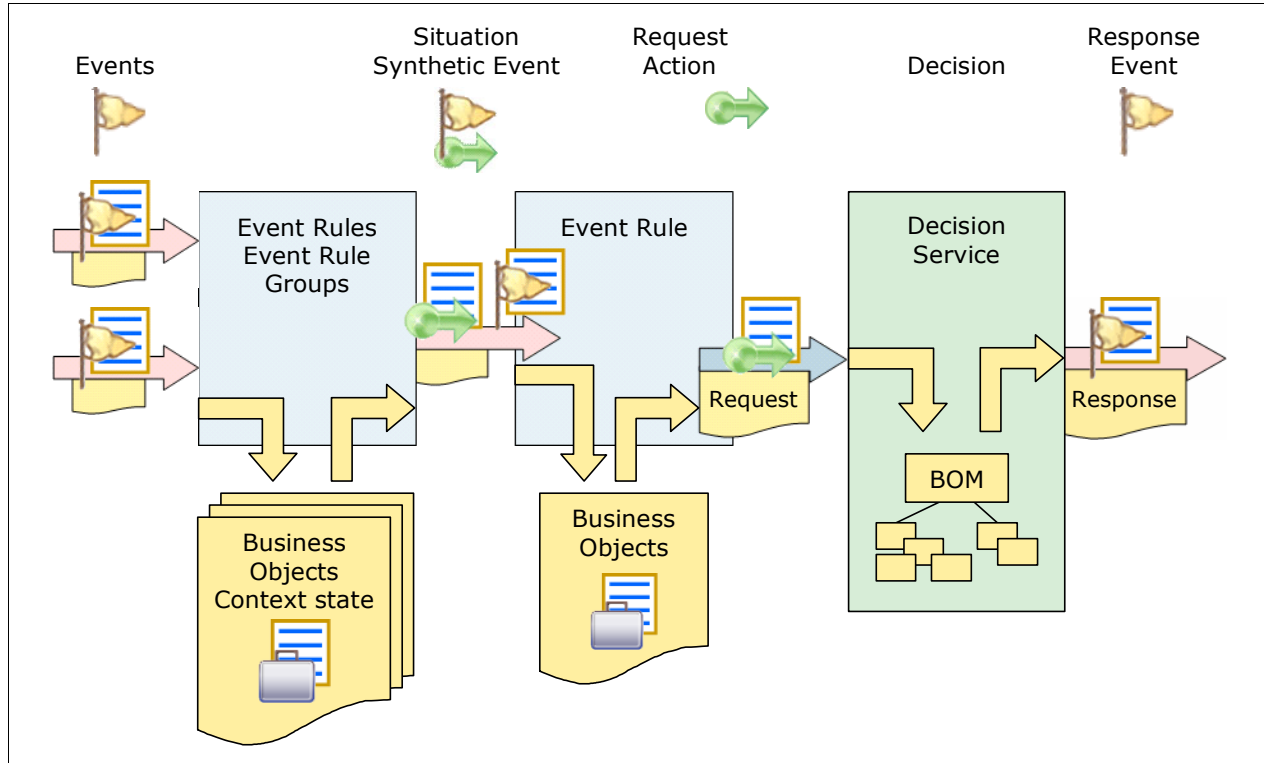


Figure 5-12 Starting the decision service from the situation

This approach is useful if you need to use the Situation synthetic event in other event rules, but it impacts the processing of the synthetic event and another set of mappings.

The decision call can alternatively replace or be combined with the synthetic event (Figure 5-13).

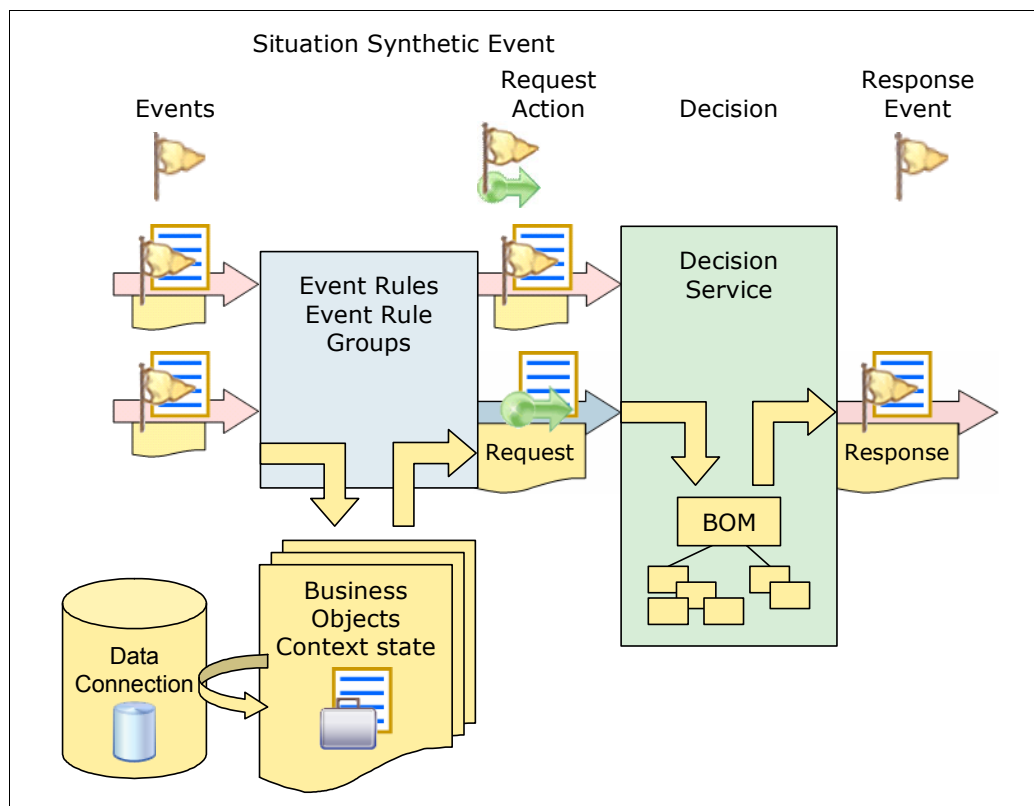


Figure 5-13 Starting the decision service from the situation identification rules

If the situation is only used to trigger the decision, then the situation context can be mapped directly to the decision service request, and you have a simplified form of Detect - Decide pattern. The Situation synthetic event can then be omitted or, if it is required, a second set of mappings can be provided.

After the decision is invoked, all the facilities of managed rule-based decision making are available. Business Users can modify how the decision is made and (in this scenario) determine how to optimize the promotions offered to individual customers. With the architecture set up, the business now can rapidly change the customer acquisition rule.

The decision response is normally returned to the calling application, which waits until the response is available. In an event-based system, there is no notion of a calling system, so the response is represented as a new event.

You now have an event coming back from the decision that is used to trigger the Respond part of the interaction.

5.5 Responding to a decision

The previous sections described how you provide business users with the control to identify situations and make decisions about what should be done in those situations. The final step allows those business users to control what actions should result from those decisions. Realizing the actions involves sending the appropriate messages to control points within the solution applications and processes.

In the WODM Insurance scenario, there are a number of possible actions, depending on the channel the customer is using to interact with the solution.

If you look at the actions that are provided through the touchpoint systems, you see that the promotion offer needs to be provided through the call center. For any customers coming through the website, you must refer them to the call center (Table 5-1).

Table 5-1 WODM Insurance solution touchpoint systems and actions

Touchpoint system	Actions	Description
Website	<ul style="list-style-type: none">▶ CustomerReminder▶ ReferToCallCenter	<ul style="list-style-type: none">▶ Display a reminder alert to customer.▶ Refer the customer to the call center.
Call center	<ul style="list-style-type: none">▶ FollowUp▶ OfferPromotion▶ InvestigateFraud	<ul style="list-style-type: none">▶ Instructs a call center operator to follow up a quote.▶ Instructs an operator to offer a quote promotion.▶ Warns an operator of potentially fraudulent quotes.

Architecturally, you are performing the same actions described in 4.6, “Using event rules to invoke actions in response to patterns of events” on page 48 and 4.7, “Mapping business objects to action data and executing the action” on page 50. The overall approach is summarized in the two event rules shown in Figure 5-14 and Figure 5-15.

Event rule: PromotionAction

▼ General Information ► Documentation

Name: PromotionAction

▼ Event Rule Context and Event

Context ID: [Change Context...](#) [Remove](#)

Event: [Change Event...](#)

Content

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
then
offer customer a promotion with message: the message of the quote ;
```

Overview

Figure 5-14 Promotion action rule to send quote promotion details to the call center

Event rule: WebPromotionAction

▼ General Information ► Documentation

Name: WebPromotionAction

▼ Event Rule Context and Event

Context ID: [Change Context...](#) [Remove](#)

Event: [Change Event...](#)

Content

Directly type in this section to create or modify your event rule definition. Press Ctrl+Space to display options available.

```
if
the channel of the quote is "Web" then
refer customer to call center with message:
"Contact Call Center who can offer: "
+ the message of the quote ;
```

Overview

Figure 5-15 Promotion action rule to send quote promotion details to the call center

In Figure 5-14 on page 72, the Promotion action rule indicates to the call center that they should offer a promotion using the message and price provided by the promotion decision. The action mapping undertaken by IT passes all the essential fields to make sure that the promotion is offered correctly.

The event rule shown in Figure 5-15 on page 72 sends a message to a customer who applies from the website, informing them that a promotion is available from the call center.

This rule uses an additional field in the "the quote" business object called "channel", which is populated from the Event field using a JavaScript field definition type (described in 5.2, "Using context variables to accumulate situation state" on page 61) (Example 5-1).

Example 5-1 Event quote

```
if ( Event.contains( "Web" ) )  
  "Web" ;  
else  
  "Call Center";
```

This example shows how the vocabulary available to the Business User can be made easier to understand by choosing suitable business object fields and verbalizing them to convey the correct meaning.

5.6 Raising events using web services

This chapter has focused so far on how to use a rule-based decision to decide what to do in a particular situation. Many solutions use decision services directly from their applications, as described in Chapter 3, "Decision design" on page 21. The rules in these decision services can often identify characteristics, patterns, or situations in the information being processed that is not relevant to the system invoking the decision, but might need to be acted upon by another system.

As an example, the website accepts only eligible quotes; all others are rejected. The Eligibility decision rules can, however, identify situations where a quote may be offered subject to manual underwriter approval.

Although code could be inserted into each application that invokes the Eligibility decision to check and perform the necessary processing, it is better to raise an event and have the event processing route that to the correct system.

To send an event to the event run time, you can provide a web service that the rules or other client raising the event can call.

In Event designer, first create an Event that is to be generated. In the WODM Insurance scenario, it makes sense to have a QuoteApproved event that uses the same event structure and objects as existing events (Figure 5-16).

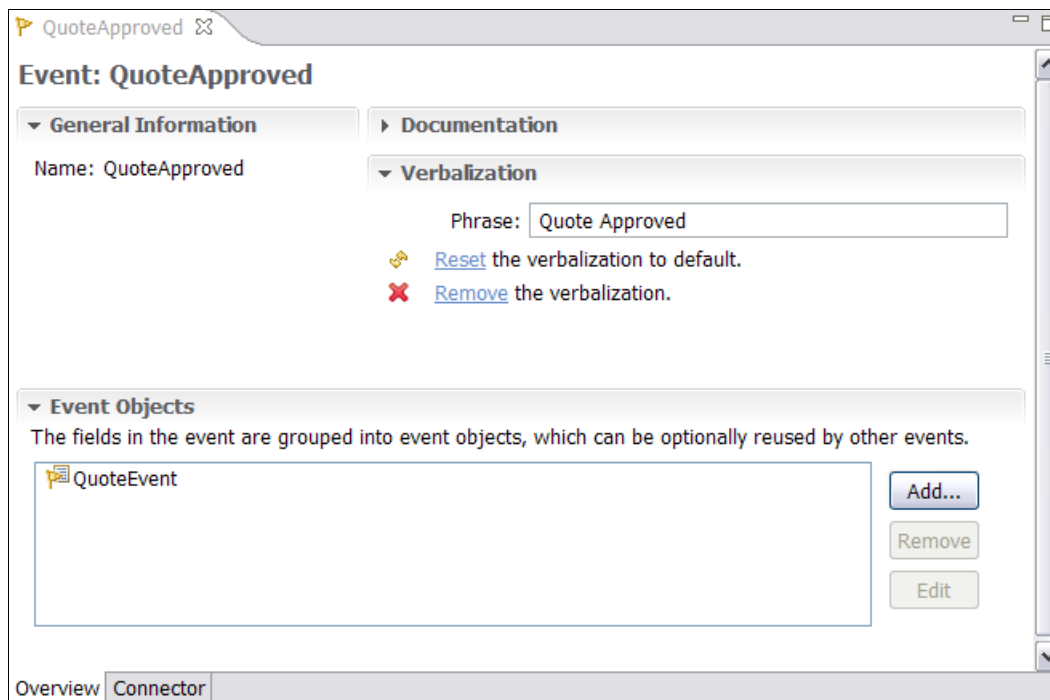


Figure 5-16 Defining the Quote Approved event to be sent from BPM to WebSphere ODM event run time

To make this event available as a web service, select the SOAP connector type in the **Connector** tab (Figure 5-17).

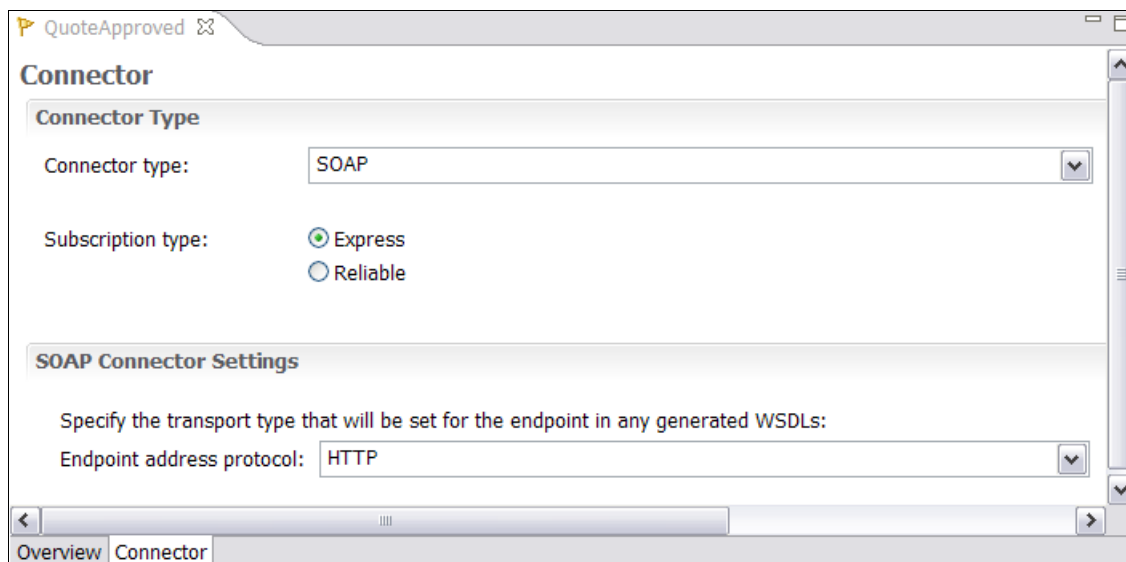


Figure 5-17 Configuring the event connector to make it available as a web service

In the WODM Insurance scenario, you have two events that are initiated through web services:

- ▶ The ManualApproval request provided from the rules projects
- ▶ The QuoteApproved event from BPM.

By combining these events in the same folder (Figure 5-18), we can group them into a single web service.

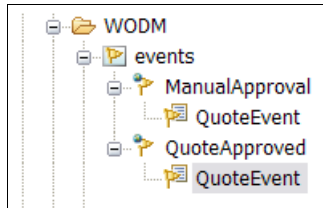


Figure 5-18 Grouping events in a folder to form web service operations

After the connectors are configured and the project deployed the WSDL, describing the service can be obtained from the run time from the following URL:

`http://<host>:<port>/wbecasoap/SOAPEventWSDL?folder=<folder>`

If this WSDL is downloaded and viewed in a WSDL editor, the structure of the service can be seen in Figure 5-19 as a one-way call containing the event payload.

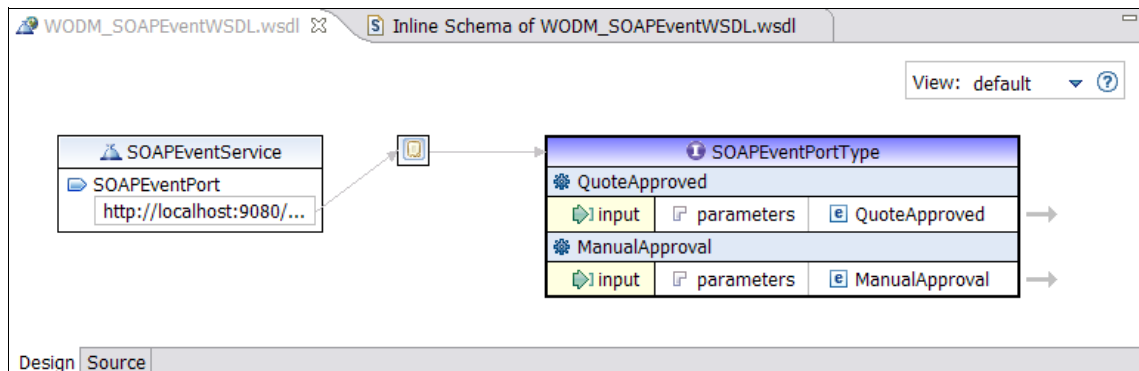


Figure 5-19 Web Service signature provided by exposing events use SOAP connector

The payload and structure of the events can be seen from the inline schema provided in the WSDL (Figure 5-20).

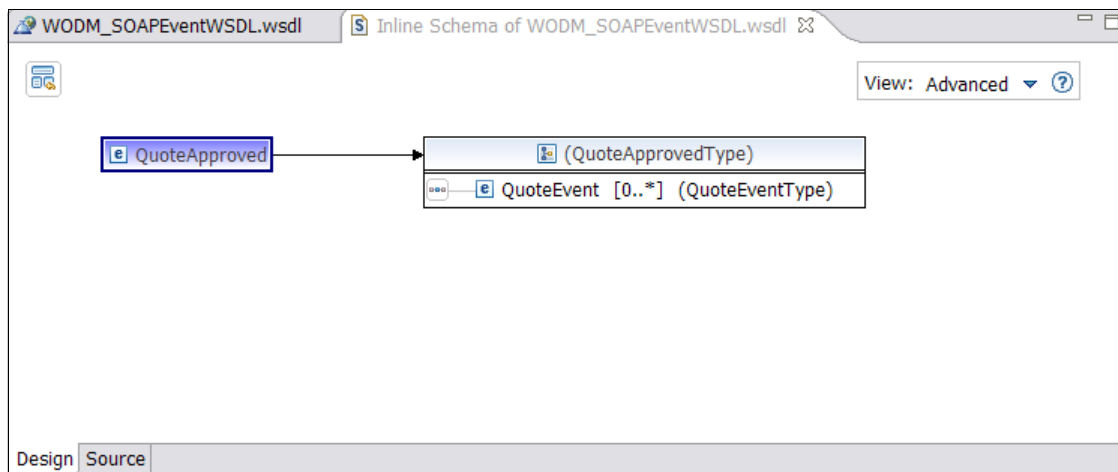


Figure 5-20 Web Service signature showing repeated event payload

The detailed fields can then be seen to match those fields in the original definition (Figure 5-21).

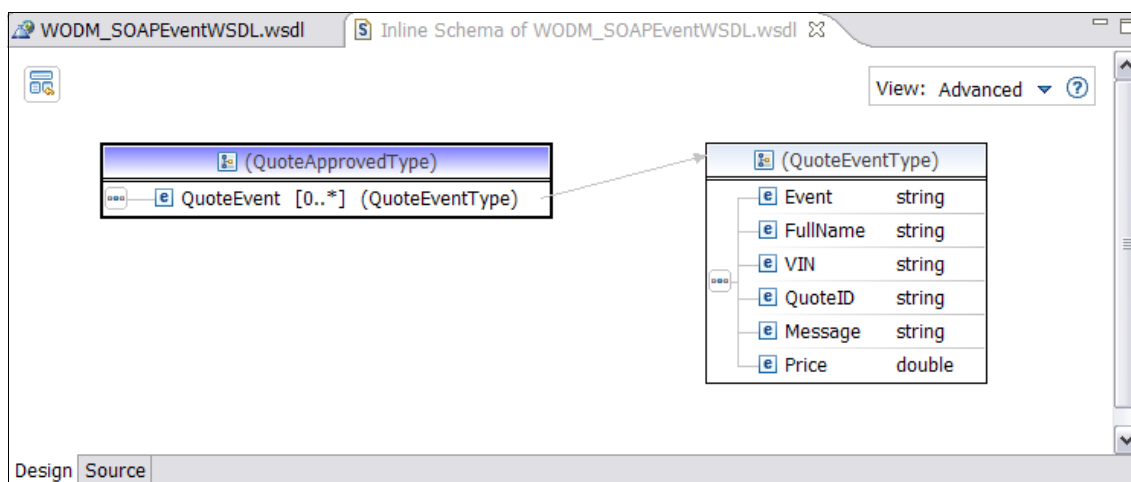


Figure 5-21 Structure of individual event payload

This web service URL can then be started by an application or custom rules project to send the event. This section describes how to create an event that can be raised by an external application or decision using web services. It is also possible to raise an event using JMS messages. In this case, the event must be created in the form that the event run time understands and then published to a JMS topic. Event designer provides generated resources that define both schema definitions and sample XML files for the message sent to the JMS topic (Figure 5-22).

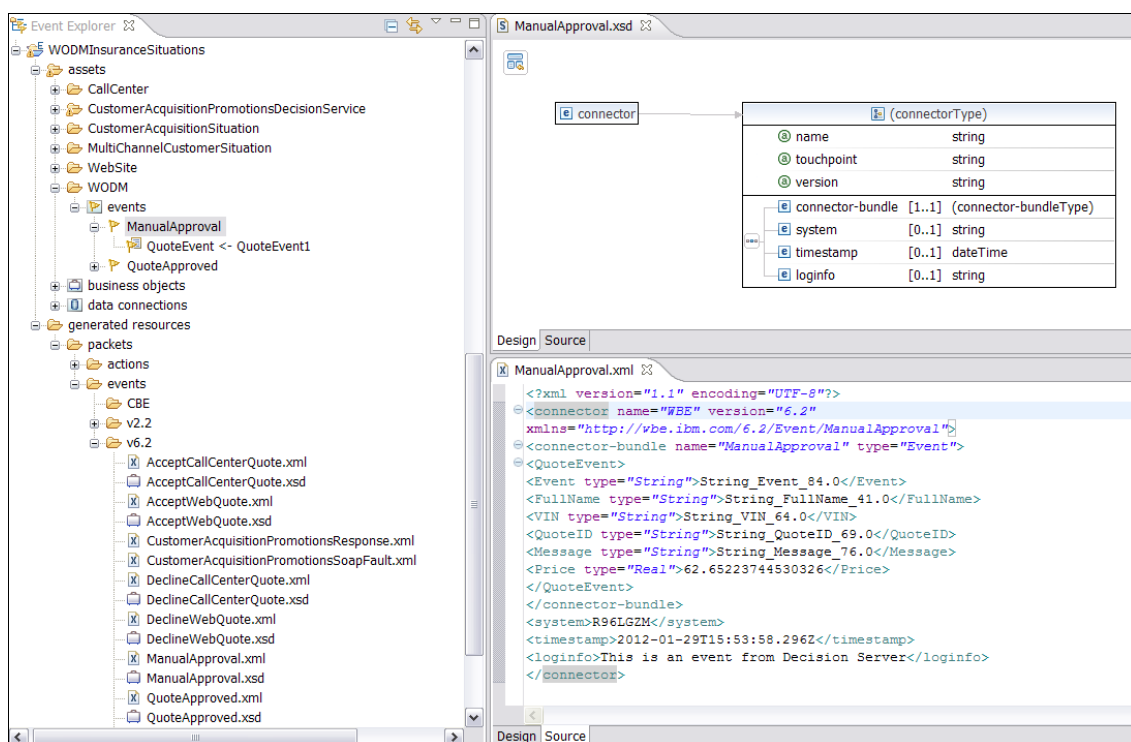


Figure 5-22 Runtime representation of Event Message provided in Event Designer

Use this representation of the event in 5.7, “Raising events from within rules” on page 77 to show how to publish an event as part of a rules-based decision.

5.7 Raising events from within rules

The usage of decision management products such as WebSphere ODM means that decision making gets concentrated around reusable decision services. Although there might be different client applications invoking these decision points, the role of the decision in the business is consistent. The nature of decision services is that they provide a request - response pattern to their client (Figure 5-23).

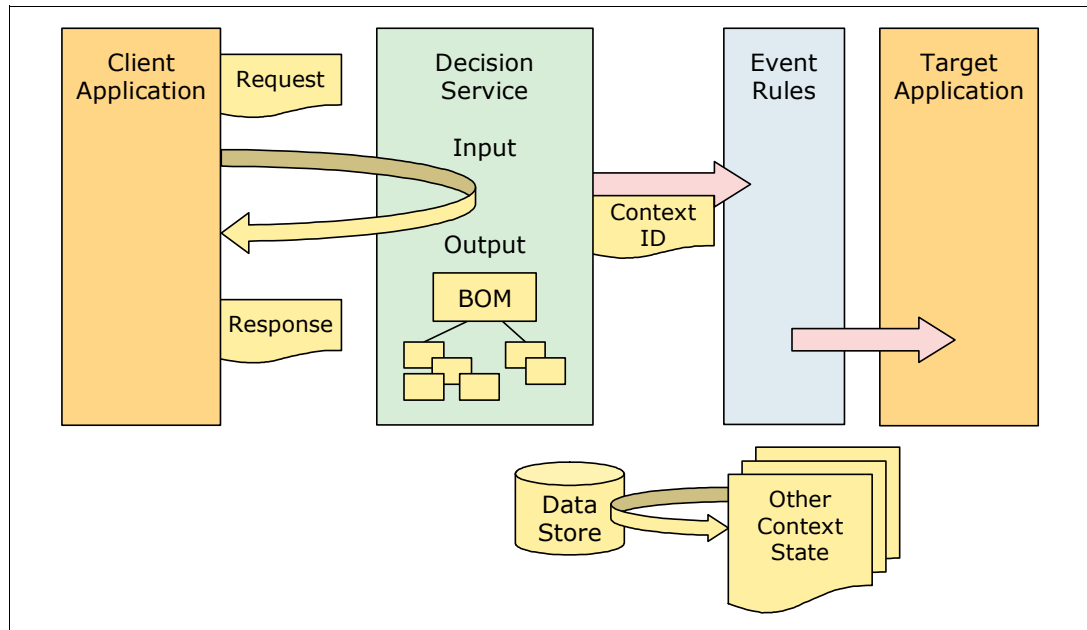


Figure 5-23 Raising events from within decision services

When a client application makes a call to a decision service, it needs to contain all the information needed for the decision, but probably does not contain any other state, including the context in which the decision was made. In the WODM Insurance scenario, an Eligibility decision can be made without knowing if the customer came in on the website or the call center.

The rules can also be used to detect situations that the client application cannot handle. In this scenario, the manual approval cannot be handled by the website or call center.

The solution is for the Decision Service to send an event to the run time, passing in enough context for the event processing to acquire additional information and perform the action (in this case, starting a new Underwriting Approval Process).

The essential information in this case is the driver name, the VIN, and the quote ID generated by the Eligibility decision. As the Decision Warehouse stores all the input and output parameters, this quote ID is sufficient to retrieve any other information needed. The driver name and VIN are needed to allow the event processing to correlate with other events for the same user or vehicle and thus associate the Eligibility decision with the broader correlation context.

The use of web services to raise an event is described in 5.6, “Raising events using web services” on page 73. In this section, you generate a JMS message and send it directly to the event run time. In this scenario, we describe the generation of the "ManualApproval" event from the eligibility rules. This scenario assumes that the structure of the "ManualApproval" event is declared in the schema being used in the Business Object model available to the Eligibility rules.

5.7.1 Creating a JMS event publishing client

The first step is to create a JMSEventClient that can be started from within the rule vocabulary. This action takes the form of a Java project that must be built as a JAR file and deployed into the Rule execution server class path together with any other JAR files that the project depends on. This client code consists of a single class with three methods (Figure 5-24).

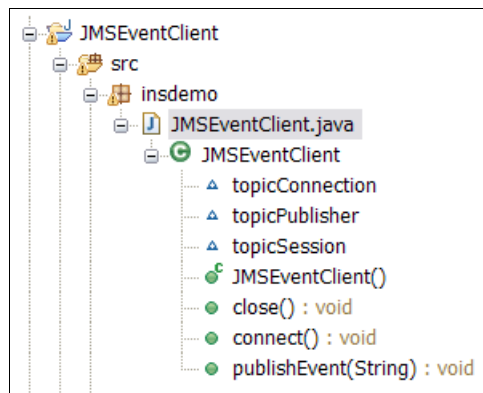


Figure 5-24 JMS Event Client Java Project

The **connect()** method first establishes a connection to the JNDI context that can resolve the Decision server JMS resources needed to make a connection. The values shown in Example 5-2 assumes deployment on a local WebSphere Application Server using default ports. When connecting to a remote server with security enabled, the environment properties also need to include an administrator user name and credentials. When run from within Decision Server (in deployed rules), the default initial context should be sufficient, and the environment properties should not be needed.

Example 5-2 Establishing a connection to a JNDI context

```
Hashtable<String, String> env = new Hashtable<String,String>();
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
env.put(Context.PROVIDER_URL,
"iiop://localhost:2809/");
Context jndiContext = new InitialContext(env);
```

The two resources that need to be resolved are the Connection Factory and the Topic to which the event must be published. These resources can be resolved using JNDI (Example 5-3). The values shown are the ones used in a default installation of the event run time.

Example 5-3 Default resource values

```
String DEFAULTCONNECTIONFACTORY = "jms/WbeTopicConnectionFactory";
String DEFAULTPUBLISHTOPIC = "jms/eventDestination";
```

```
TopicConnectionFactory topicConnectionFactory =  
(TopicConnectionFactory)  
jndiContext.lookup(DEFAULTCONNECTIONFACTORY);  
Topic topic = (Topic) jndiContext.lookup(DEFAULTPUBLISHTOPIC);
```

After references to these resources are obtained, a connection and session can be established and a publisher is obtained that can send the event messages to the Decision server event run time (Example 5-4).

Example 5-4 Establishing a connection and session

```
topicConnection =  
topicConnectionFactory.createTopicConnection();  
topicSession =  
topicConnection.createTopicSession(false,  
Session.AUTO_ACKNOWLEDGE);  
topicPublisher =  
topicSession.createPublisher(topic);
```

This code then establishes a connection to the event run time. When designing the client, it is worth considering the lifetime of the connection. If the client or publisher might be used several times during the processing of a decision, then the connection and session should be established at the beginning of the decision rule flow and closed at the end when all events are sent. To close the session, use the `close()` method using the code similar to the code shown in Example 5-5.

Example 5-5 Closing the session using the close() method

```
public void close() {  
    try {  
        topicSession.close();  
        topicConnection.close();  
    } catch (JMSEException e) {  
        e.printStackTrace();  
    }  
}
```

While the session is open, the client can be used to send event messages to the Decision Server event run time. This action can be exposed as a method on the client (Example 5-6).

Example 5-6 Sending event messages

```
public void publishEvent(String eventMessage) {  
    try {  
        TextMessage message = topicSession.createTextMessage();  
        message.setText(eventMessage);  
        topicPublisher.publish(message);  
    } catch (JMSEException e) {  
        e.printStackTrace();  
    }  
}
```

This Java project has dependencies on the JNDI and JMS libraries that must be included in the class path for testing and debugging in an Eclipse environment. The server hosting the event run time also needs to be running on the host and port configured in the JNDI connection.

5.7.2 Exposing the JMS client in the rules business object model

This JMSEventClient can then be used to create a Java XOM in the Business object model and expose the methods to the decision making (Figure 5-25).

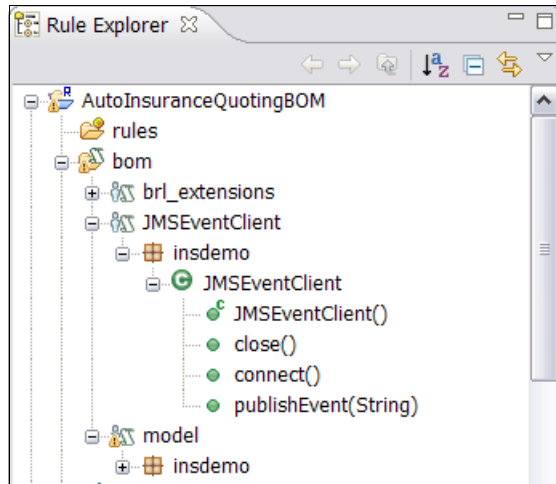


Figure 5-25 JMS Event Client exposed within the decision Business Object Model

This task can be accomplished by specifying the JMSEventClient project as a Java XOM and then creating a BOM from that XOM. This action automatically (as of WebSphere ODM V7.5) includes the compiled Java code in JMSEventClient when a Rule App archive is produced from the ruleset. The referenced libraries for JNDI and JMS should already be on the server class path if they are running in WebSphere, so they do not need to be included.

5.7.3 Verbalizing the means to publish an event

The JMSEventClient operations are not verbalized or exposed to the business user, as they must be used carefully to ensure that the structure of the event messages corresponds to the structure required by the event run time. The simplest way of accomplishing this task is to define a class in the business object model that represents each event message that is required (Figure 5-26).

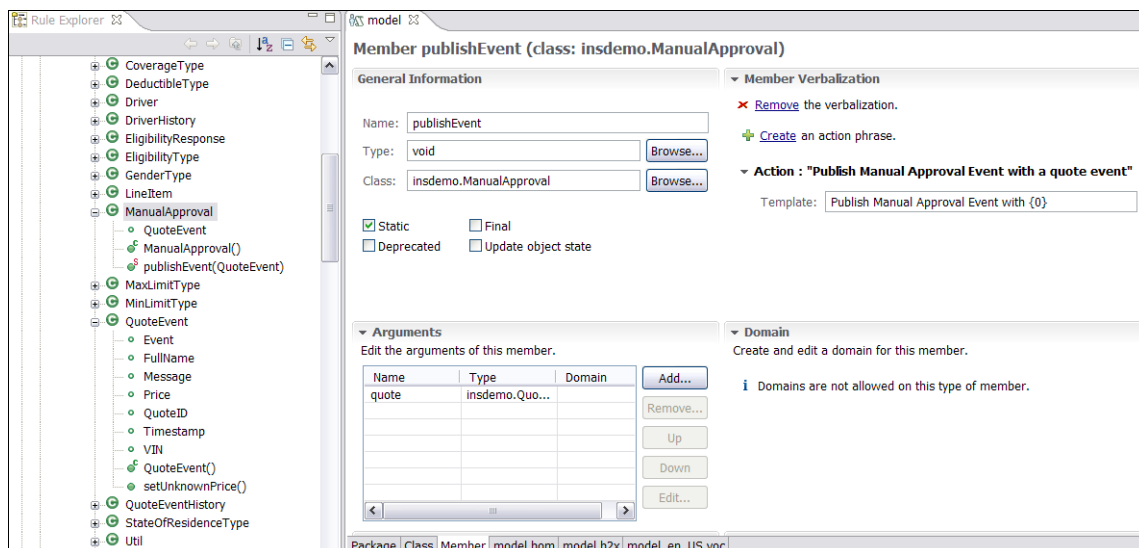


Figure 5-26 Manual approval event representation and publish verbalization

There are a number of approaches that could be used to verbalize the scheduling of an event. In this case, a static method (`publishEvent`) is provided on the `ManualEvent` class that takes as an argument the objects needed to provide the information contained in the event. The information is the summary information defined in a quote (`QuoteEvent` class). Using this representation allows rules to be inserted into the rule flow that define the information needed and schedule an event using a natural expression (Figure 5-27).

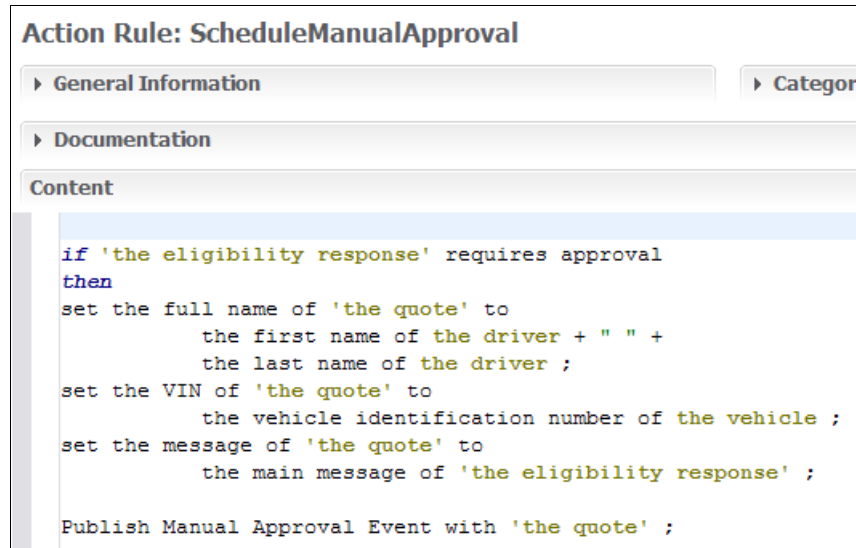


Figure 5-27 Scheduling an event from within rules using the event verbalization

When this rule fires, the `ManualApproval.publishEvent()` method is invoked, which then uses the IRL rule language to define the event message in the form needed by the event run time and to send the event to the `JMSEventClient`. This mapping is undertaken by the Rule developers using rule studio and defined in the `BomToXom` mapping window for this method (Figure 5-28).

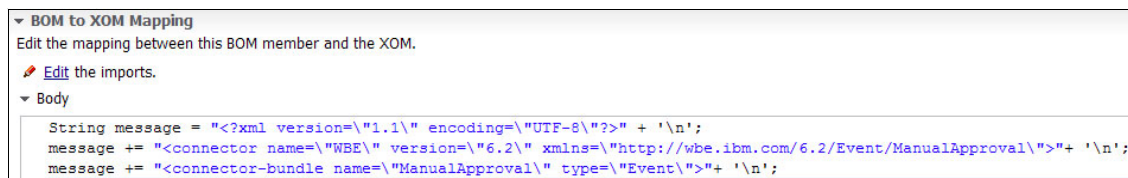


Figure 5-28 IRL definition of `publishEvent()` method

The `BOM to XOM Mapping` window contains Java expressions (IRL) to define the event message. The objective is to set up a message that is in the form needed by the event run time. The first part of this message defines the Manual Approval event in the form available from the Event designer. The namespace and connector bundle name both use the event name to define them (Example 5-7).

Example 5-7 Defining the event message

```

String message = "<?xml version=\"1.1\" encoding=\"UTF-8\"?>" + '\n';
message += "<connector name=\"WBE\" version=\"6.2\" " +
    " xmlns=\"http://wbe.ibm.com/6.2/Event/ManualApproval\">" + '\n';
message += "<connector-bundle name=\"ManualApproval\" type=\"Event\">" + '\n';

```

After the Event is defined, the event objects (in this case, a QuoteEvent) are defined (Example 5-8).

Example 5-8 Defining the event objects

```
message += "<QuoteEvent>" + '\n';
message += "<Event type=\"String\">" + quote.Event + "</Event>" + '\n';
message += "<FullName type=\"String\">" + quote.FullName +
"</FullName>" + '\n';
message += "<VIN type=\"String\">" + quote.VIN + "</VIN>" + '\n';
message += "<QuoteID type=\"String\">" + quote.QuoteID + "</QuoteID>" + '\n';
message += "<Message type=\"String\">" + quote.Message + "</Message>" + '\n';
message += "<Price type=\"Real\">" + quote.Price + "</Price>" + '\n';
message += "</QuoteEvent>" + '\n';
```

Each field in the event needs to have a type attribute that corresponds to the representation of the field in the event run time. By using this approach, the mapping can select the fields required for each event object from the rule variables passed into the method.

After the event objects are mapped, the event message must be completed by completing the connector bundle and connector fields (Example 5-9).

Example 5-9 Completing the event message

```
message += "</connector-bundle>" + '\n';
message += "<system>Decision Server</system>" + '\n';
DateFormat dfm = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS'Z'");
IlrDateTime ts = new IlrDateTime(System.currentTimeMillis());
Date a = ts.toDate() ;
message += "<timestamp>" + dfm.format(a) + "</timestamp>" + '\n';
message += "<loginfo>This is an event from Decision Server</loginfo>";
message += "</connector>";
```

This example also shows some of the mapping techniques available to convert from the rules representation of dates and times into a form compatible with the event run time. Usage of these Java classes requires the usage of import statements for the ManualApproval class (Figure 5-29).

The screenshot shows a configuration window for the **ManualApproval** class. The **General Information** tab is active, showing the class name, namespace (**insdemo**), and superclasses (**java.lang.Object, ilog.rules.xml.IlrXmlObject**). The **Class Verbalization** section shows a term of **manual approval**. The **BOM to XOM Mapping** section is expanded, showing the **Imports** list with the following code:

```
import java.text.SimpleDateFormat;
import java.util.Date;
import ilog.rules.xml.types.IlrDateTime;
import insdemo.QuoteEvent;
import insdemo.JMSEventClient;
```

Figure 5-29 ManualApproval BOM to XOM mapping imports

These imports include **JMSEventClient**, which then allows the method to be completed by connecting to the run time and sending the event (Example 5-10).

Example 5-10 Sending the event

```
JMSEventClient client = new JMSEventClient();
client.connect();
client.publishEvent(message);
client.close();
```

This final bit of the method **BtoX** code creates a **JMSEventClient**, connects to the event run time and establishes a session, sends the event message that has been constructed, and then closes the connection. Although this solution is not the most elegant one, it shows the essential steps needed to schedule an event from within a decision.



Business process design with rules and events

This chapter describes how to use rules-based decision making and event processing within business processes.

Chapter 2, “Development process overview” on page 9 provides an overview of the key tasks needed when designing a process. This chapter provides practical illustrations of how the WODM Insurance scenario processes are integrated with the Decisions and Situations described in earlier chapters.

6.1 Overview

This chapter uses an extension of the scenario described in previous chapters based around the underwriting approval solution shown in Figure 6-1.

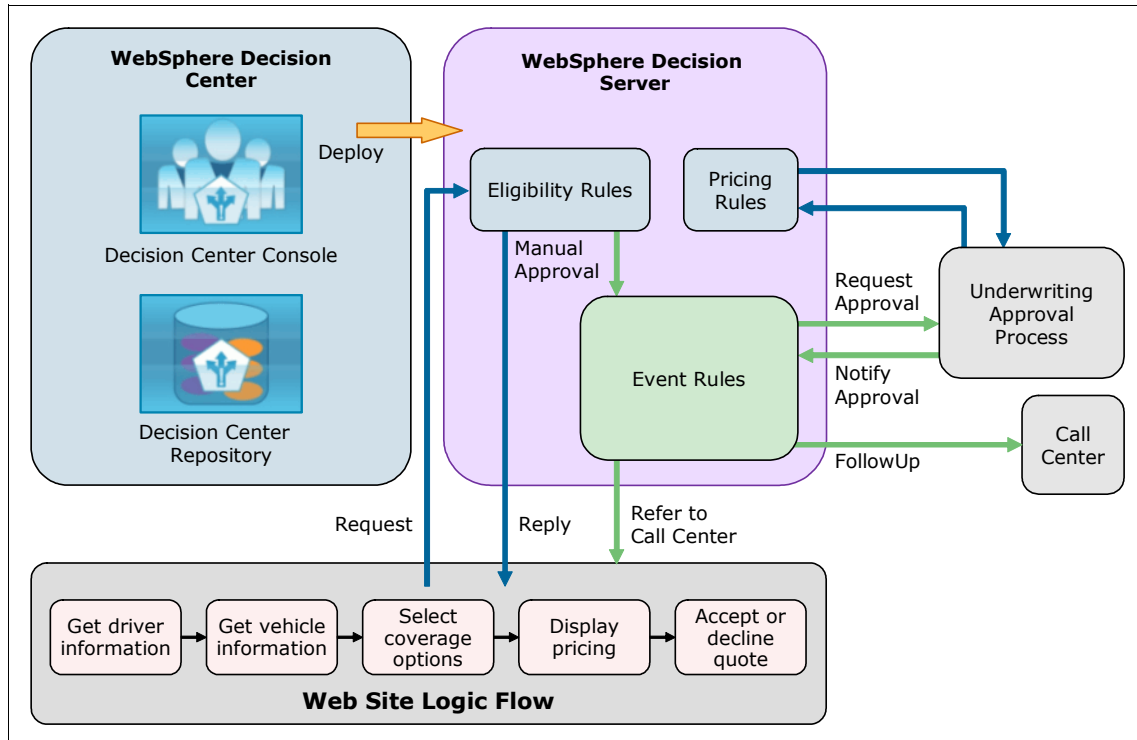


Figure 6-1 WODM Insurance underwriting approval solution

The rules in the Eligibility decision indicate if the quote request is eligible, ineligible, or requires manual approval. If manual approval is required, an event is generated indicating the quote request to be approved.

This Eligibility decision can be stored in the decision warehouse (or other database) so that the details of the decision response do not need to be contained within the event, which requires more bandwidth and memory in the event processing.

The Event rules first raise alerts to inform the customer that a manual approval is required and refer them to come back to the call center later.

An event is then sent to the Underwriting Approval Process to initiate a new approval. On completion of the approval (or rejection), an event is then sent back to the Event rules, including the new quote identity and status. The call center is then alerted to follow up with the customer to close the approved quote or to explain the rejection.

The Underwriting approval process is then followed, as described in 6.2, “Identifying the underwriting approval process activities” on page 87.

The remaining sections in this chapter describe how each step of this solution can be implemented.

6.2 Identifying the underwriting approval process activities

The underwriting approval process is based on supporting the original WODM Insurance application process where applications were received by post, keyed into the system, and then taken through the Eligibility decision and approvals before pricing the quote and notifying the applicant. The introduction of the website and call center automated most quote capture and eligibility assessment, but there are still situations when the approval process must be applied to the quote applications received from these channels.

This process and its activities can be captured in Process Designer (Figure 6-2).

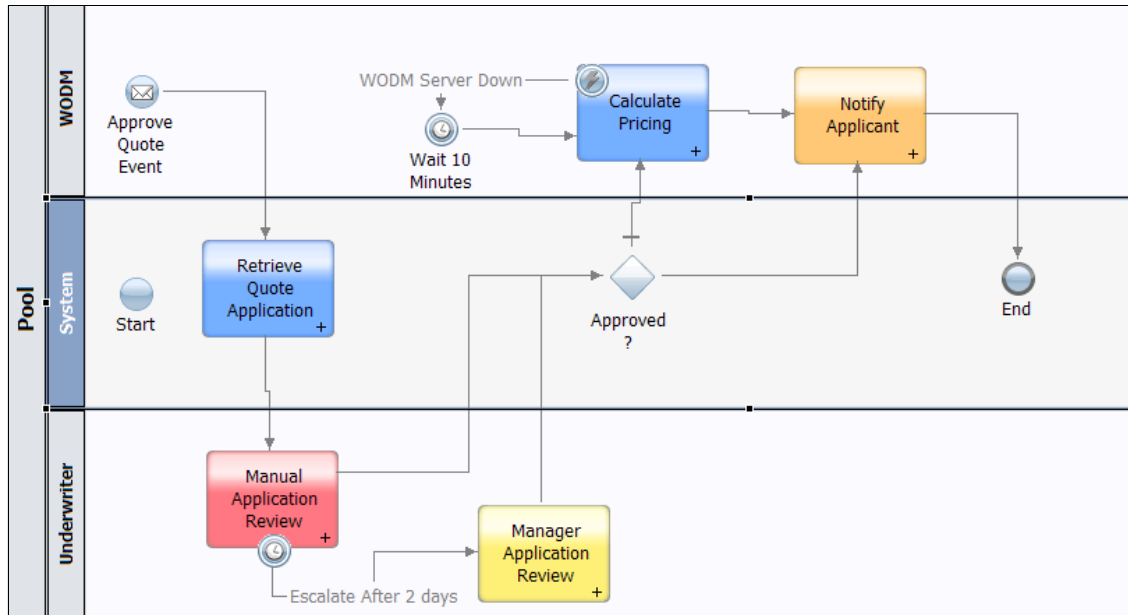


Figure 6-2 Underwriting approval process BPMN diagram

The process is triggered from an Approve Quote Event, which provides a key to the driver, vehicle, and coverage request details. The Retrieve Quote Information service retrieves the information needed from the database or decision warehouse to allow the review to occur. This process is described in more detail in 6.6, “Providing the information for the manual review” on page 101.

The Manual Application Review activity routes the quote application and eligibility rationale to an underwriter who either approves the application or rejects it, providing a reason for the rejection.

If the assigned Underwriter fails to respond to the task within two days, an escalation is raised to send the application to the Manager Application Review activity, allowing the manager to make the decision instead.

The “Approved ?” gateway determines which path should be taken through the following process:

- ▶ An “Approved” response routes the quote application to the Calculate Pricing activity.
- ▶ A “Rejected” response routes the quote application directly to the Notify Applicant Service together with the reason provided by the Underwriter.

The Pricing Decision service is the same one you used for the website and Call Center. After the price is calculated, a new quote is generated and details are stored in the quote database or decision warehouse. The means of invoking the decision service are described in 6.5, “Triggering a BPM process from a WebSphere ODM action” on page 95.

Finally, the Notify Applicant service schedules an event to the Event Runtime to allow the decision and price notification to be routed to the applicant by the most appropriate channel.

6.3 Defining decision services in a BPM process

This section describes the general approach to using decision points within a process.

The simplest approach to invoking decisions from BPM is by using Hosted Transparent Decision Service (HTDS). Process Designer supports decision discovery and import together with the mapping from process variables to the information used to make the decision.

The first step is to define a decision in the Process App. In this case, it is the Pricing Decision that is invoked after the quote is approved. You insert a JRules Decision service into this decision. This action allows the HTDS decisions that are deployed on a WebSphere ODM Decision Server to be discovered and imported (Figure 6-3).

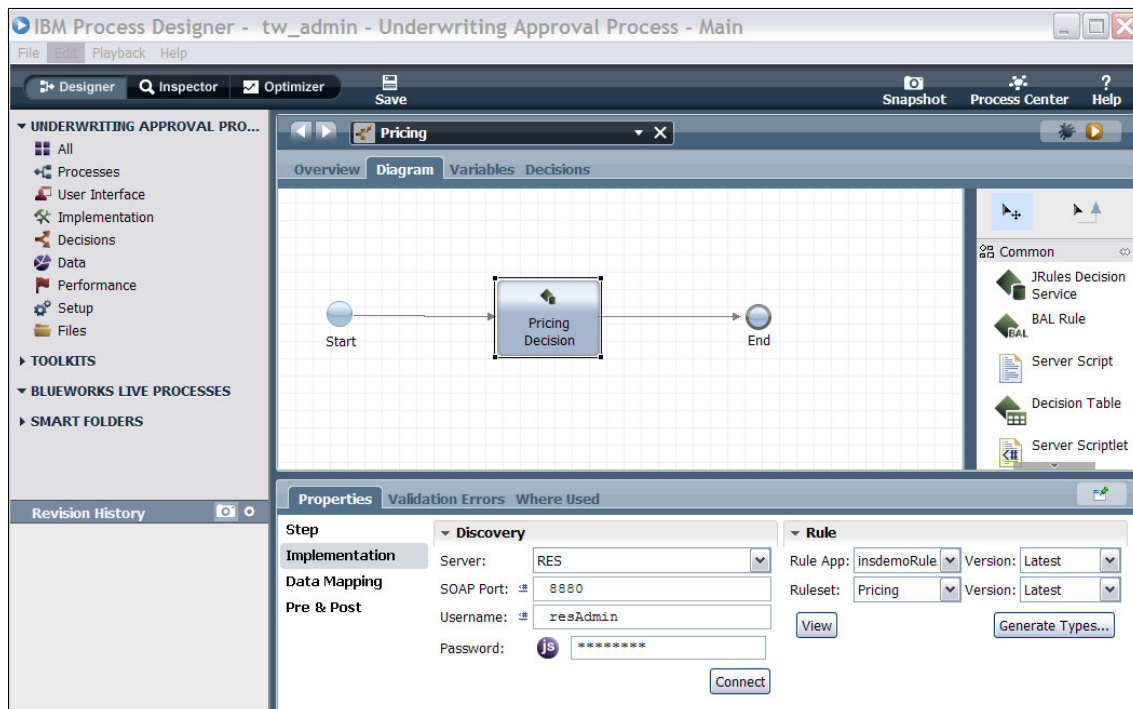


Figure 6-3 Process Designer Decision showing the rule discovery capabilities

To use the decision discovery capabilities, you need to register the server hosting the HTDS decision (RES in this case) and then click **Connect**. This action displays a drop-down menu in the Rule window that allows you to select the Rule App and Ruleset that is required together with the version.

Clicking **Generate Types** imports the information model needed to make that decision into the Process designer Process App.

Select the decision operation required, and the wizard generates all the types required based on the schema provided in the decision service WSDL (Figure 6-4). This wizard now reflects the BOM classes used in the decision.

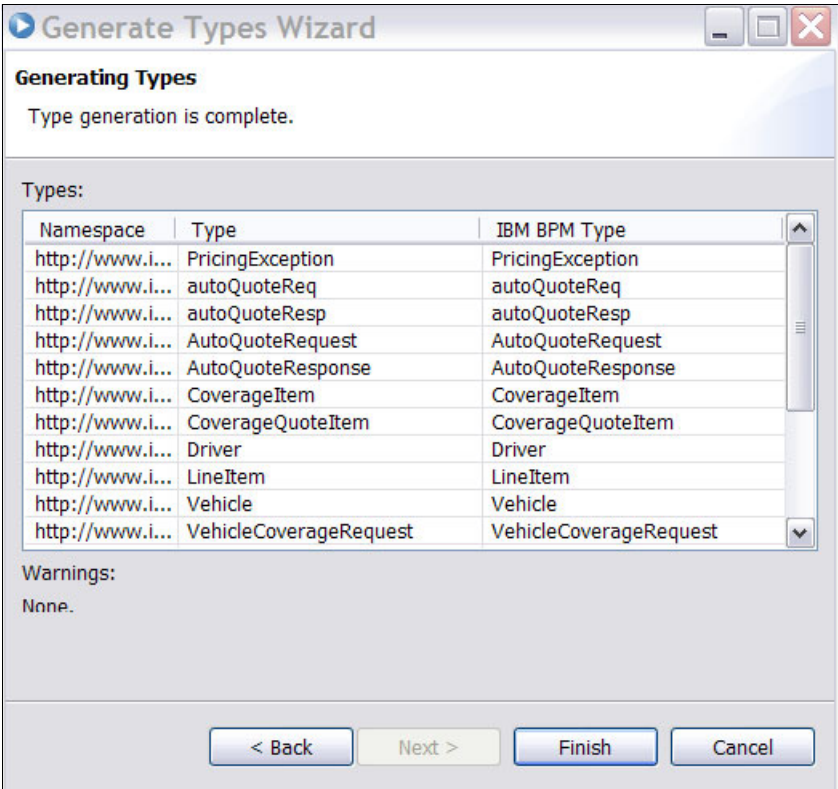


Figure 6-4 Process Designer decision service import wizard type generation

When you click **Data** → **Business Objects**, you see that the important classes are imported into the Process App. The AutoQuoteRequest Business Object shown in Figure 6-5 identifies the information that needs to be sent to the Pricing Decision.

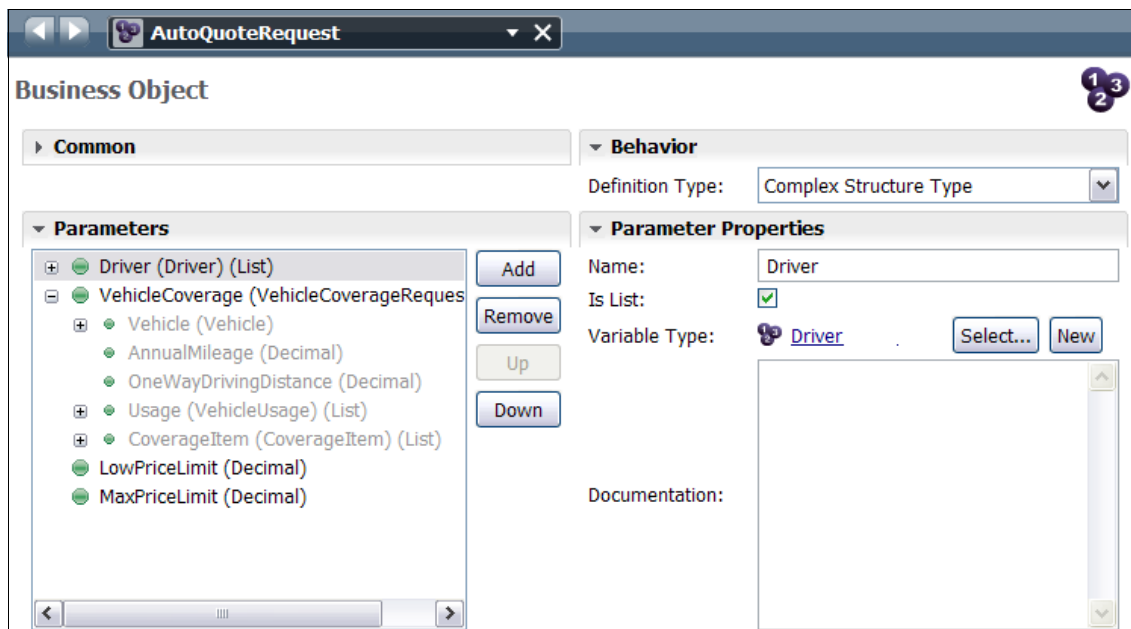


Figure 6-5 Process Designer Business Objects created from the decision import

The next step is to define the signature to the decision and map it to the HTDS decision service. In the Pricing decision, we map the Decision service signature into an identical decision signature within BPM (Figure 6-6).

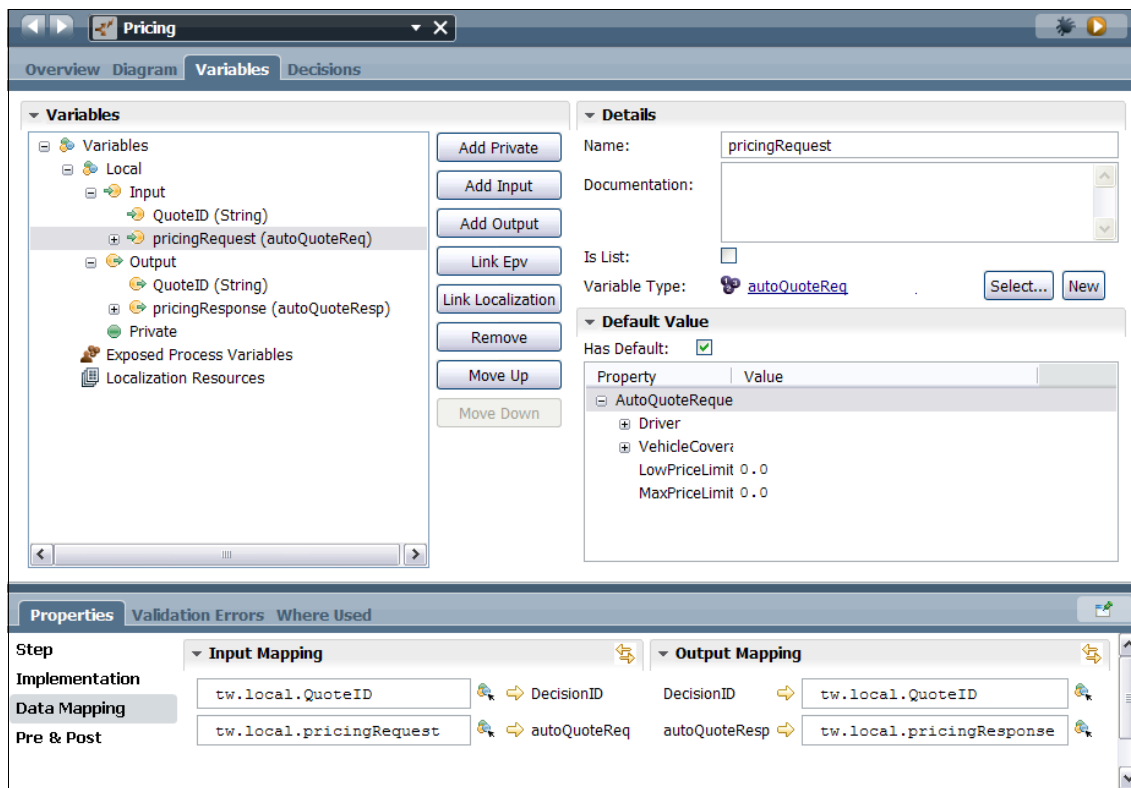


Figure 6-6 Process Designer Decision signature definition and mapping to WebSphere ODM decision service

The pricingRequest parameter is typed as an autoQuoteReq element, which contains the main AutoQuoteRequest business object. This extra layer of wrapping can be removed using a combination of the data mapping capabilities and pre- and post-scripts. It might be desirable to provide a specific signature for the Process Designer decision that better reflects the shared interface exposed to the rest of the process.

The decision is now available to use within the process and can be tested locally by inserting values into the pricingRequest input parameter and clicking the **Debug** or **Run** icons.

6.4 Initiating a BPM process from an event

The first part of initiating the process is to provide a means to initiate a process from an Internal BPM event. After this action is possible, you create the BPM event from WebSphere ODM.

The first step is to define the context information available in the event. This information is passed to the process when it is initiated. Using a Business Object is the best way to define this structure (Figure 6-7).

The screenshot shows a web browser window titled 'QuoteEvent'. The main content area is titled 'Business Object'. Under the 'Common' tab, the 'Name' field is 'QuoteEvent', the 'Modified' field is 'tw_admin (Dec 15, 2011 11:24:05 PM)', and the 'Documentation' field contains the text 'Minimal key information available for a quote event'. Below this, the 'Parameters' tab is active, displaying a list of parameters: 'QuoteID (String)', 'FullName (String)', 'VIN (String)', 'Price (Decimal)', 'Message (String)', and 'Status (String)'. To the right of the list are buttons for 'Add', 'Remove', 'Up', and 'Down'.

Figure 6-7 Defining a Business Object to represent the event payload

The main event handler in BPM is an Undercover Agent (UCA). The structure of the events it handles needs to be defined by a service signature in a General System Service (Figure 6-8).

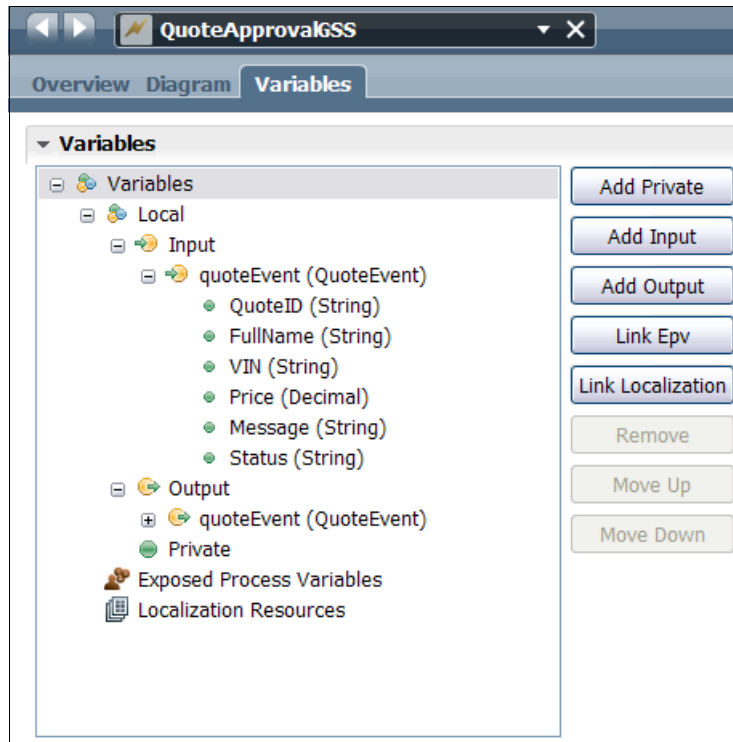


Figure 6-8 Defining a General System Service to process the event payload

The next item to create is the Undercover Agent referencing the QuoteApprovalGSS (Figure 6-9).

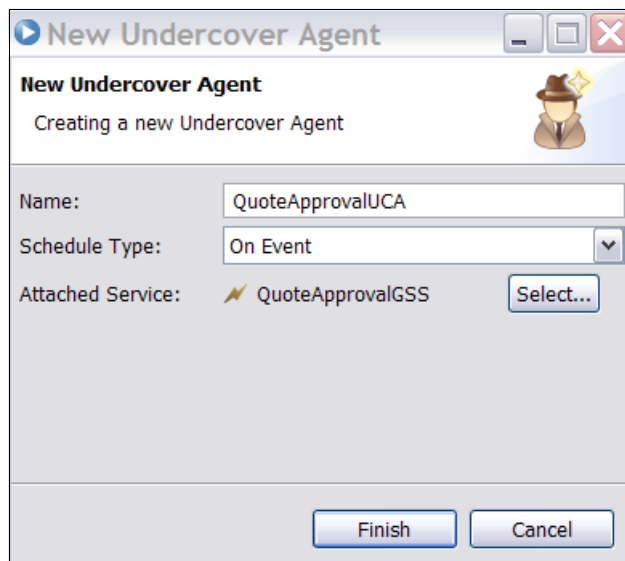


Figure 6-9 Defining an Undercover Agent to initiate a BPM process

The UCA can now schedule an event that you use to start the Underwriting Approval Process.

In the Underwriting Approval Process, create a local variable called quoteEvent to accept the event when it arrives. Select the Start Message event that initiates the process and attach the UCA that was created. Map the UCA event to the local process event (Figure 6-10).

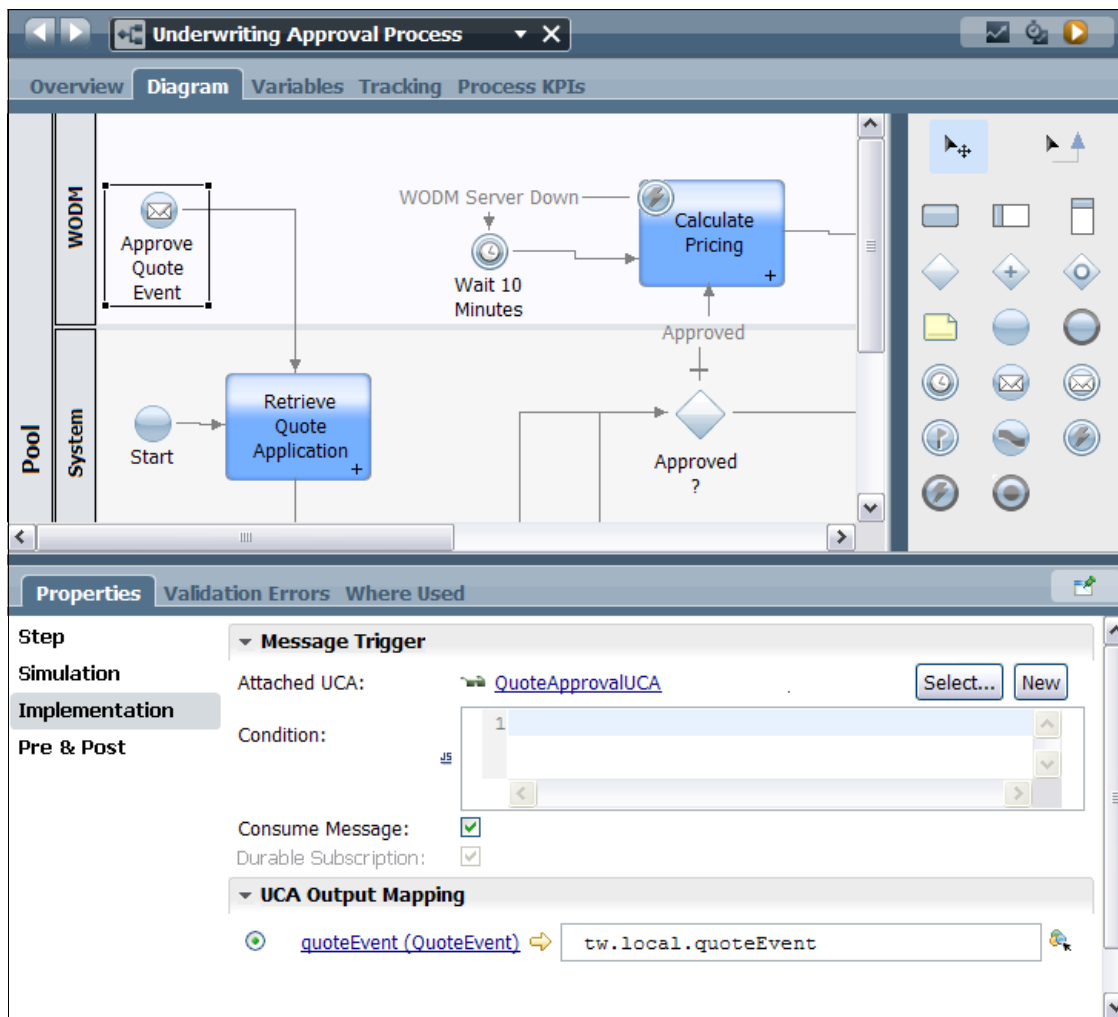


Figure 6-10 Configuring the process to respond to the UCAage - defining a Start Message

When the UCA is triggered, the process should start.

6.5 Triggering a BPM process from a WebSphere ODM action

The Undercover Agent needs to be triggered from outside the BPM process as a result of an action from the WebSphere ODM event processing. Working backwards from the UCA, the steps needed to provide this integration are shown in this section.

The UCA needs to be included in an integration service (Figure 6-11).

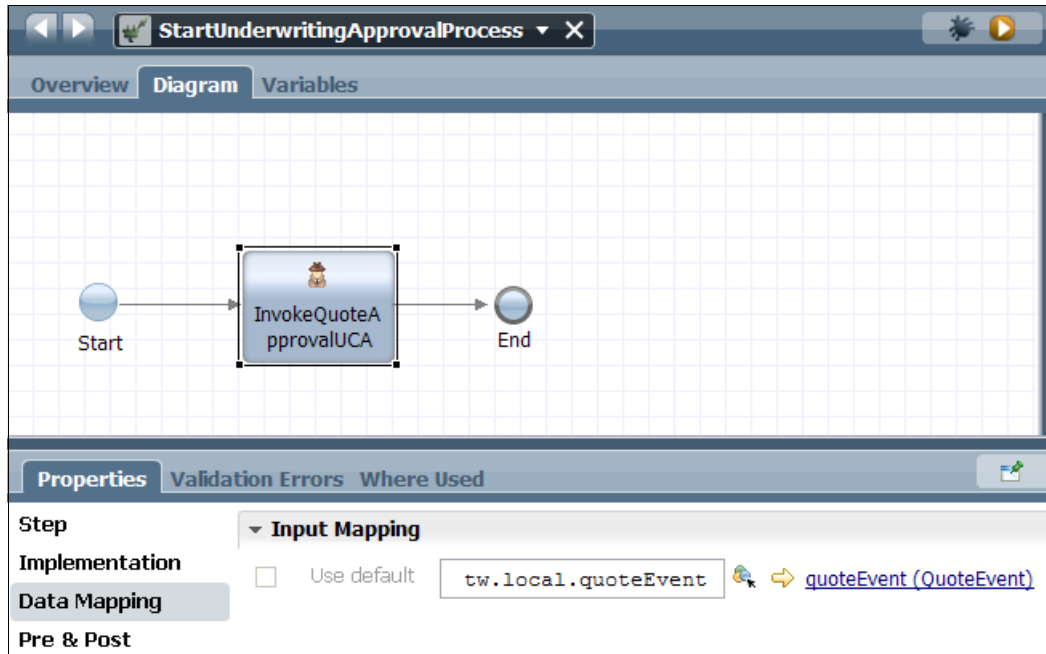


Figure 6-11 Defining an integration service to trigger the UCA

There is an input parameter to this service containing the event fields to be placed in the UCA event. This service is then added as an operation to a web service that can be exposed outside of Process Server (Figure 6-12).

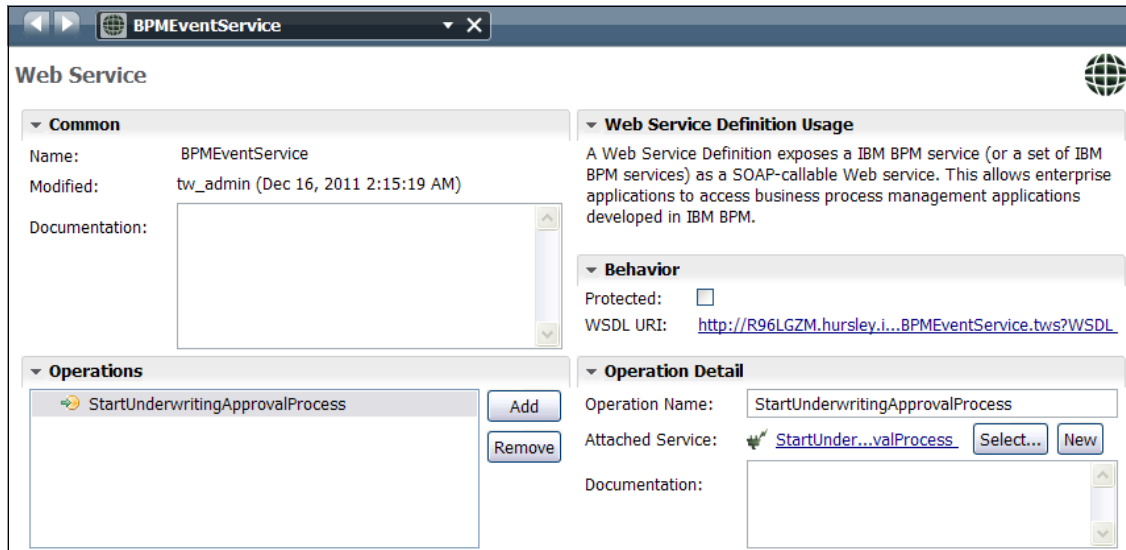


Figure 6-12 Exposing the UCA trigger through a web service

The WSDL URI indicates where the WSDL describing that service can be obtained from. You need the URI to import the WSDL into Event Designer to establish a connection to Process Server.

From the Event Designer project where you are defining the event processing, you can import this web service and create an action connector by using the Import WSDL wizard (Figure 6-13). This action is the same way we connected from events to the decision service in Chapter 5, “Detect-Decide-Respond pattern design” on page 59.

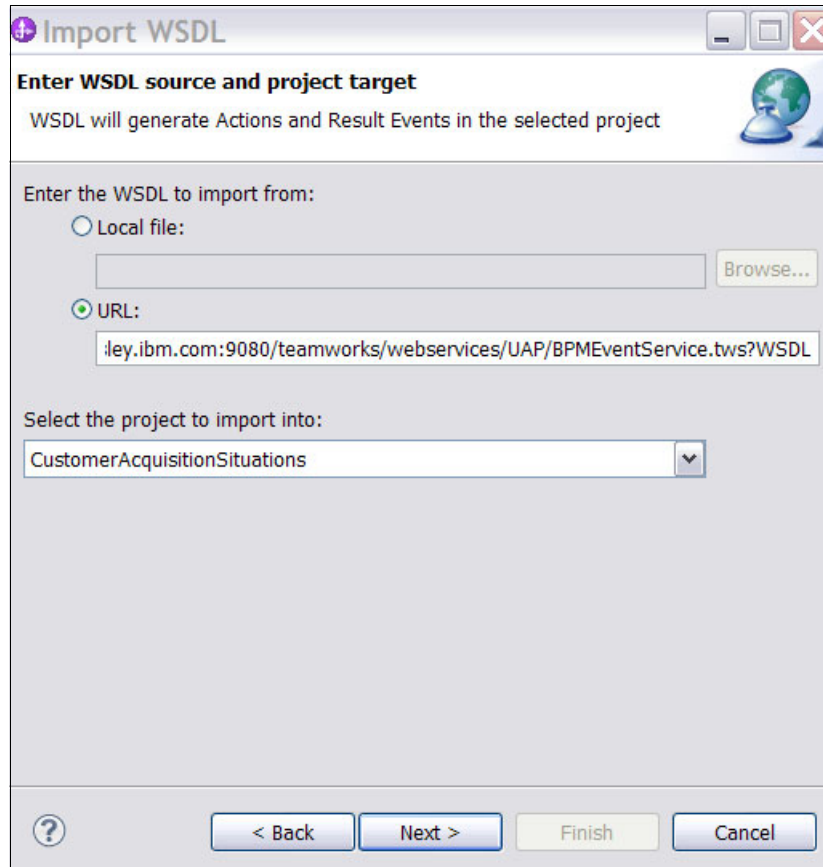


Figure 6-13 Use the Event Designer WSDL Import wizard to create an action from the BPM web service

When the wizard completes, an action and return event are created in the Event project (Figure 6-14).

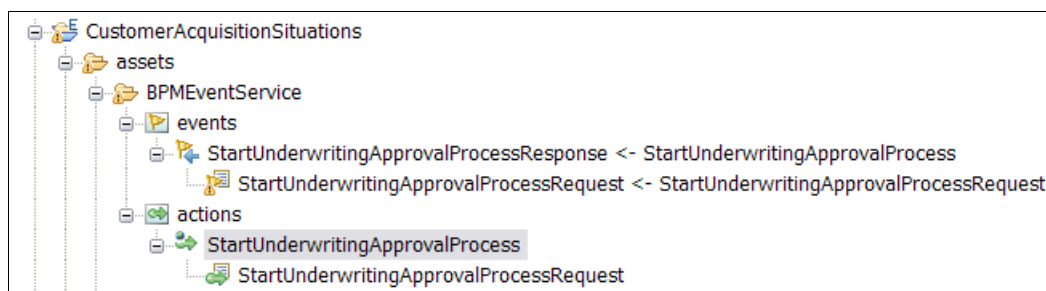


Figure 6-14 Action and return event created by and imported into the BPM web service

The action verbalization can be modified so that it is easy for business users to recognize it when writing the event rules (Figure 6-15). This verbalization also shows the action object that contains the event payload.

Action: StartUnderwritingApprovalProcess

▼ General Information

Name: StartUnderwritingApprovalProcess

► Documentation

▼ Verbalization

Navigation:

Phrase: Start Underwriting Approval Process

Action:

Template: Start Underwriting Approval Process

[Reset](#) the verbalization to default.

[Remove](#) the verbalization.

▼ Action Objects

The fields in the action are grouped into action objects, which can be optionally reused by other actions.

StartUnderwritingApprovalProcessRequest

[Add...](#)

[Remove](#)

[Edit](#)

☐ Automatically generate multiple actions if the number of occurrences of an Action Object would exceed its maximum.

Figure 6-15 Action verbalization and event payload

The Connector tab shows the details of the target web service that is invoked by the action. This invocation includes the BPM web service endpoint, which might need to be changed for particular deployments. Security settings can also be configured here (Figure 6-16).

The screenshot shows a web application window titled '*StartUnderwritingApprovalProcess'. The main content area is titled 'Connector'. It is divided into two sections: 'Connector Type' and 'SOAP Connector Settings'. In the 'Connector Type' section, 'Connector type' is set to 'SOAP' and 'Action packet version' is set to '2.2'. Under 'Subscription type', the 'Express' radio button is selected. The 'SOAP Connector Settings' section prompts the user to 'Enter the connection details for the server:'. It includes an 'Endpoint address' field with the value 'http://localhost:9080/teamworks/webservices/UAP/BPMEventService.tws', an 'Authentication type' dropdown set to 'None', and an empty 'Timeout (seconds)' field. At the bottom, there is a tabbed interface with 'Overview' and 'Connector' tabs, where 'Connector' is the active tab.

Connector Type	
Connector type:	SOAP
Action packet version:	2.2
Subscription type:	<input checked="" type="radio"/> Express <input type="radio"/> Reliable

SOAP Connector Settings	
Enter the connection details for the server:	
Endpoint address:	http://localhost:9080/teamworks/webservices/UAP/BPMEventService.tws
Authentication type:	None
Timeout (seconds):	

Figure 6-16 Action connector configuration

The wizard also defines the action object that represents the event payload that is sent to trigger the BPM process. The fields in this object need to be populated from Business Objects in the situation context from which the action is invoked. This action allows the fields to be completed by any incoming event that causes the action to be invoked. This mapping is shown in Figure 6-17.

Action object: StartUnderwritingApprovalProcessRequest

General Information
Name: StartUnderwritingApprovalProcessRequest

Documentation
Enter a description of this asset:
(string)
whiteSpace=preserve

Fields
The fields in the action are grouped into action objects, which can be optionally reused by other actions.

Name	Type	Definition Type	Definition Va...
impl:QuoteID	String	Field	QuoteID
impl:FullName	String	Field	FullName
impl:VIN	String	Field	VIN
impl:Price	Real	Field	Price
impl:Message	String	Field	Message
impl>Status	String	Constant value	Manual

Buttons: Add..., Remove, Edit, Move Up, Move Down

Number of Occurrences
Specify the number of instances of this action should appear in the action.

☐ Unlimited

1 Minimum number of occurrence

1 Maximum number of occurrence

Overview Field

Figure 6-17 Action payload field mapping

When the action is configured in this way, event rules can now be easily written by business users that start the Underwriting Approval Process in response to an event. In the WODM Insurance Underwriting Approval scenario, this response comes from the Manual Approval event raised as part of the event processing. This response is shown in Figure 6-18.

The screenshot shows the 'Event: ManualApproval' configuration window. It has two tabs: 'LaunchUnderwritingApproval' and 'ManualApproval'. The 'ManualApproval' tab is active. The window is divided into two main sections: 'General Information' and 'Documentation'. Under 'General Information', the 'Name' is 'ManualApproval'. Under 'Documentation', there is a 'Verbalization' section with a text field containing 'Request Manual Approval'. Below this field are two links: 'Reset' (with a key icon) and 'Remove' (with a red X icon). At the bottom, there is an 'Event Objects' section with a text area containing 'QuoteEvent' and three buttons: 'Add...', 'Remove', and 'Edit'. The bottom of the window has two tabs: 'Overview' and 'Connector'.

Figure 6-18 WebSphere ODM ManualApproval event verbalization

The event rule that business users need to write to start an underwriting approval process in response to a situation then becomes simple (Figure 6-19).

The screenshot shows the 'Event rule: LaunchUnderwritingApproval' configuration window. It has two tabs: 'LaunchUnderwritingApproval' and 'ManualApproval'. The 'LaunchUnderwritingApproval' tab is active. The window is divided into two main sections: 'General Information' and 'Documentation'. Under 'General Information', the 'Name' is 'LaunchUnderwritingApproval'. Under 'Documentation', there is an 'Event Rule Context and Event' section with two text fields: 'Context ID' (containing 'the applicant name of the quote') and 'Event' (containing 'Request Manual Approval'). Each field has a 'Change Context...' or 'Change Event...' button and a 'Remove' button. Below this is a 'Content' section with a text area containing 'then Start Underwriting Approval Process ;'. The bottom of the window has two tabs: 'Overview' and 'Connector'.

Figure 6-19 Event rule to start the approval process

6.6 Providing the information for the manual review

The next step is to use the QuoteEvent fields to retrieve the full Quote Request details from the Decision Warehouse. This retrieval depends on the implementation of the database implementation and must be designed to suit the solution.

A DecisionWarehouseAccess service is defined to provide this capability and it is configured as shown in Figure 6-20.

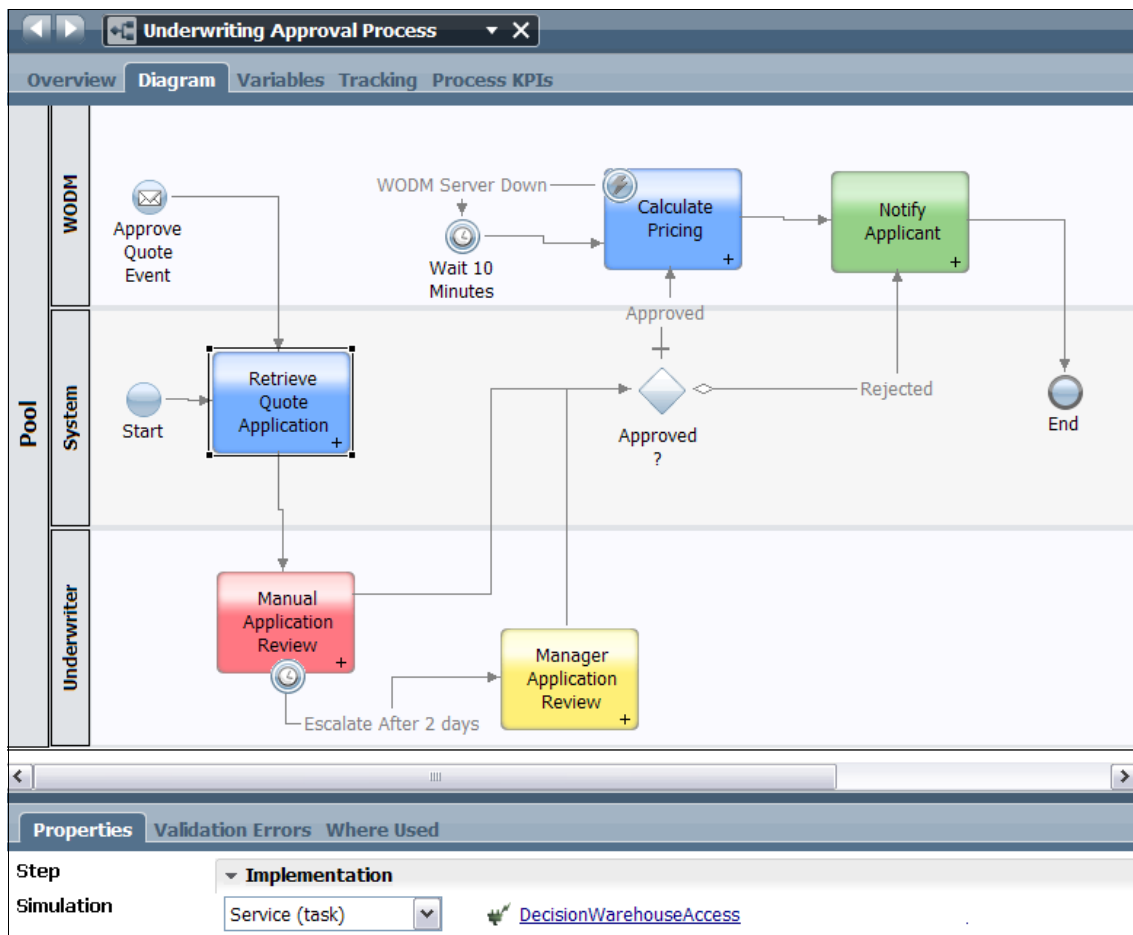


Figure 6-20 Invoking a service to retrieve the quote request details from the Decision Warehouse

After the full Quote Request is retrieved from the decision warehouse and mapped into process variables, the Coach for the Manual Application Review can be invoked by passing in the quote Event variable with the status and messages. You also pass in the request details (Figure 6-21).

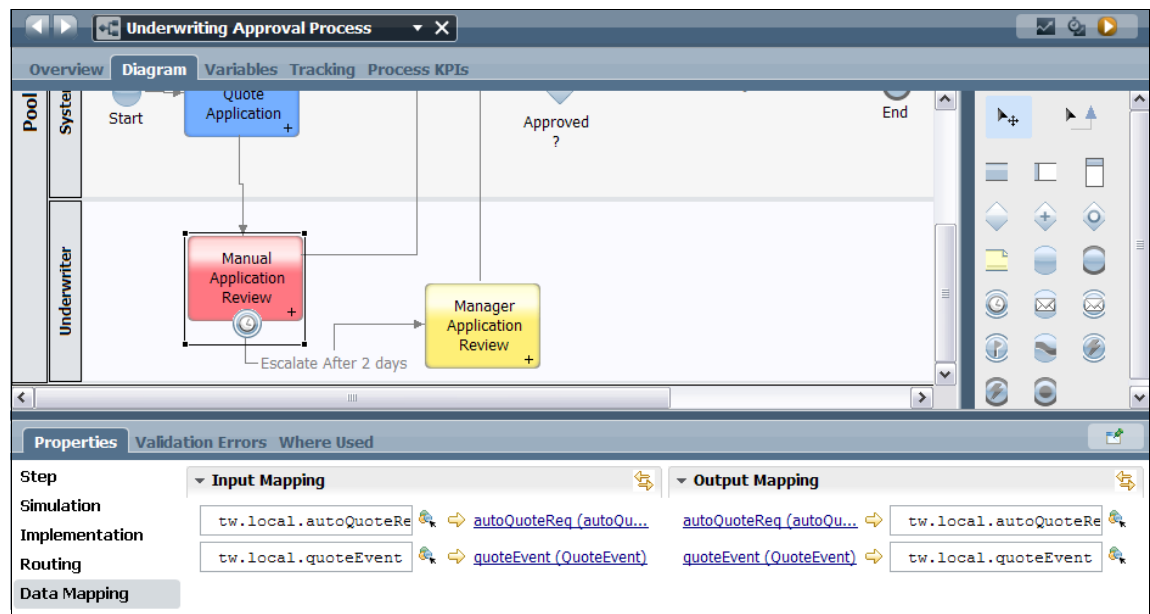


Figure 6-21 Invoking a coach to allow the underwriter to approve the quote

The coach then displays the information to the underwriter, allowing the underwriter to approve or reject the quote (Figure 6-22).

WODM

Insurance Demo

Manual Application Review

Summary of Drivers

First Name	Last Name	Age	Gender	State:
Cindy	Davis	22	Female	CA

Summary of Vehicles

Make	Model	Year
Brand_A	A1	2,006

Coverages

Coverage Type	Max Limit	Min Limit	Deductible
Liability	30,000	15,000	0
Uninsured and Underinsured Motorist	30,000	15,000	0
Comprehensive	0	0	1,000
Collision	0	0	1,000
Mechanical Breakdown Insurance	0	0	0

Manual Review

Manual Review Reason: Females between 16 and 25 in CA require manual review.

Underwriter Decision:
☒ Approve Application
☐ Reject Application

Submit

Figure 6-22 Possible underwriter approval window provided by the coach

Although this window shows only simple information currently, the decision warehouse also has access to the rules that fired, so other metrics associated with why the quote was selected for manual review could also be displayed.

The response from the review is stored in the quote event process variable and can then be used in the Approved ? gateway shown in Figure 6-23.

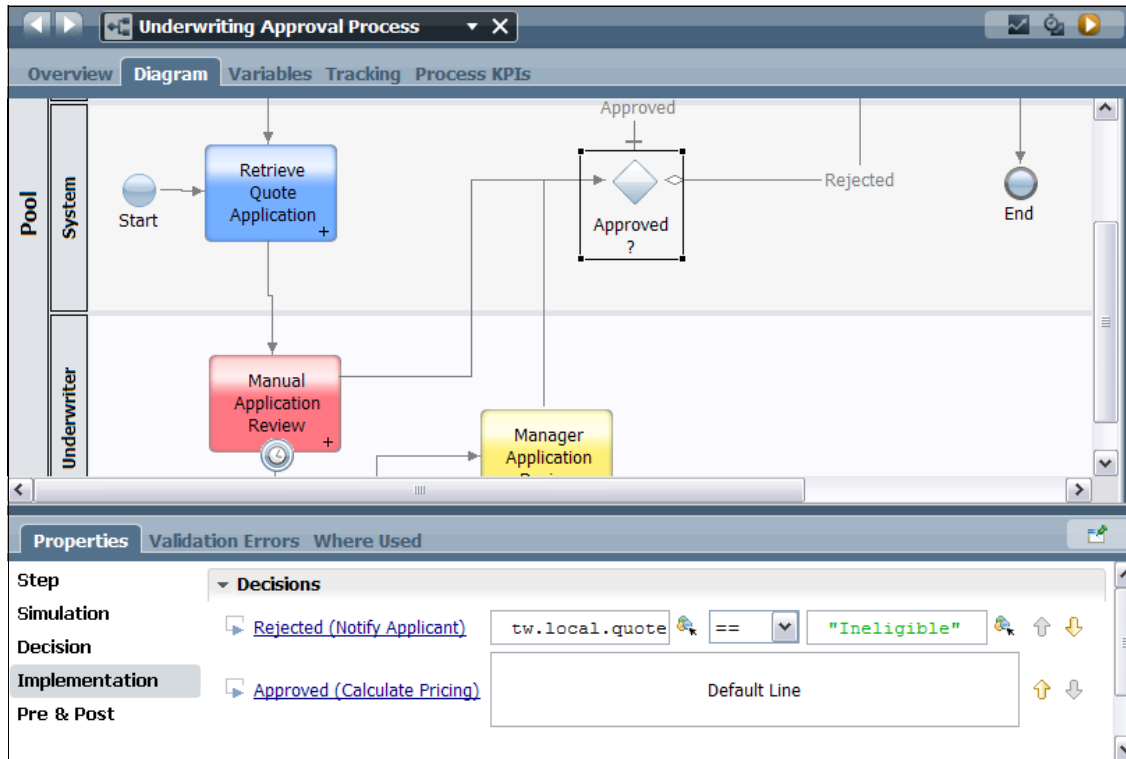


Figure 6-23 Reacting to a decision or approval using a gateway

Now that the Quote Request is approved, the Pricing decision service can be invoked from the process using the BPM decision developed earlier. Because we already retrieved the Quote Request, there is no need to retrieve it again, and the process variable set up earlier is simply mapped to the input parameter (Figure 6-24).

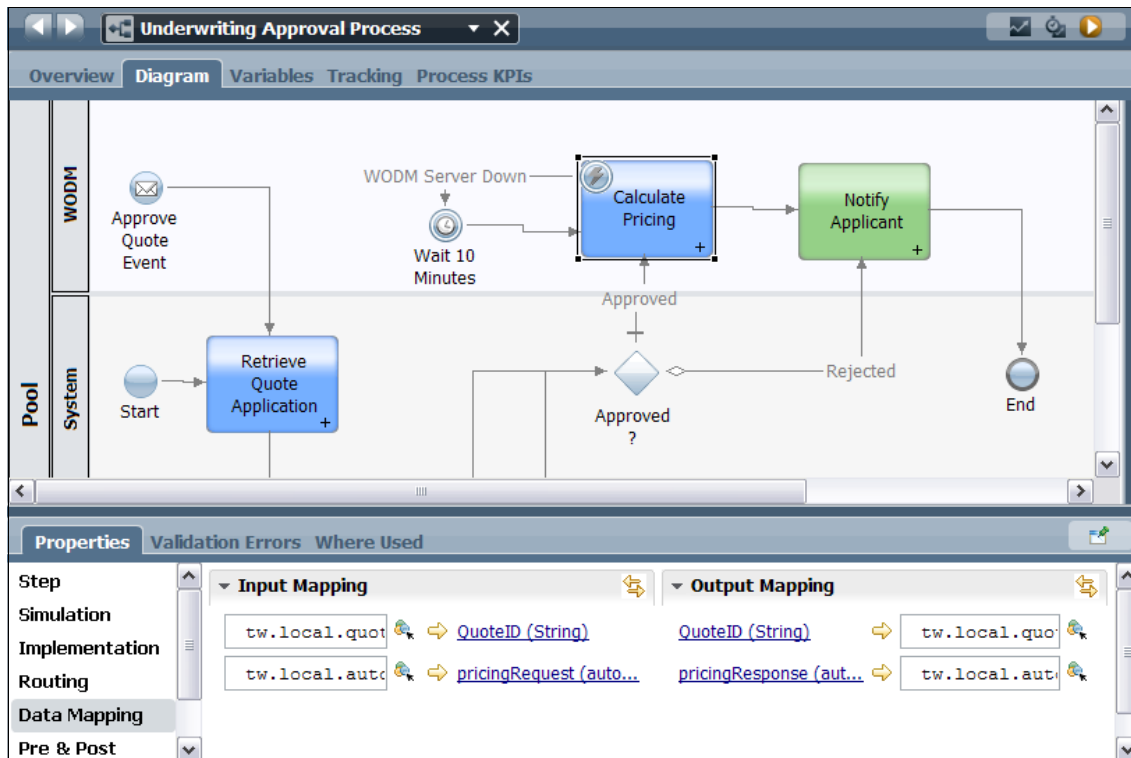


Figure 6-24 Invoking a decision from a process

The request is passed to the decision and the pricing rules are applied. The Pricing decision can be stored in the decision warehouse and is passed back to the process, allowing it to be stored elsewhere for future reference or displayed in a coach (Figure 6-25).

WODM Insurance Demo

Application Approved

Congratulations - your auto insurance application has been approved!

Based on the information in your form, we are pleased to offer you the following rate quote:

Total Quote Price: \$357.04 per six month period

Coverage Type	Coverage Price	Adjustments Text
Liability	\$105.00	Base Premium - Liability: \$105.00
Uninsured and Underinsured Motorist	\$15.60	Base Premium - Uninsured and Underinsured Motorist: \$15.60
Comprehensive	\$49.44	Base Premium - Comprehensive: \$48.00 Surcharge for Sedan cars: +3%
Collision	\$157.00	Base Premium - Collision: \$150.00 Anti-lock Brakes Discount: -2% Driver Profile Surcharge: +\$10
Mechanical Breakdown Insurance	\$30.00	Base Premium - Mechanical Breakdown Insurance: \$30.00

OK

Figure 6-25 Using coaches to display the results of a decision in a process

The final stage of the process is to send an event to notify the applicant. Section 6.7, “Raising an event from within a BPM process” on page 105 describes this process.

6.7 Raising an event from within a BPM process

Process Designer provides many integration features that allow a process to invoke an external application, but using web services is one of the quickest and easiest to implement this process. To send an event to WebSphere ODM, you must provide the WebSphere ODM web service that you must call.

In Event Designer, you created a QuoteApproved event that uses the same event structure and objects as existing events. This event is exposed as a web service using the SOAP event connector.

If this WSDL is downloaded and viewed in a WSDL editor, the structure of the service is a one-way call containing the event payload (Figure 6-26).

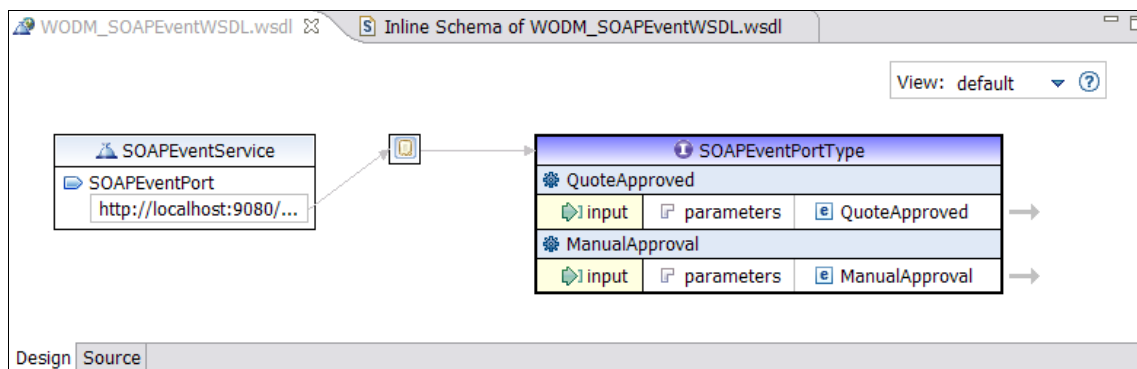


Figure 6-26 Web service signature provided by exposing events using a SOAP connector

The payload and structure of the events are shown in the inline schema provided in the WSDL (Figure 6-27).

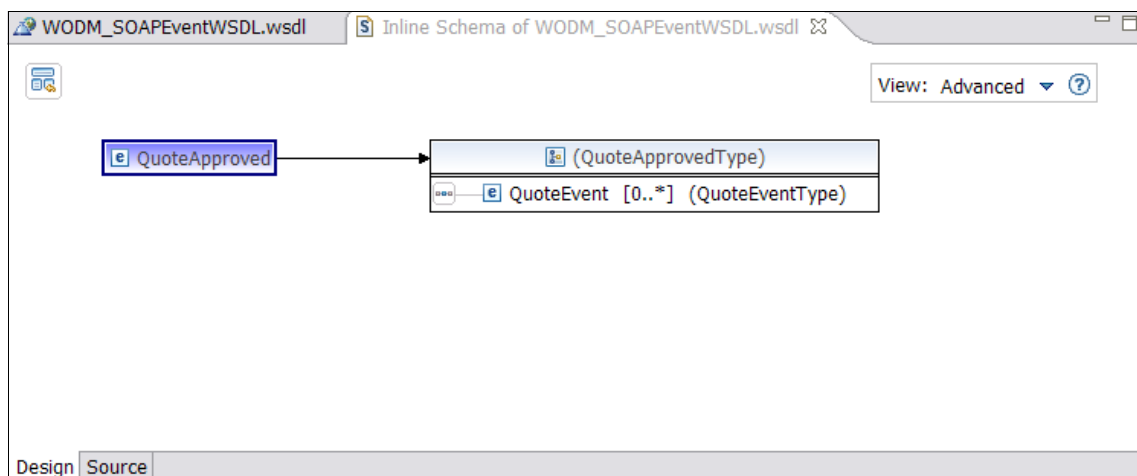


Figure 6-27 Web service signature showing repeated event payload

The detailed fields now match those fields in the original definition (Figure 6-28).

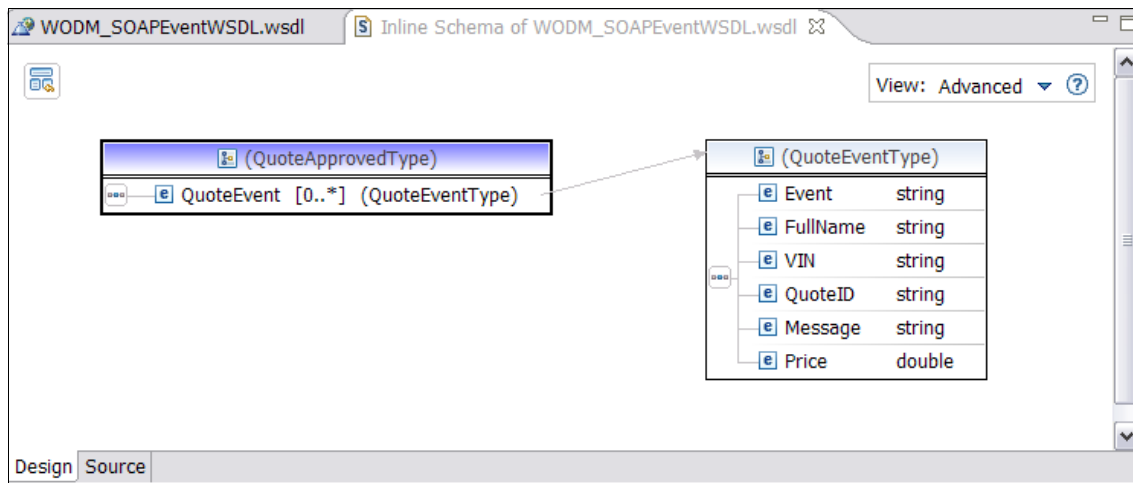


Figure 6-28 Structure of individual event payload

This web service URL can then be used in Process Designer to create a web service call that invokes the event run time (Figure 6-29).

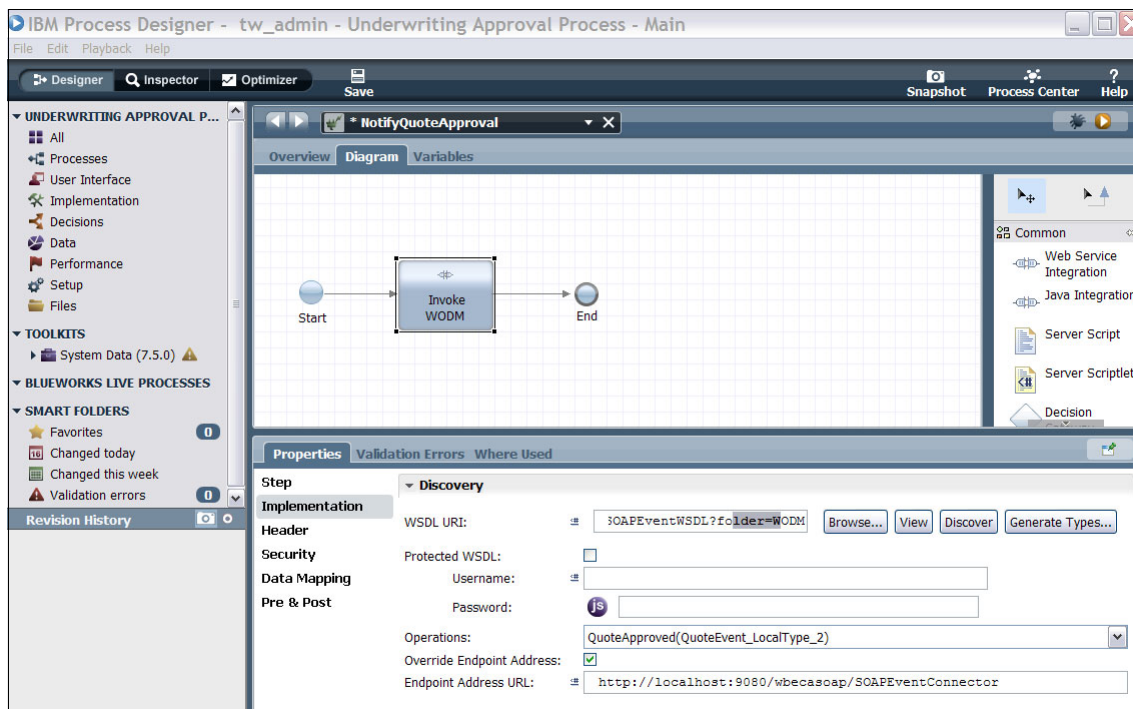


Figure 6-29 Creating the service to invoke the WebSphere ODM event web service

Create the Process Designer decision signature and map the fields to the web service call (in Figure 6-30). In this case, the status provided by the underwriter is applied to the event field, allowing the event process to determine whether the quote was approved or rejected.

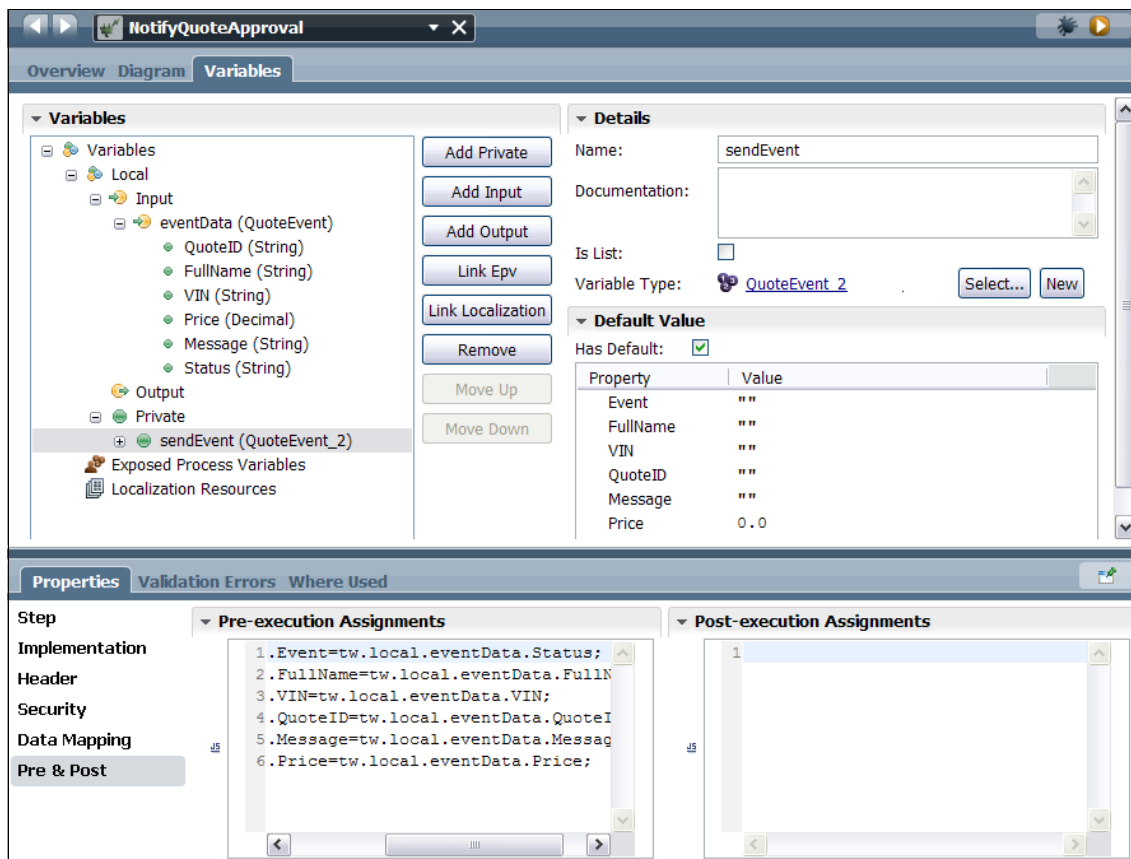


Figure 6-30 Mapping the service signature fields to the event web service

The service is complete and can be integrated into the process in the Notify Applicant activity, and the eventData process variable can be used to provide the information needed to send the data (Figure 6-31).

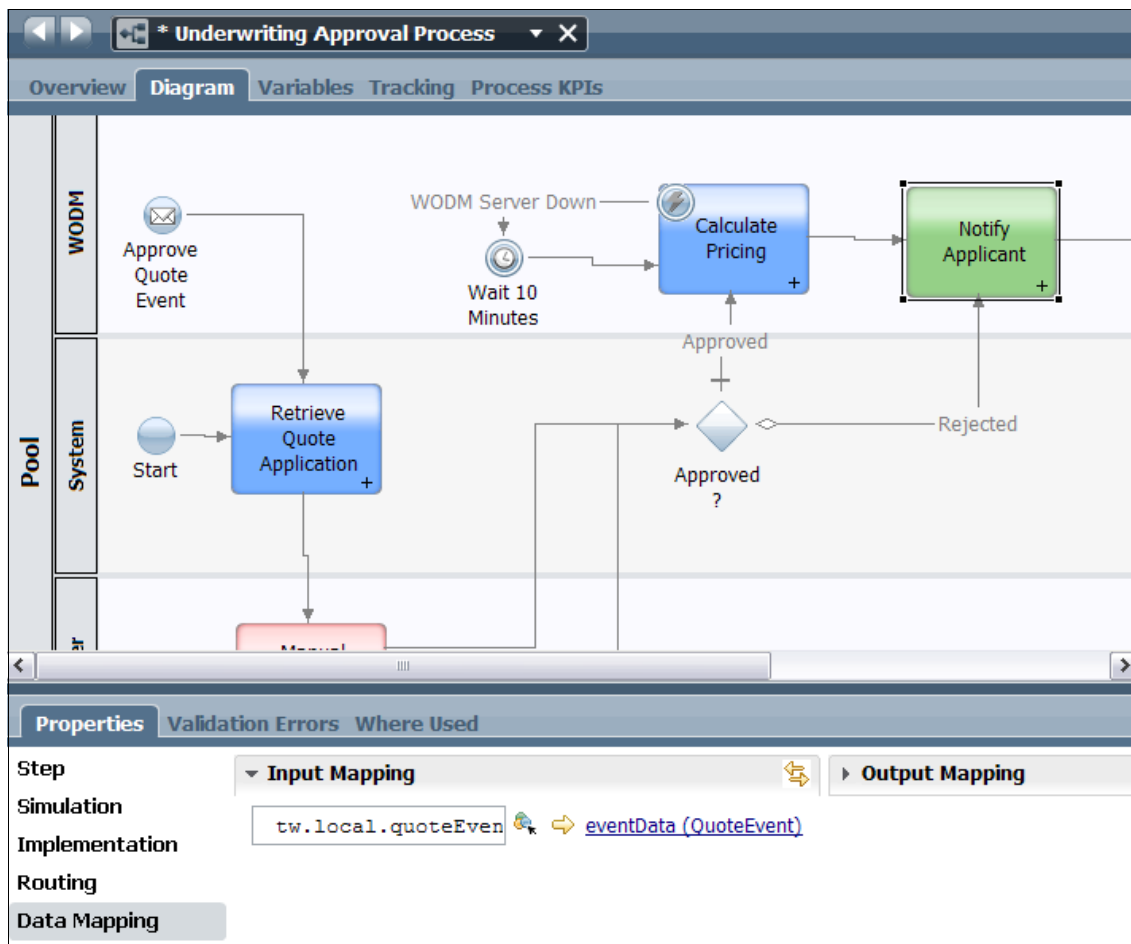


Figure 6-31 Integrating NotifyQuoteApproval into the process

The updated QuoteID contained within the event data is used by other systems to retrieve the actual pricing breakdown and continue the rest of the solution processing. The Underwriting Approval Process is now complete and terminates.



Managing decision changes across rules, events, and processes

This chapter describes the key governance scenario for changing and deploying new rules for both events and rules. The default lifecycle is used and the necessary simulations and governance checkpoints are summarized, together with how the tools support consistency of the changed rules when deployed.

7.1 Support for managing and governing change in WebSphere ODM

The WebSphere ODM platform provides a set of mechanisms and tools to allow change and support the enforcement of governance at the level of the individual decision artifacts and at the overall decision level itself. The following sections present these concepts, features, and tools.

7.1.1 Access control and permission mechanisms

The ownership of rule and event projects is often distributed among different groups, potentially belonging to different lines of business in the enterprise. The business people responsible for determining the eligibility of a quote, involving state regulatory considerations, are most likely not the actuaries of the company who determine the pricing policies. Also, being assets that implement critical company business policy and strategies, rules and events are often subject to confidentiality considerations. Therefore, access to artifacts from a project should be clearly established and tightly controlled by the platform. Similarly, access to operations available to a user group should be controlled (in this case, groups reflect the concept of role on the platform).

By partitioning the users of Decision Center into different user groups and associating permissions and enforcing access control through those groups in Decision Center, we guarantee that only the designated groups are authorized to access certain artifacts and perform certain operations:

- ▶ Establishing branch security: Security for rules and event projects can be specified at the project branch level by listing the user groups that are allowed to access the project branch.
- ▶ Establishing artifacts permissions: After branch security is established, access permission can be defined at the artifact level, and you can specify view, create, update and delete permissions.

7.1.2 Rule and event status

A lifecycle is associated with rules and events in Decision Center to separate the responsibilities for the different activities that must be performed on an artifact. In particular, it allows enforcing constraints such as:

- ▶ The person who validates an artifact is not the same person who initially authored or updated it.
- ▶ The person who moves an artifact to a deployable state has the authority to make that decision.
- ▶ When an artifact is in the deployable state, its definition and other characteristics should not be modified by anyone.

The lifecycle definition and enforcement applies similarly to rule and event project artifacts. It is defined using the set of values associated with the Status property of the artifacts (the default set is *new*, *defined*, *validated*, *rejected*, and *deployable*) and properties associated to the user groups. Using the information that a user belongs to a certain group, and that an artifact has a status value, we can decide:

- ▶ Which transitions of status this user is authorized to perform.
- ▶ Which fine-grained action can be applied to the artifact and its properties.

The status of the artifacts used in the rule lifecycle is also used in conjunction with the extraction step of decision deployment to support the filtering of the artifacts from the repository that should be deployed to the runtime engine. The extraction process in Decision Center is associated with a query that decides which artifact should be deployed.

This process allows filtering out artifacts that are not completed by their authors and have a status of new, and therefore are not considered for extraction. For deployment outside of the development environment (for example, QA or Production), only the artifacts with a status of deployable are included for deployment.

7.1.3 Decision test suites

Part of governing the change in decisions is performing functional testing of the decision to ensure that it conforms to the intended business policy change requirement and performing regression testing to ensure that the existing rules are still valid.

These tests are performed before the updated decision is deployed outside of the development environment. The test suites defined using the Decision Validation Service (DVS) allow this type of activity to be performed by a business person using spreadsheets to define the test scenarios.

7.1.4 Decision simulations

Simulations allow running a set of test scenarios for a decision and providing aggregated views of the decision response's characteristics, for example, the percentage of eligible versus ineligible quotes (Figure 7-1).

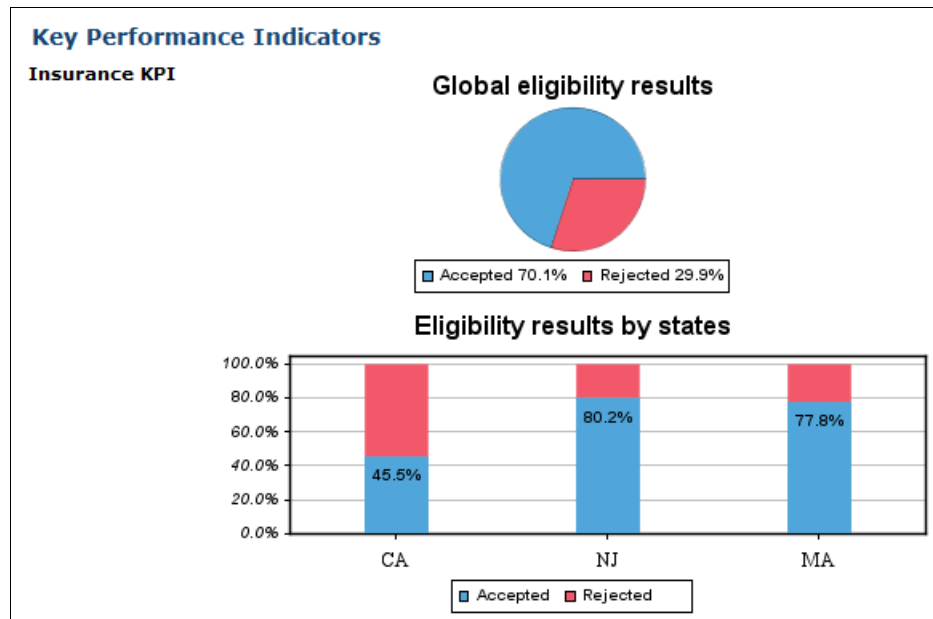


Figure 7-1 Outcome from simulation execution

Simulations are a valuable tool in the change process. They help assess the impact of rule changes to the decision and help you decide whether the effected changes meet the requirement results. Although test suites assess the correctness of the decision, simulations are design to assess the quality of the decision. They are run on a large number sample of scenarios, using sources from a production environment, instead of in a spreadsheet.

The results of simulations can be compared side by side, thus providing a champion-challenger view of the decision.

7.1.5 Event testing and simulation

Testing and simulating event processing can be performed using a number of Business Space widgets.

A tester widget allows events to be injected into the selected event run time. The effect of those events and the ones received from event connectors can be viewed in the Context Data tab of the tester widget (Figure 7-2).

The screenshot shows the WebSphere Operational Decision Management interface. The top navigation bar includes links for Home, Go to Spaces, Manage Spaces, and Actions. Below this is a blue header with the text 'WebSphere Operational Decision Management'. A secondary navigation bar contains tabs for Decision Authoring, Event Testing (selected), Event Capture and Replay, and Reporting. The main content area is titled 'Event Tester' and includes a 'Restart Testing' button. Below this are tabs for Send an Event, Filters, Actions, Context Data (selected), and Delayed Rules and Actions. There are also 'Refresh' and 'Delete Selected Items' buttons. The 'Context Data' tab displays a table with the following data:

	Context Id	Name	Type	Count
<input type="checkbox"/>	Cindy Davis	WebQuoteRejected	Event	2
<input type="checkbox"/>	Cindy Davis	DeclineWebQuote	Event	1
<input type="checkbox"/>	Cindy Davis	WebQuoteOffered	Event	5
<input type="checkbox"/>	Cindy Davis	CustomerAcquisitionPromotionsResponse	Event	1
<input type="checkbox"/>	Cindy Davis	AcceptWebQuote	Event	1
<input type="checkbox"/>	VIN-0	WebQuoteRejected	Event	2
<input type="checkbox"/>	VIN-0	WebQuoteOffered	Event	5
<input type="checkbox"/>	Cindy Davis	OfferPromotion	Action	1
<input type="checkbox"/>	Cindy Davis	CustomerAcquisitionPromotions	Action	1
<input type="checkbox"/>	Cindy Davis	UserConsoleNotification	Action	1
<input type="checkbox"/>	Cindy Davis	ReferToCallCenter	Action	1

Figure 7-2 Tester widget Context Data tab

Other tabs show information about filter conditions and the status of rules that are not evaluated yet. Information about the actions, together with the action data objects sent, is available in the Actions tab (Figure 7-3).

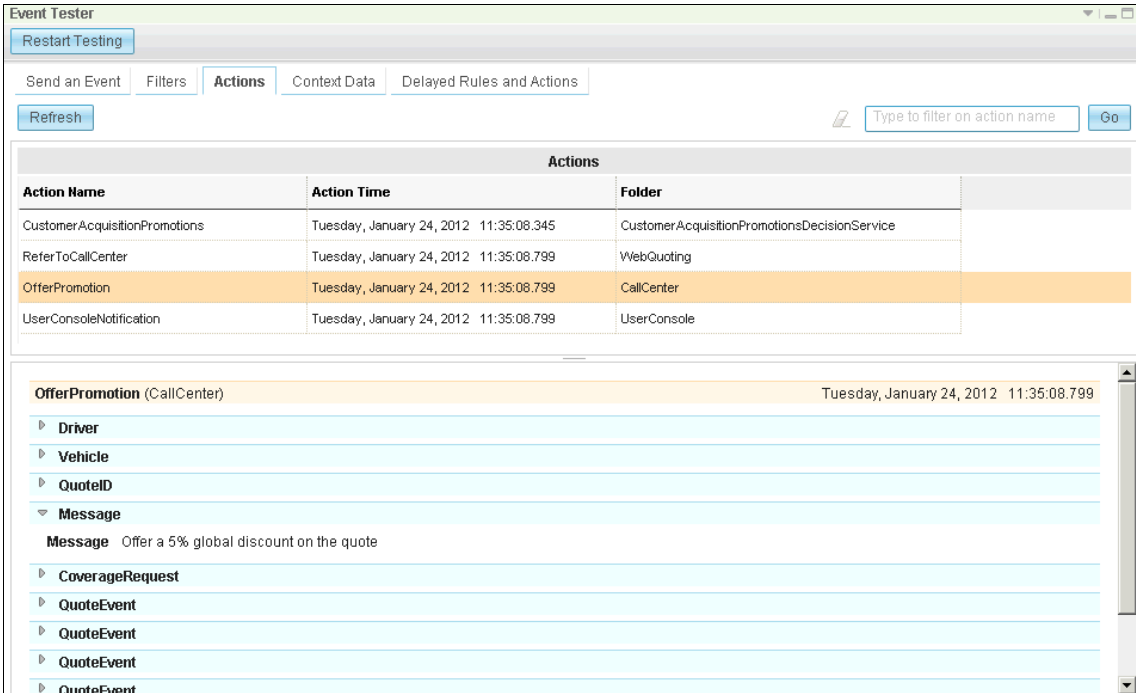


Figure 7-3 Tester widget Actions tab

These testing facilities provide step by step debugging of the event rules, allowing the Situation Detection and Respond characteristics to be fully understood.

It is also possible to record sequences of incoming events and then play these events back through the event run time. This feature provides a facility similar to the rules-based simulation, allowing the effect of changing the rules to be observed in the tester widget. This Event Capture and Replay widget is shown in Figure 7-4.

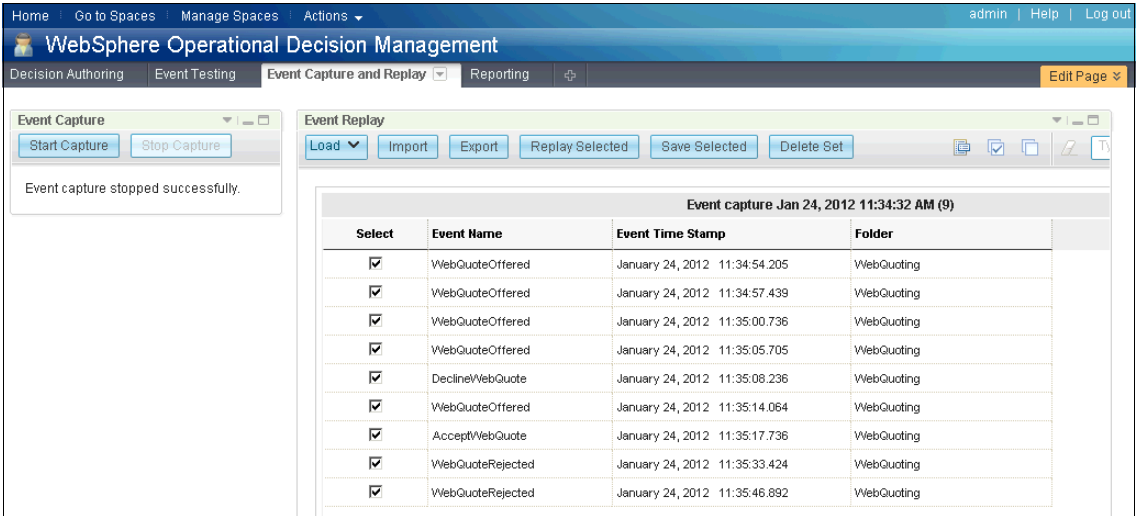


Figure 7-4 Event Capture and Replay Widget

The event processing capabilities of WebSphere ODM also support event charts, which provide many different means of illustrating the KPIs and characteristics of events and actions. An example that shows the proportion of incoming events for the previous scenario is shown in Figure 7-5.

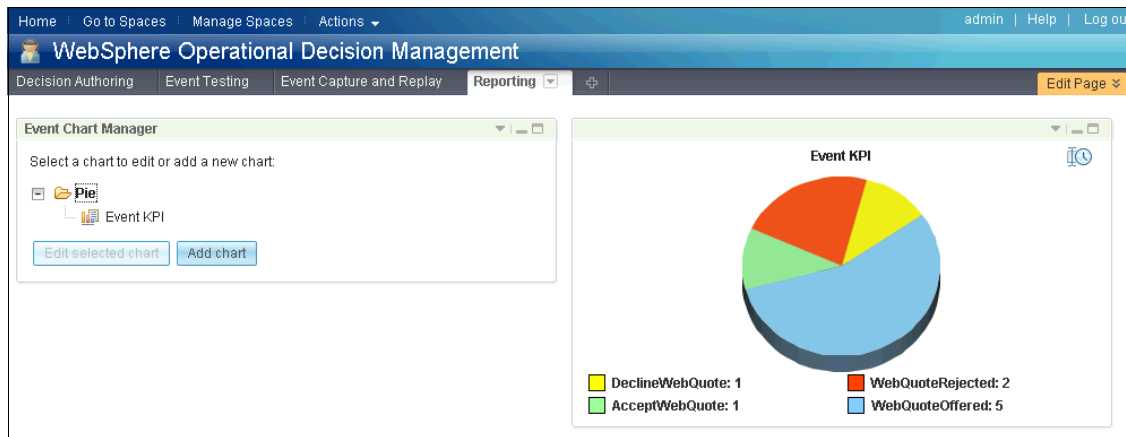


Figure 7-5 Event chart and KPI dashboard widget window

These widgets mean that there are mechanisms available to the Line of Business User to test and validate their event rules. Although these facilities are not as mature as the simulation and testing facilities for business rules, the same activities can be undertaken, and the decision lifecycle and governance steps can be considered identical.

7.1.6 Deployment baselines

Deployment baselines are created when a RuleApp is deployed to a rule execution server or an event project is deployed to an event runtime server (Figure 7-6 and Figure 7-7).

The 'Deployment Baseline' dialog box is shown. It has a title bar and a main area with a checked checkbox labeled 'Create a baseline for this deployment'. To the right of the checkbox is a text field containing 'Eligibility-v2.43'. Below the checkbox are three buttons: 'Cancel', 'Previous', and 'Next'.

Figure 7-6 Creating a deployment baseline for a RuleApp

The 'Deploy Project' dialog box is shown. It has a title bar and a main area. The 'Query' field is set to '<none>'. The 'Server*' field is set to 'Insdemo Event Runtime'. Below these fields is a checked checkbox labeled 'Create a baseline for this deployment'. To the right of the checkbox is a text field containing '2011-12-13 12:53:39 EST'. At the bottom are two buttons: 'OK' and 'Cancel'.

Figure 7-7 Creating a deployment baseline for an event project

For a RuleApp, one deployment baseline is created for each project that is associated with the rule sets that participate with the RuleApp, and for the projects that they reference. The deployment baseline provides a snapshot of the content of the project at the time the RuleApp or the event project is deployed.

Creating a deployment baseline is an essential step in a decision change management lifecycle, as it that allows capturing a state of the repository that can be reused through the successive redeployment of the decision through different environments. Creating a deployment baseline on the first deployment and then using the Redeploy command on the previously created baseline guarantees that the exact same RuleApp or event project content is deployed from one environment to the next.

7.1.7 Branch management

Project branches provide an easy way to manage the concurrent development of updates to a decision project. When the development of multiple sets of changes overlaps, but their planned deployment dates are different, one branch can be created for each change set and the changes can be developed and tested concurrently until they are ready to be deployed. Then the final changes can be merged with the main branch (trunk) and deployed from there.

Project branches are also used in conjunction with deployment baselines to manage bug fixes. The baseline is the reference point from which to create a bug fix branch in case the newly deployed decision is kicked back from future environments, such as QA.

When this situation is the case, the rule authors can request the creation of a project branch from the deployment baseline. This branch allows them to bring the needed changes to fix the defects and perform patch deployments from this branch (Figure 7-8).

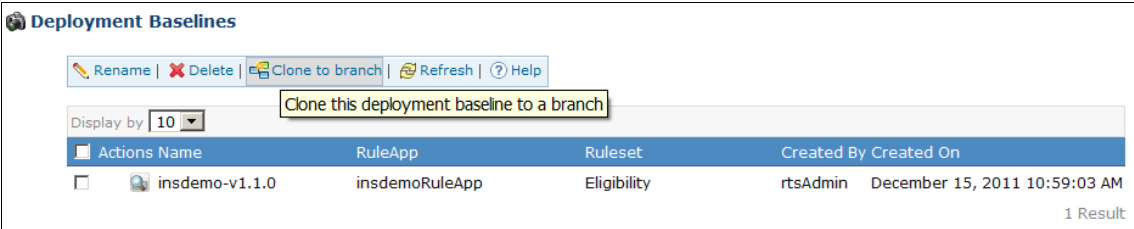


Figure 7-8 Cloning a deployment baseline to a branch for a bug fix

7.2 Rule lifecycle and governance

Rule lifecycle and governance is implemented through status management, access control, and permissions, as described in 7.1.1, “Access control and permission mechanisms” on page 110 and 7.1.2, “Rule and event status” on page 110. The WODM Insurance application defines, for example, a group for Regional Managers of the Pricing decision for New York state. Ryan is a member of this group.

1. When Ryan logs on to Decision Server, he sees only the projects to which his group has access because of the security defined on all the projects (Figure 7-9).

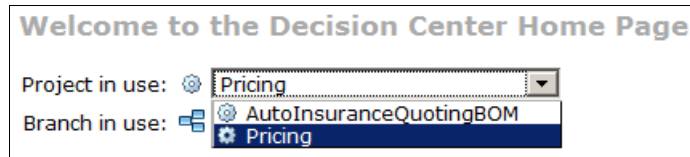


Figure 7-9 Project access control

2. Going further, within the Pricing rule project, Ryan can see all the common pricing rules, but only the state-specific rules for the state he manages because of the defined permissions on the rule artifacts (Figure 7-10).



Figure 7-10 Rule artifact access control

3. Ryan can create a rule in the Pricing project and move the status of the rule from New to Defined when he finishes authoring it. To modify an existing deployed rule, Ryan must have the rule status moved from Deployable to New by the Administrator because of the customized rule lifecycle (Figure 7-11).

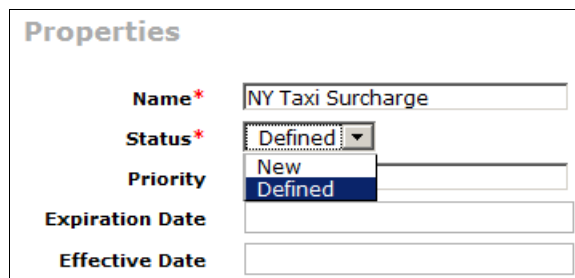


Figure 7-11 Rule Artifact Status

4. After Ryan moves a rule to the Defined status, it is validated by the Pricing Policy Manager, who assigns a status of either Validated or Rejected to the rule.
5. After the rule is approved by the Policy Manager, the Administrator can then move the rule to the Deployable, so that it is included in the rule set that is extracted for integration test and production.

The governance enforced by setting up access control, permission, and status management at the fine-grain level of the rules and events should be defined carefully so that it does not become a hindrance to the change process.

Although establishing rule and event governance allows you to enforce the segregation of responsibilities for the artifact's authoring task, it does not provide a vision of what should happen at the decision level to manage change.

7.3 Decision lifecycle and governance

In addition to artifact-level governance, a solution involving decision management should define the change lifecycle and governance at the level of the decision itself. The goal is to define who is responsible for what task at which moment in time and in which environment to carry out a business policy update, from its initial request by the Policy Manager to its deployment in the production environment, to the evaluation of the newly deployed business decision implementations. Decision change thus follows a Define, Deploy, Measure, and Update cycle.

7.3.1 Define phase

The requirements for the change in the decision come from the need to improve a KPI of the decision or to adapt it to a changed business context. An analysis activity determines the extent of the change needed to support the business policy update; the outcome should tell if the update involves:

- ▶ A change to the rules alone, an activity which can be performed from the Decision Center by the Rule Authors. This activity is the most common change activity that is applied to a rule or an event project. It is applied to adapt the decision to either changing regulations coming from inside the enterprise or by outside regulatory body (such as a government agency), or to a change in business conditions.
- ▶ A change to a technical artifact of the rule project (for example, change to the rule flow, or more frequently, a change to the BOM),
- ▶ A change outside of the rule project (for example, in the XOM) to accommodate, for example, additional information needed by the new rules. This change can also be a consequence of a change needed by the decision service, which may include a change of the service signature and implementation.

Depending on the outcome of the analysis, the set of roles and activities involved in the implementing the change are different and involve different tools and features from the WebSphere ODM platform.

Figure 7-12 shows the workflow for this Define phase for case 1 and 2.

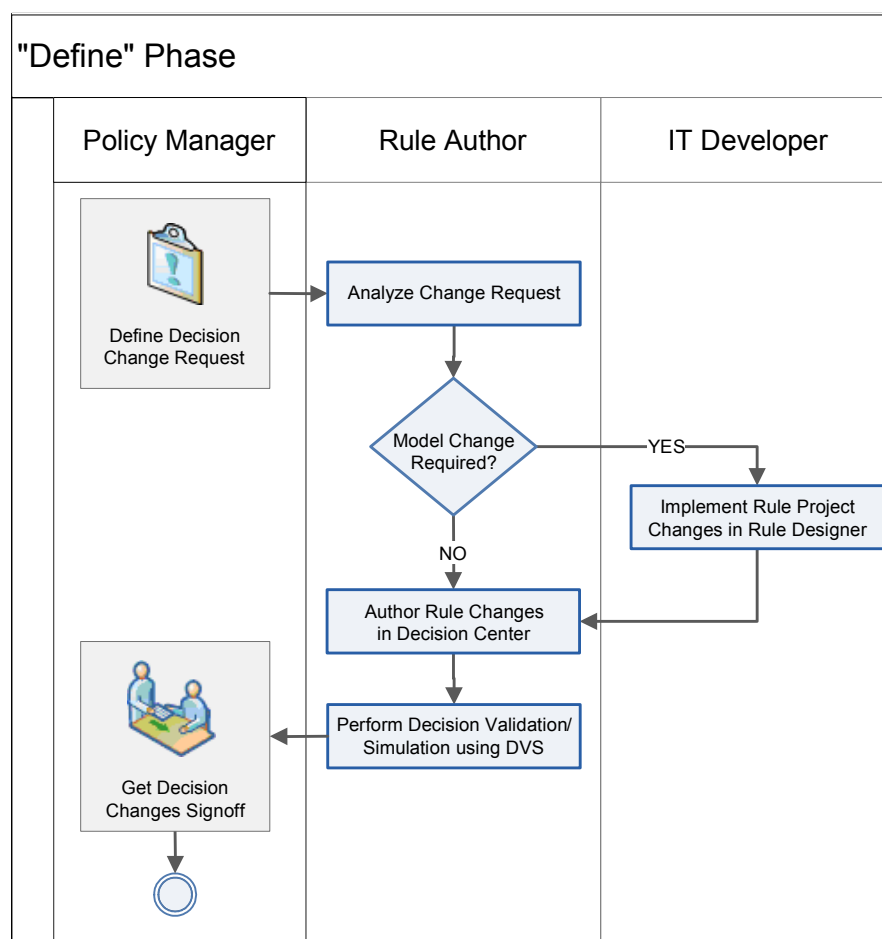


Figure 7-12 Example workflow for the Define phase

During this Define phase, correct rule change implementation is controlled using the permission, access control, and rule status management techniques described in 7.1.1, “Access control and permission mechanisms” on page 110 and 7.1.2, “Rule and event status” on page 110.

Validation of a set of rule changes is performed using the Decision Validation Services, which runs a set of positive and negative test scenarios that should be elaborated specifically for the rule change. In addition to this specific set, a set of existing DVS scenarios should be run as a regression test suite, to ensure that the newly updated rules did not introduce a defect in the existing ones.

If the change was driven by the required improvement of some KPIs, simulations are run with DVS to validate that the wanted improvement is reached.

After validation is completed, a sign-off on the decision changes should be obtained from the Policy Manager. Additionally, a development baseline can be created to record the state of the project at the end of the change implementation.

If multiple changes to the decision are implemented concurrently, they should be taking place on different project branches, as described in 7.1.7, “Branch management” on page 115. Branches allow concurrent development and testing of the rules, along with a deployment schedule that fits the agenda of the business.

7.3.2 Deploy phase

After the rule authoring teams complete rule authoring and validation of the different rule projects referenced by a RuleApp, the new RuleApp can be packaged and promoted to the next execution environment.

This next environment can be, for example, a QA or staging environment where integration tests are performed, or perhaps the production environment, if the rule change is considered innocuous or routine. Part of the governance process design work is to establish and differentiate which type of rule changes goes through an extensive promotion path through different environments and which ones can take a shortcut to the production environment.

However, before the RuleApp can be packaged and deployed, the Rule Authors performs a set of activities to make sure that the deployable artifacts are consistent and complete. These activities include the following ones:

- ▶ Use the rule analysis features of Decision Center (Check Project Consistency and Check Project Completeness on the Analyze tab) to ensure that the new rules do not introduce any semantic contradictions, gaps, or redundancy in the rules logic.
- ▶ Verify that all the rules in the projects have the correct status. The rule set extractors that are used to create a rule set deployed outside of the development environment usually collect all the rules that have a *Deployable* status. You must ensure that there are no rules remaining in the project that still have a status different from *Deployable*, or if there are, there should be a valid reason for it. In particular, rules with a *Validated* status usually are used during the functional testing of the rule set in the development environment, and not deploying these rules might change the behavior of the rule set.

The initial promotion of the updated decision is accompanied by the creation of a deployment baseline that allows redeployment of the same runtime artifact through all the successive execution environments, up to production.

The Define and Deploy phases can be supported by a tracking system that allows *both* the business and the IT stakeholders to follow the change request from its inception to its deployment.

7.3.3 Monitor and Measure phases

After the decision is deployed to the production environment, the impact of the change on the business needs to be measured through the decision's KPIs to weight its actual effectiveness. Although simulations are performed during the Define phase of the change, an actual measurement comes from monitoring the events from the production environment. If parts of the decision are derived from an analytics model, monitoring the decision result allows you to validate the model.

Some decisions, such as validation, are not associated with qualitative KPIs; the outcome of the decision is either correct or not. An example is when all invalid cases are properly recognized and associated with the correct justifications. In this case, the Measure phase does not yield any specific activity.

Also, in the context of a complex business process involving a series of decisions, the measure of improvement from the updated decision should be evaluated from the point of view of the overall business process. Look at the business process KPIs, which provide an aggregated view of the individual decisions KPIs.

7.3.4 Update phase

Based on the measurements from the previous phase, a new set of KPI goals or analytic models can be prepared for the next version of the decision. If the decision is not driven by a qualitative KPI, the update might be driven by changing regulatory constraints or the need to expand the footprint of the decision to new business areas.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

Other publications

This publication is relevant as a further information source:

- Boyer, et al., *Agile Business Rule Development: Process, Architecture, and JRules Examples*, Springer-Verlag New York, LLC, 2011, ISBN 3642190405

Online resources

This website is also relevant as a further information source:

- Technology connectors

<http://publib.boulder.ibm.com/infocenter/dmanager/v7r5/topic/com.ibm.wbe.appdev.doc/doc/whataretechnologyconnectors.html>

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



Making Better Decisions Using IBM WebSphere Operational Decision Management



Business rules and events in solution applications and processes

Decision management is emerging as an important capability for delivering agile business solutions. Decision management is not a solution in its own right, but must be integrated into the solutions or business processes that it supports.

Decision management lifecycle and governance guidelines

In this IBM Redpapers publication, we describe the recommended best practices and integration concepts that use the business events, business rules, and other capabilities of IBM WebSphere Operational Decision Management V7.5 (WebSphere ODM) to provide better decision making in those solutions and business processes.

Best practices for automating decisions

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks